RNGAVXLIB: Program library for random number generation, AVX realization.

M.S. Guskova[a,b], L.Yu. Barash[a,c,d], L.N. Shchur[a,b,c]

[a]*Science Center in Chernogolovka, 142432 Chernogolovka, Russia*
[b]*National Research University Higher School of Economics, 101000 Moscow, Russia*
[c]*Landau Institute for Theoretical Physics, 142432 Chernogolovka, Russia*
[d]*Moscow Institute of Physics and Technology, 141700 Moscow, Russia*

Abstract

We present the random number generator (RNG) library RNGAVXLIB, which contains fast AVX realizations of a number of modern random number generators, and also the abilities to jump ahead inside a RNG sequence and to initialize up to 10^{19} independent random number streams with block splitting method. Fast AVX implementations produce exactly the same output sequence as the original algorithms. Usage of AVX vectorization allows to substantially improve performance of the generators. The new realizations are up to 2 times faster than the SSE realizations implemented in the previous version of the library [1], and up to 40 times faster compared to the original algorithms written in ANSI C.

NEW VERSION PROGRAM SUMMARY

Manuscript Title: RNGAVXLIB: Program library for random number generation, AVX realization.
Authors: M.S. Guskova, L.Yu. Barash, L.N. Shchur
Program Title: RNGAVXLIB
Journal Reference:
Catalogue identifier:
Licensing provisions:
Programming language: C, Fortran
Computer: PC, laptop, workstation, or server with Intel or AMD processor
Operating system: Unix, Windows
RAM: 4 Mbytes

Email address: `barash@itp.ac.ru` (L.Yu. Barash)

Number of processors used:
Supplementary material:
Keywords: Statistical methods, Monte Carlo, Random numbers, Pseudorandom numbers, Random number generation, Advanced Vector Extensions (AVX)
Classification: 4.13 Statistical Methods
External routines/libraries:
Subprograms used:
Catalogue identifier of previous version: AEIT_v2_0
Journal reference of previous version: Comput. Phys. Commun. 184 (2013) 2367
Does the new version supersede the previous version?: Yes
Nature of problem: Any calculation requiring uniform pseudorandom number generator, in particular, Monte Carlo calculations. Any calculation requiring parallel streams of uniform pseudorandom numbers.

Solution method: The library contains realization of the following modern and reliable generators: `MT19937` [2], `MRG32K3A` [3], `LFSR113` [4], `GM19`, `GM31`, `GM61` [5, 6], and `GM29`, `GM55`, `GQ58.1`, `GQ58.3`, `GQ58.4` [7, 8]. The library contains realizations written in ANSI C, realizations based on SSE command set and realizations based on AVX command set. Usage of vectorization allows to substantially improve performance of all generators. The library also contains the abilities to jump ahead inside RNG sequence and to initialize independent random number streams with block splitting method for each of the RNGs. C and Fortran are supported.

Reasons for the new version: Modern CPUs better support vectorization compared to CPUs available two years ago when the previous version of the library was prepared. In particular, Advanced Vector Instructions 2 (AVX2) are now supported by CPUs fabricated by Intel and AMD. AVX2 is supported by Intel CPUs since the Haswell microarchitecture was released in June 2013, and is supported by AMD CPUs since the Streamroller Family 15h microarchitecture was released in January 2014. An important new feature of this version is the ability to employ the AVX2 instruction set of a CPU in order to speed up the calculations. As a result, the new RNG realizations employing AVX2 are up to 2 times faster than the realizations implemented in the previous version of the library.

Restrictions: For AVX realizations of the generators, Intel or AMD CPU supporting AVX2 command set is required. For SSE realizations of the generators, Intel or AMD CPU supporting SSE2 command set is required. In order to use the SSE realization for the `lfsr113` generator, CPU must support SSE4.1 command set.

Additional comments: The function call interface has been simplified compared to the previous versions. For each of the generators, RNGAVXLIB supports the following functions, where `rng` should be replaced by the particular name of the RNG:

```
void rng_init_(rng_state* state);
void rng_init_sequence_(rng_state* state,unsigned long long SequenceNumber);
void rng_skipahead_(rng_state* state, unsigned long long N);
unsigned int rng_generate_(rng_state* state);
float rng_generate_uniform_float_(rng_state* state);
unsigned int rng_ansi_generate_(rng_state* state);
float rng_ansi_generate_uniform_float_(rng_state* state);
unsigned int rng_sse_generate_(rng_state* state);
float rng_sse_generate_uniform_float_(rng_state* state);
unsigned int rng_avx_generate_(rng_state* state);
float rng_avx_generate_uniform_float_(rng_state* state);
void rng_print_state_(rng_state* state);
```

The function call interface for the `rng_skipahead_` function, which jumps ahead N output values inside an RNG sequence, can be slightly different for some of the RNGs. For example, the function
`void mt19937_skipahead_(mt19937_state* state, unsigned long long a, unsigned b);`
skips ahead $N = a \cdot 2^b$ numbers, where $N < 2^{512}$, and the function
`void gm55_skipahead_(gm55_state* state, unsigned long long offset64,`
`unsigned long long offset0);`
skips ahead $N = 2^{64} \cdot \texttt{offset64} + \texttt{offset0}$ numbers. The detailed function call interface can be found in the header files of the `include` directory. The examples of using the library can be found in the `examples` directory.

Some of the generators have several versions of the `rng_init_sequence_`, routine, for example, `rng_init_short_sequence_`, `rng_init_medium_sequence_`, `rng_init_long_seque`
(see details in [1, 10]). Maximal number of sequences and maximal length of each sequence for pseudorandom streams are indicated in [1, 10]. The algorithms used to jump ahead in the RNG sequence and to initialize parallel streams of pseudorandom numbers are described in detail in [9, 10].

This version of the library automatically detects whether the CPU supports SSE and/or AVX vectorization at the compilation stage. During the compilation of the library, the `-march=native` compiler option is used, which allows to use predefined macros such as `__SSE2__` and `__AVX2__` in the source code. This is supported by both GNU and Intel compilers. The functions `rng_generate_` and `rng_generate_uniform_float` employ SSE and AVX vectorization if the CPU supports them.

Table 1: Speed of the realizations. CPU: Intel Xeon E5-2650v3 (2.3 GHz); Compiler: gcc; Optimization: -O3.

Generator	ANSI C (Gbit/sec)	SSE (Gbit/sec)	AVX (Gbit/sec)	speed(SSE)/ speed(ANSI C)	speed(AVX)/ speed(SSE)
GM19	0.27	1.43	3.12	5.30	2.17
GM29	0.32	1.86	3.11	5.84	1.67
GM31	0.31	1.29	2.17	4.14	1.69
GM55	0.93	2.30	3.27	2.47	1.42
GM61	0.14	0.48	0.89	3.39	1.84
GQ58.1	0.27	0.63	1.12	2.30	1.79
GQ58.3	0.58	1.31	2.15	2.25	1.65
GQ58.4	0.98	1.97	3.10	2.02	1.57
LFSR113	5.98	3.83	12.6	0.64	3.29
MRG32K3A	2.84	5.21	7.64	1.83	1.46
MT19937	7.43	9.84	12.2	1.32	1.24

This version of the library also supports simultaneous generation of two independent output sequences for the LFSR113 generator using the AVX vectorization:
`void lfsr113_avx_generate_two_(lfsr113_state* state, unsigned * out1, unsigned *out2)`
This is the fastest possible way to generate LFSR113 random numbers using the CPU which supports the AVX2 instruction set. The function `lfsr113_skipahead_` jumps ahead only in the first LFSR output sequence. Jumping ahead in the secound output sequence can be performed with the separate `lfsr113_skipahead2_` routine.

GNU Fortran does not have compiler directives for data alignment to assist vectorization, although Intel Fortran has directives for that, such as `!dir$ attributes align:32`. By default, GNU Fortran aligns all variables to 16-byte boundaries, which is sufficient to efficiently use SSE, but is not sufficient for AVX. We find that applying additional `SAVE` command to the generator state in Fortran results, in particular, in alignment of the data to 32-byte boundaries. This allows one to employ AVX realizations from Fortran (see the `examples` directory). We have tested this on workstations with various CPUs and various versions of Linux.

Development and optimization of the algorithms were supported by the Russian Science Foundation project No. 14-21-00158. Benchmark testing was partially supported by Russian Foundation for Basic Research project No. 13-07-00570 and by the Supercomputing Center of Lomonosov Moscow State University [11].

Table 2: Speed of the realizations. CPU: Intel Core i7-4790K (4 GHz); Compiler: gcc; Optimization: -O3.

Generator	ANSI C (Gbit/sec)	SSE (Gbit/sec)	AVX (Gbit/sec)	speed(SSE)/ speed(ANSI C)	speed(AVX)/ speed(SSE)
GM19	0.45	2.05	4.52	4.55	2.21
GM29	0.51	2.69	4.69	5.27	1.74
GM31	0.53	1.92	3.32	3.63	1.73
GM55	1.36	3.57	5.16	2.62	1.45
GM61	0.21	0.75	1.32	3.50	1.76
GQ58.1	0.44	0.93	1.67	2.12	1.80
GQ58.3	0.85	2.01	3.30	2.37	1.63
GQ58.4	1.38	2.98	4.28	2.16	1.44
LFSR113	10.7	5.60	18.2	0.52	3.25
MRG32K3A	4.20	8.79	12.4	2.09	1.41
MT19937	8.85	14.5	17.8	1.64	1.23

Running time: Running time is of the order of 20 sec for generating 10^9 pseudorandom numbers with a PC based on Intel Core i7-940 CPU. Speed of the random number generation on CPUs widely used in modern servers and workstations is shown in Tables 1 and 2 respectively (see also [6, 7]).

[1] L.Yu Barash, L.N. Shchur, RNGSSELIB: Program library for random number generation. More generators, parallel streams of random numbers and Fortran compatibility, Computer Physics Communications, 184(10), 2367-2369 (2013).

[2] M. Matsumoto and T. Tishimura, Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator, ACM Trans. on Mod. and Comp. Simul. 8 (1), 3-30 (1998).

[3] P. L'Ecuyer, Good Parameter Sets for Combined Multiple Recursive Random Number Generators, Oper. Res. 47 (1), 159-164 (1999).

[4] P. L'Ecuyer, Tables of Maximally-Equidistributed Combined LFSR Generators, Math. of Comp., 68 (255), 261-269 (1999).

[5] L. Barash, L.N. Shchur, Periodic orbits of the ensemble of Sinai-Arnold cat maps and pseudorandom number generation, Phys. Rev. E 73, 036701 (2006).

[6] L.Yu Barash, L.N. Shchur, RNGSSELIB: Program library for random number generation, SSE2 realization, Computer Physics Communications, 182 (7), 1518-1527 (2011).

[7] L.Yu. Barash, Applying dissipative dynamical systems to pseudorandom number generation: Equidistribution property and statistical independence of bits at distances up to logarithm of mesh size, Europhysics Letters (EPL) 95, 10003 (2011).

[8] L.Yu. Barash, Geometric and statistical properties of pseudorandom number generators based on multiple recursive transformations // Springer Proceedings in Mathematics and Statistics, Springer-Verlag, Berlin, Heidelberg, Vol. 23, 265280 (2012).

[9] L.Yu. Barash, L.N. Shchur, On the generation of parallel streams of pseudorandom numbers, Programmnaya inzheneriya, 1 (2013) 24 (in Russian)

[10] L.Yu. Barash, L.N. Shchur, PRAND: GPU accelerated parallel random number generation library: Using most reliable algorithms and applying parallelism of modern GPUs and CPUs, Computer Physics Communications, 185(4), 1343-1353 (2014).

[11] Voevodin Vl.V., Zhumatiy S.A., Sobolev S.I., Antonov A.S., Bryzgalov P.A., Nikitenko D.A., Stefanov K.S., Voevodin Vad.V., Practice of "Lomonosov" Supercomputer // Open Systems J. - Moscow: Open Systems Publ., 2012, no.7. (In Russian)