PRNGCL: OpenCL Library of Pseudo-Random Number Generators for Monte Carlo Simulations

Vadim Demchik
vadimdi@yahoo.com

http://hgpu.org/

Dnipropetrovsk National University
Dnipropetrovsk, Ukraine

GTC’14

March 26, 2014
Motivation
Pseudo-random number generator (PRNG) – keystone of Monte Carlo (MC) algorithms

General requirements for PRNG to be used in MC simulations:

- good statistical properties (correct distribution, independence of outputs, absence of correlations) and strong theoretical background
- tested in corresponding field of research
- long period
- high performance
There are many different PRNGs (different algorithms, performances, scope, etc.)

**MAIN REASON:** there is no “ideal” PRNG

**MOST FREQUENT QUESTION:** which PRNG should be used for MC?

**ANSWER:**
There are many different PRNGs (different algorithms, performances, scope, etc.)

MAIN REASON: there is no “ideal” PRNG

MOST FREQUENT QUESTION: which PRNG should be used for MC?

ANSWER: The best PRNG is the one which exploited for producing the key results in the corresponding field of research

Notorious generators R250 and RANDU – are the examples of “good” PRNGs failed in real MC simulations

The final results must be independent of used PRNG (!) the key results should be checked with different PRNGs
**Motivation**

PRNGCL is an open-source library of PRNGs for MC simulations

- contains OpenCL implementations of the most popular uniform PRNGs for MC (MRG32k3a, RANLUX, RANMAR, RANECU, XOR128, seven-XOR-shift)
- single or double precision output with full resolution according to IEEE-754 standard

PRNGCL library is a part of QCDGPU package and was tested in real lattice MC simulations of pure SU(N) gluodynamics in external chromomagnetic fields at finite temperature and in investigation of phase transition in O(N) model.
PRNG algorithm produces next state $S_i$ from previous $S_{i-1}$ and generates next item of PRN sequence $X_i$

$$S_i = f(S_{i-1}), \quad X_i = g(S_i)$$

(1)

PRNG algorithm is deterministic – its application to a given state leads to generation of the unique number and the unique next state

General structure of PRNG
**Methods of PRNs sequence parallelization:**

- **random seeding:** each PRNG thread operates its own state $S^i_j$
- **parametrization:** different threads utilize the same PRNG implementation with different parameters
- **block splitting:** PRNs sequence is splitted into equal-size independent subsequences, each of which is produced by the corresponding thread, initial PRNG states have to be computed
- **leapfrogging:** each following member of PRNs sequence is produced by the next thread. The sequence is built by union of several items of subsequences in one. To perform the leapfrogging a PRNG must be able to skip a certain number of PRNs
Existing implementations of PRNG packages on GPUs

CUDA implementations:
- CURAND, Thrust, NAG, GASPRNG, TRNG, Random123, PRAND, MPRNG, GPU-rand, ShoveRand, MTGP

OpenCL implementations:
- Random123, MTGP, OpenCLRNG, RANLUXCL, MWC64X

The most popular PRNGs used in MC: LCGs, RANLUX, MRG32k3a and MT
Implementation
Requirements for PRNGCL development:

1. implementation of classical PRNGs, which are used for actual MC simulations
2. portability and computational hardware-independence
3. independence of a given PRN sequence on computational devices
4. independence of the number of started PRNG threads
5. simple change of PRNG (by one parameter) for further PRNs utilization
6. optimization of PRNG implementations for GPU-like devices
7. simple supplement with other PRNG implementations (GSL-like architecture)
Why OpenCL?

main reason - portability

Currently \(\sim 20\%\) of all scientific researches using GPU accelerators are performed with OpenCL
PRNGCL contains implementations of

- **MRG32k3a**: a 32-bit combined multiple recursive generator with two components of order 3
- **XOR128**: a 32-bit Marsaglia xor-shift generator
- **seven-XOR-shift**: a 32-bit Panneton-L’Ecuyer xor-shift generator
- **RANLUX**: a 24-bit generator with arbitrary luxury levels
- **RANMAR**: a 24-bit lagged Fibonacci generator
- **RANECU**: a 24-bit L’Ecuyer multiplicative linear congruential generator
- **PM**: a 24-bit Park-Miller linear congruential generator (LCG)
- **CONSTANT**: a “toy generator”, that outputs some constant value
PROBLEM: direct or indirect (by dividing integer 32-bit value by double precision value) converting single-to-double precision, does not allow to realize full double precision resolution

(!) Operations with single precision value or even with 32-bit integers do not allow to realize all $2^{52}$ binary states (IEEE 754 standard)

The obvious conclusion – to produce PRNs with double precision from several PRNs with single precision – fill lower bits of output number with first PRN, and upper bits with the second one

To keep the distribution uniformity the accept-reject method is applied

V. Demchik and A. Gulov: Increasing precision of uniform pseudorandom number generators. 

arXiv:1401.8230 [cs.MS], 2014
**PRNGCL key features:**

1. Connection to **PRNGCL** is provided through OpenCL context – library creates/fills memory object with PRNs.
2. **PRNGCL** is based on OpenCL context and supports multi-GPU (MPI: supported with “manual” initialization on nodes).
3. **PRNGCL** contains built-in generator for PRN sequence reproduction on different computational devices.
4. PRNG implementations are independent and may be used separately.
5. **PRNGCL** is not optimized for particular hardware but is independent of the number of started PRNG threads.
7. Open architecture – other PRNG implementations may be easily supplemented.
8. **PRNGCL** contains minimal external library dependencies and is OS platform-independent.
Performance results
Single precision

Performance results

Single precision mode

Vadim Demchik

PRNGCL: OpenCL Library of Pseudo-Random Number Generators for MC Simulations

GTC'14
Double precision

Performance results

Double precision mode

Vadim Demchik

PRNGCL: OpenCL Library of Pseudo-Random Number Generators for MC Simulations

GTC’14

17/19
Conclusions
Conclusions:

1. a new open-source package PRNGCL of pseudo-random number generators for Monte Carlo simulations on OpenCL-compatible devices is developed
2. PRNGCL supports single- or multi-GPU mode
3. PRNs can be generated with single or double precision
4. PRNGCL library is available online:
   https://github.com/vadimdi/PRNGCL

John von Neumann (1951):
“Any one who considers arithmetical methods of producing random digits is, of course, in a state of sin”
Conclusions:

1. A new open-source package PRNGCL of pseudo-random number generators for Monte Carlo simulations on OpenCL-compatible devices is developed.
2. PRNGCL supports single- or multi-GPU mode.
3. PRNs can be generated with single or double precision.
4. PRNGCL library is available online: https://github.com/vadimdi/PRNGCL

John von Neumann (1951):
“Any one who considers arithmetical methods of producing random digits is, of course, in a state of sin”

Thank you for your attention!