Software

INTEL® MATH KERNEL LIBRARY - VECTOR STATISTICS (INTEL® MKL VS)

Gennady Fedorov - Technical Consulting Engineer

Intel Architecture, Graphics and Software (IAGS)

PRACE workshop, June 2020

Gennady.Fedorov@intel.com

Intel® Math Kernel Library

Linear Algebra
- BLAS
- LAPACK
- ScaLAPACK
- Sparse BLAS
- Iterative sparse solvers
- PARDISO*
- Cluster Sparse Solver

FFTs
- Multidimensional
- FFTW interfaces
- Cluster FFT

Neural Networks
- Convolution
- Pooling
- Normalization
- ReLU
- Inner Product

Removed since MKL v.2020

Vector RNGs
- Congruential
- Wichmann-Hill
- Mersenne Twister
- Sobol
- Neiderreiter
- Non-deterministic

Summary Statistics
- Kurtosis
- Variation coefficient
- Order statistics
- Min/max
- Variance-covariance

Vector Math
- Trigonometric
- Hyperbolic
- Exponential
- Log
- Power
- Root

And More
- Splines
- Interpolation
- Trust Region
- Fast Poisson Solver

Benchmarks
- Intel(R) Distribution for LINPACK* Benchmark
- High Performance Computing Linpack Benchmark
- High Performance Conjugate radient Benchmark

Intel® Architecture Platforms

Operating System: Windows*, Linux*, MacOS[1]*

[1] Available only in Intel® Parallel Studio Composer Edition.

Intel MKL - Random Number Generators (RNG) Agenda

- **Introduction**
- **RNG API & Usage Modes**
- **Demo – General Case**
- **Parallel Computing, Demos:**
  - BRNG set
  - Skip-Ahead
- **Non-deterministic Generator**

Introduction - Intel® MKL VS components

- Random Number Generators (RNG)
  - Pseudorandom, quasi-random and non-deterministic random number generator
  - Continuous and discrete distributions of various common distribution types
- Summary Statistics
  - Parallelized algorithms for computation of basic statistical estimates for single and double precision multi-dimensional datasets

Introduction - Random Number Generators (RNG)

- Intel® MKL VS provides a set of commonly used continuous and discrete distributions
  - All distributions are based on the highly optimized Basic Random Number Generators and Vector Mathematics

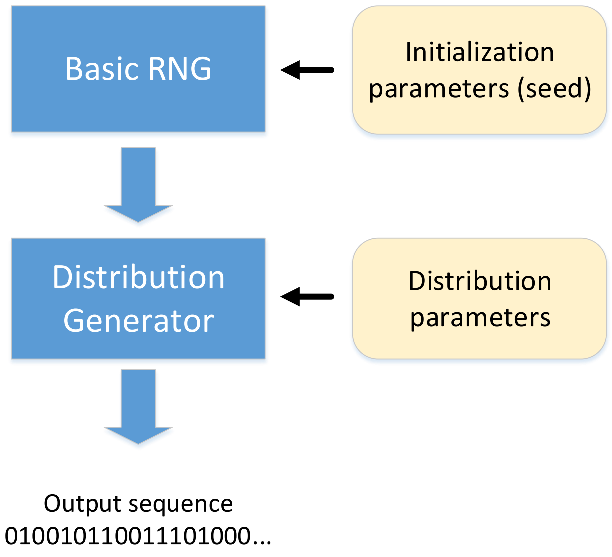
Basic Random Number Generators			
Pseudorandom		**Quasi-random**	**Non-deterministic**
Multiplicative Congruential 59-bit	Multiplicative Congruential 31-bit	Sobol	RDRAND based (HW dependent)
Multiple Recursive	Wichmann-Hill	Niederreiter	
Mersenne Twister 19937	Mersenne Twister 2203		
SIMD-oriented Fast Mersenne Twister 19937	Philox4x32-10 Counter-Based		
ARS-5 Counter-Based (HW dependent)	R250 Shift-Register		

Distribution Generators			
Continuous		**Discrete**	
Uniform	Cauchy	Uniform	Binomial
Gaussian	Rayleigh	UniformBits	Hypergeometric
GaussianMV	Lognormal	UniformBits32	Poisson
Exponential	Gumbel	UniformBits64	PoissonV
Laplace	Gamma	Bernoulli	NegBinomial
Weibull	Beta	Geometric	Multinomial
ChiSquare			

RNG – API & Usage Model

- A typical algorithm for VS random number generation is as follows:
  - Create and initialize stream

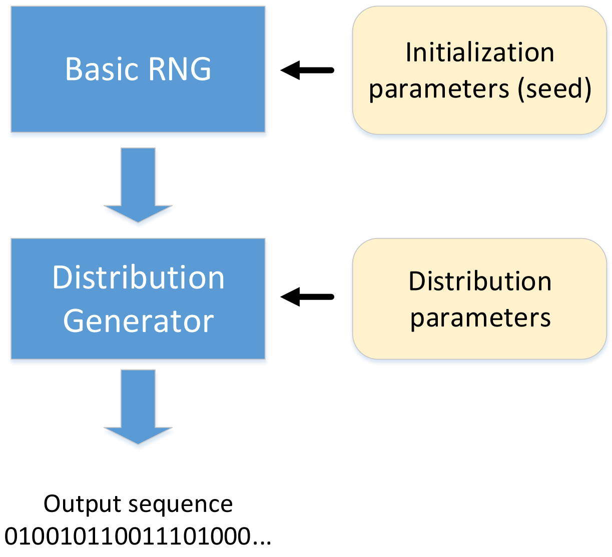
Step	Step description
RNG stream initialization	vslNewStream (&stream, VSL_BRNG_MT2203, 777); BRNG Type: VSL_BRNG_MT2203; Seed: 777



RNG – API & Usage Model

- A typical algorithm for VS random number generation is as follows:
  - Create and initialize stream
  - Call RNG and process the output

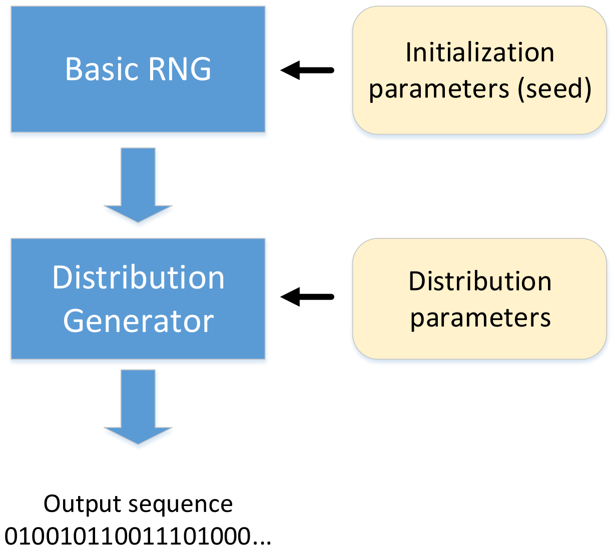
Step	Step description
RNG stream initialization	vslNewStream (&stream, VSL_BRNG_MT2203, 777); BRNG Type (VSL_BRNG_MT2203); Seed (777)
Random number generation	vsRngUniform(VSL_RNG_METHOD_UNIFORM_STD, stream, N, r, a, b); Distribution type (vsRngUniform); Generation method (VSL_RNG_METHOD_UNIFORM_STD); Generation parameters (Used RNG stream, number of elements, etc.) (stream, N, r, a, b)



RNG – API & Usage Model

- A typical algorithm for VS random number generation is as follows:
  - Create and initialize stream
  - Call RNG and process the output
  - Delete the stream

Step	Step description
RNG stream initialization	vslNewStream (&stream, VSL_BRNG_MT2203, 777); BRNG Type: VSL_BRNG_MT2203; Seed: 777
Random number generation	vsRngUniform(VSL_RNG_METHOD_UNIFORM_STD, stream, N, r, a, b); Distribution type: vsRngUniform; Generation method: VSL_RNG_METHOD_UNIFORM_STD; Generation parameters (Used RNG stream, number of elements, etc.): stream, N, r, a, b
RNG stream de-initialization	vslDeleteStream(&stream);



RNG – Service Routines

Service Routines

Routine	Short Description
vslNewStream	Creates and initializes a random stream.
vslNewStreamEx	Creates and initializes a random stream for the generators with multiple initial conditions.
vsliNewAbstractStream	Creates and initializes an abstract random stream for integer arrays.
vsldNewAbstractStream	Creates and initializes an abstract random stream for double precision floating-point arrays.
vslsNewAbstractStream	Creates and initializes an abstract random stream for single precision floating-point arrays.
vslDeleteStream	Deletes previously created stream.
vslCopyStream	Copies a stream to another stream.
vslCopyStreamState	Creates a copy of a random stream state.
vslSaveStreamF	Writes a stream to a binary file.
vslLoadStreamF	Reads a stream from a binary file.
vslSaveStreamM	Writes a random stream descriptive data, including state, to a memory buffer.
vslLoadStreamM	Creates a new stream and reads stream descriptive data, including state, from the memory buffer.
vslGetStreamSize	Computes size of memory necessary to hold the random stream.
vslLeapfrogStream	Initializes the stream by the leapfrog method to generate a subsequence of the original sequence.
vslSkipAheadStream	Initializes the stream by the skip-ahead method.
vslGetStreamStateBrng	Obtains the index of the basic generator responsible for the generation of a given random stream.
vslGetNumRegBrngs	Obtains the number of currently registered basic generators.

Requirements

- Intel® Parallel Studio XE 2020 Composer Edition with Intel® C++ Compiler
- Linux* OS supported by Intel® C++ Compiler
- Recommended to have at least 3rd **generation Intel® Core™ processor** (with Intel® AVX2)
- Setting the PATH, LIB, and INCLUDE environment variables

Compiler:

source /opt/intel/compilers_and_libraries_2020.1.127/linux/bin/compilervars.sh intel64

MKL: source <mklroot>/bin/mklvars.sh intel64

All experiments were done at the Intel® Xeon® Gold 6148 Processor

Demo – General Case

directory: <mkl_workshop>/RNG/#1General

- Review test: rng_philox.c test (and errcheck.inc and engine.inc)
- Compiling: **icc –mkl rng_philox.c**
- Running : ./a.out

Expected Outputs:

……………………………………………

```
[gfedorov@skl10 #1_general]$ ./a.out
r[0]=0.0836
……..
r[9]=0.5227
Vector length = 100000, CPE = 0.8012
```

Demo – General Case, IA dispatching

review **run_dispatch.sh**

export MKL_ENABLE_INSTRUCTIONS=

{SSE2, SSE4_2, AVX, AVX2,AVX512}

Expected performance: Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz

SSE4.2 CPE = 3.4773
AVX CPE = 3.4277
AVX2 CPE = 2.1180
AVX-512 CPE = 0.7906

Demo – General Case, CNR mode

Review **./run_cnr.sh**

unset MKL_ENABLE_INSTRUCTIONS

./run_cnr.sh

Expected performance (Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz)

CBWR COMPATIBLE,	CPE = 9.0394
CBWR SSE2,	CPE = 9.0405
CBWR SSE4_2,	CPE = 3.4765
CBWR AVX,	CPE = 3.4275
CBWR AVX2,	CPE = 2.1060
CBWR AVX-512,	CPE = 0.7999

Demo – General Case, Vector

- review **makefile** and **run_size.sh**
- **make** to build lp64 and ilp64, threaded and sequential
  - ./1.out 100000
  - ./2.out 100000

  Conclusion?

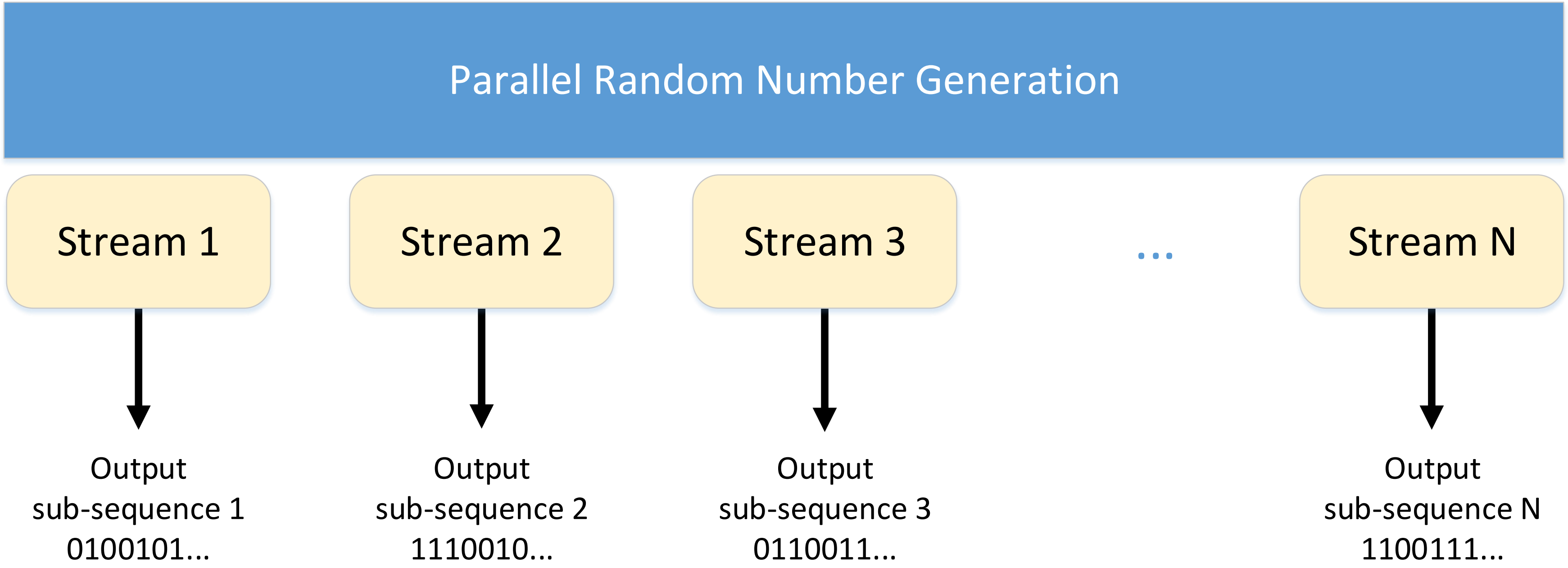
- Running : ./run_size.sh

Expected performance:

Vector Length	CPE (Clock per elements)
2	88.4865
10	31.8759
100	3.6266
1000	1.1264
10000	0.8438
100000	0.7947
1000000	0.7902
……………….	
2000000000	0.8558

RNG - Parallel Computing

- Basic requirements for random number streams are their mutual independence and lack of inter-correlation
- Independent streams can be generated by the following VS methods:
  - BRNG set
  - Skip-ahead
  - Leapfrog



RNG - Parallel Computing. BRNG Set

- The sequence of random numbers can be generated by the set of mutually “independent” streams
  - Wichmann-Hill contains a set of 273 combined multiplicative congruential generators
  - MT2203 contains a set of 6024 Mersenne Twister pseudorandom number generators
- The produced sequences are independent according to the spectral test

Demo-Parallel Computing: BRNG set

directory: <mkl_workshop>/**RNG/#2BRNG**

- Review test cat **rng_mt_parallel.cpp** | less:
  - **NS = #RNG/#streams** // Number of RNG per stream
  - **VSLStreamStatePtr streamS**[N_STREAMS]; // Set of streams
  - for(i=0; i<N_STREAMS;i++)
    vslNewStream(**&streamS[i],** VSL_BRNG_MT2203 + i, ...); // Creating array of streams
  - for (i=0;i<N_STREAMS;i++)
    vsRngUniform(*.*, streamS[I], #RNG_per_Stream, &(rS[i*NS]), *.*); // RNG generation in parallel

Demo-Parallel Computing: BRNG set, cont

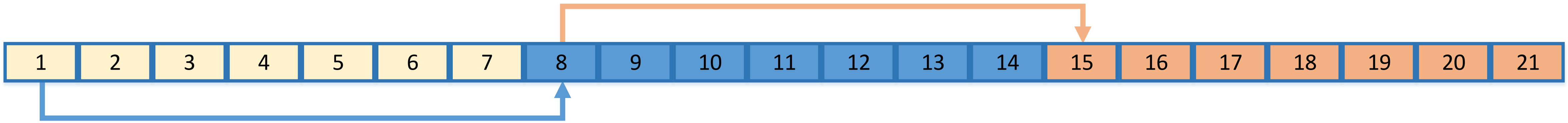
Directory: <mkl_workshop>/RNG/#2BRNG

- Compiling: **icc -qopenmp -mkl rng_mt_parallel.cpp**
- Running: ./a.out <#threads>
- **./run.sh**

Performance of parallel random number generations by MT2203						
#threads	1	2	4	8	16	30
CPE of seguential version	0.697669					
CPE of OpenMP version	0.659293	0.300453	0.155069	0.082701	0.061925	0.297636

MKL RNG - Parallel Computing. Skip-Ahead

- The original sequence is splitted into #STREAMS non-overlapping blocks
  - where #STREAMS - the number of independent streams
- Each of the streams generates random numbers only from the corresponding block



1st node stream

2nd node stream

3rd node stream

At the 1st node the stream contains 1, 2, 3, 4, 5, 6, 7.
At the 2nd node the stream contains 8, 9, 10, 11, 12, 13, 14.
At the 3rd node the stream contains 15, 16, 17, 18, 19, 20, 21.

Demo - MKL RNG, Skip-Ahead method

directory: <mkl_workshop>/RNG/#3skipahead

- Review test cat rng_skipahead.cpp | less

- int NS = #RNG/N_STREAMS; // Number of RNG per stream
- VSLStreamStatePtr streamS[N_STREAMS]; // Set of streams
- vslNewStream (&stream, RNG_method, seed); // Create Base Stream
- vslCopyStream(&streamS[i], stream); // Copy Base stream
- vslSkipAheadStream(streamS[i], <Number of skipped elements>) // Initializes SkipAhead stream
- for (i=0;i<N_STREAMS;i++)

  viRngUniformBits(method, streamS[i], NS, dst); //RNG generation in parallel

Demo – MKL RNG, Skip-Ahead

directory: <mkl_workshop>/RNG/#3skipahead

- Compiling: **icc -qopenmp -mkl rng_skipahead.cpp**
- Running: ./a.out <#threads>

- **./run.sh**

Performance (*) of parallel random number generations by SkipAhead method						
#threads	1	2	4	8	16	32
CPE of seguential version	1.400897					
CPE of OpenMP version	1.404006	0.711407	0.37554	0.196238	0.163317	0.4386

MKL RNG - Skip-Ahead/Leapfrog Support

BRNG	Skip-Ahead and Leapfrog Support	
	Leapfrog	Skip-Ahead
MCG31m1	Supported	Supported
R250	-	-
MRG32k3a	-	Supported
MCG59	Supported	Supported
WH	Supported	Supported
MT19937	-	Supported
SFMT19937	-	Supported
MT2203	-	-
SOBOL	Supported to pick out individual components of quasi-random vectors	Supported
NIEDERREITER	Supported to pick out individual components of quasi-random vectors	Supported
PHILOX4X32X10	-	Supported
ARS5	-	Supported
ABSTRACT	-	-
NON-DETERMINISTIC	-	-

MKL RNG – Non-deterministic Generator

Available since version of MKL v.11.1 and Compiler 13.1

Supported since **Intel Ivy Bridge 2012 microarchitecture and later**

This is non-deterministic random number generator - aka “True Generator”

- DRNG passed all NIST SP800-22 tests
- Supported by Intel Compiler and MKL

Intel Compiler : Generate random numbers of 16/32/64 bit wide random integers. These intrinsics are mapped to the hardware instruction RDRAND

Examples:

```
extern int _rdrand16_step(unsigned short *random_val);
extern int _rdrand32_step(unsigned int *random_val);
extern int _rdrand64_step(unsigned __int64 *random_val);
```

MKL RNG – Non-deterministic Generator, cont

directory: <mkl_workshop>/**RNG/#4nondeterm**

review test: rng_non_determ.cpp:

- VSLStreamStatePtr stream;
- vslNewStream(&stream, BRNG, SEED); //BRNG == **VSL_BRNG_NONDETERM**
- vsRngUniform (VSL_RNG_METHOD_UNIFORM_STD, stream, <N_of_RNG>, r, a, b);
- vslDeleteStream(&stream);

MKL RNG – TRUE Generator, cont

icc –mkl rng_non_determ.cpp

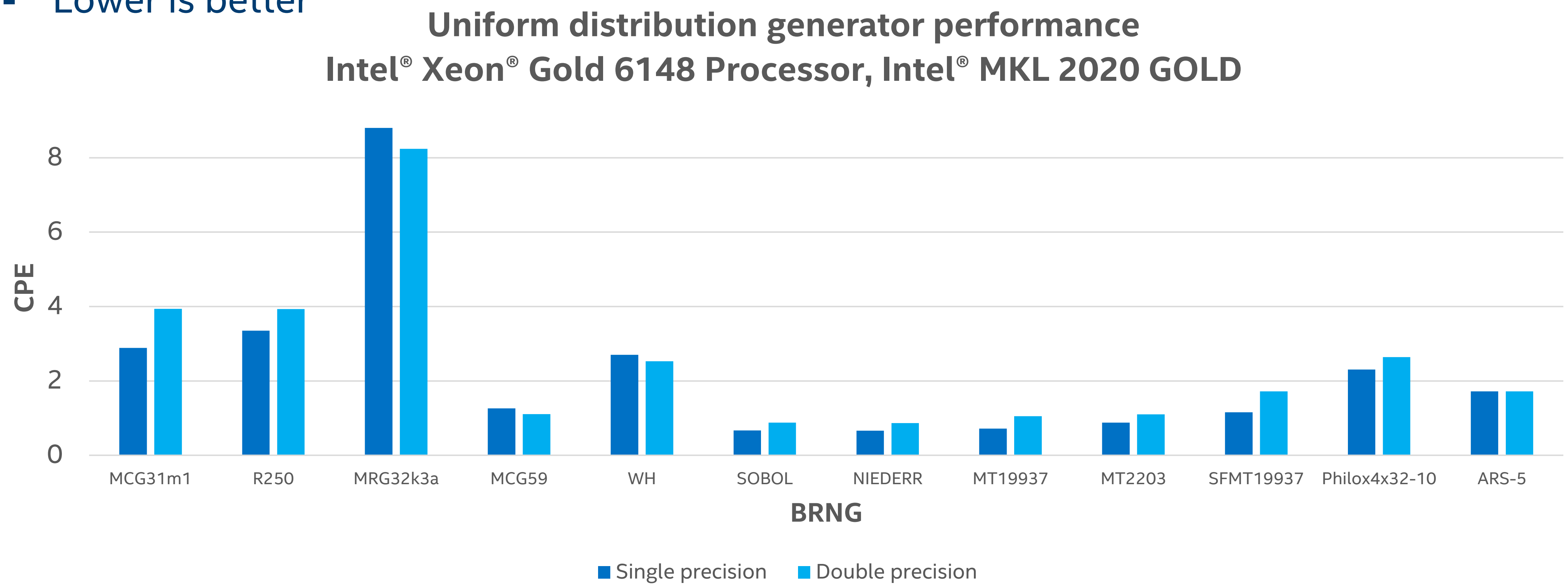
./run_size.sh

Vector length	NONDETERM, CPE
2	43837.4
10	8253.7
100	900.4
1K	162.8
10K	16.2
100K	1.6
1000K	0.16

Philiox, CPE
88.4
31.8
3.6
1.1
0.8
0.7
0.7

MKL RNG – Performance

- Performance metric: Cycles-per-element (CPE)
  - Lower is better



Intel MKL Resources

Intel® MKL website:

– https://software.intel.com/en-us/intel-mkl

Intel MKL forum:

– https://software.intel.com/en-us/forums/intel-math-kernel-library

Intel® MKL link line advisor:

– http://software.intel.com/en-us/articles/intel-mkl-link-line-advisor/

Notes for Intel® MKL Vector Statistics:

- https://software.intel.com/en-us/mkl-vsnotes

Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

Software