Intel® Intel® oneAPI Math Kernel Library (oneMKL)

Gennady.Fedorov@intel.com

Intel® oneAPI Base Toolkit

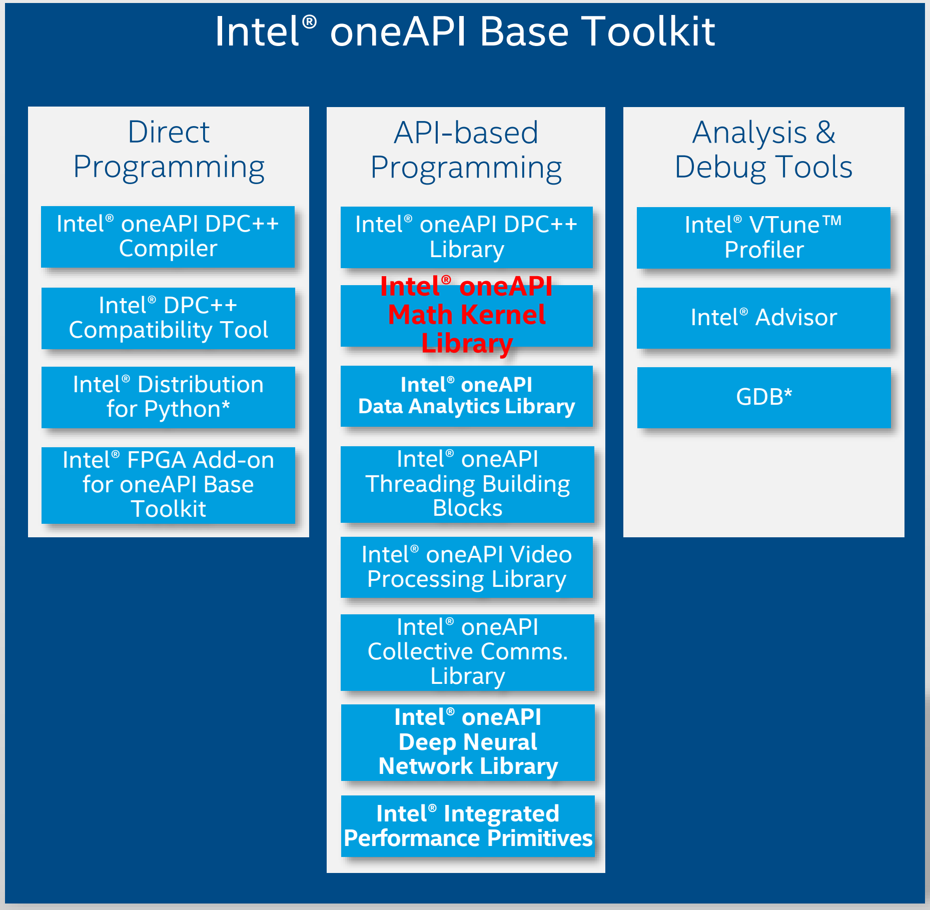
Core set of frequently used tools and libraries for developing high-performance applications across diverse architectures—CPU, GPU, FPGA.

Who Uses It?

- A broad range of developers across industries
- Add-on toolkit users because this is the base for all toolkits

Top Features/Benefits

- Data Parallel C++ (DPC++) compiler, library, and analysis tools
- DPC++ Compatibility tool helps migrate existing CUDA code
- Python distribution includes accelerated scikit-learn, NumPy, SciPy libraries
- Optimized performance libraries for threading, math, data analytics, deep learning, and video/image/signal processing



Intel® oneAPI Math Kernel Library (oneMKL) Features

1. Language support for DPC++ and Intel® C & Fortran compilers.
2. Great performance with minimal effort.
3. Full support for CPUs and select support for Intel® Processor Graphics Gen9, Gen12, and discrete Intel® GPUs.
4. Speeds computations for scientific, engineering, and financial applications by providing highly optimized, threaded, and vectorized math functions.
5. Provides key functionality for dense and sparse linear algebra (BLAS, LAPACK, MKL PARDISO), FFTs, vector math, summary statistics, splines, and more.
6. Dispatches optimized code for each processor automatically without the need to branch code.
7. Optimized for single-core vectorization and cache utilization.
8. Automatic parallelism for multi-core CPUs, GPUs, and scales from core to clusters.
9. Available at no cost and royalty-free.
10. Available: Intel® oneAPI Base Toolkit, standalone. IA32 is separated.

Intel® oneAPI Math Kernel Library (oneMKL)

Linear Algebra
- BLAS
- LAPACK
- ScaLAPACK
- Sparse BLAS
- Graph
- Cluster Sparse Solver
- PARDISO/Direct Sparse Solver

FFTs
- Multi-dimensional
- FFTW interfaces
- Cluster FFT

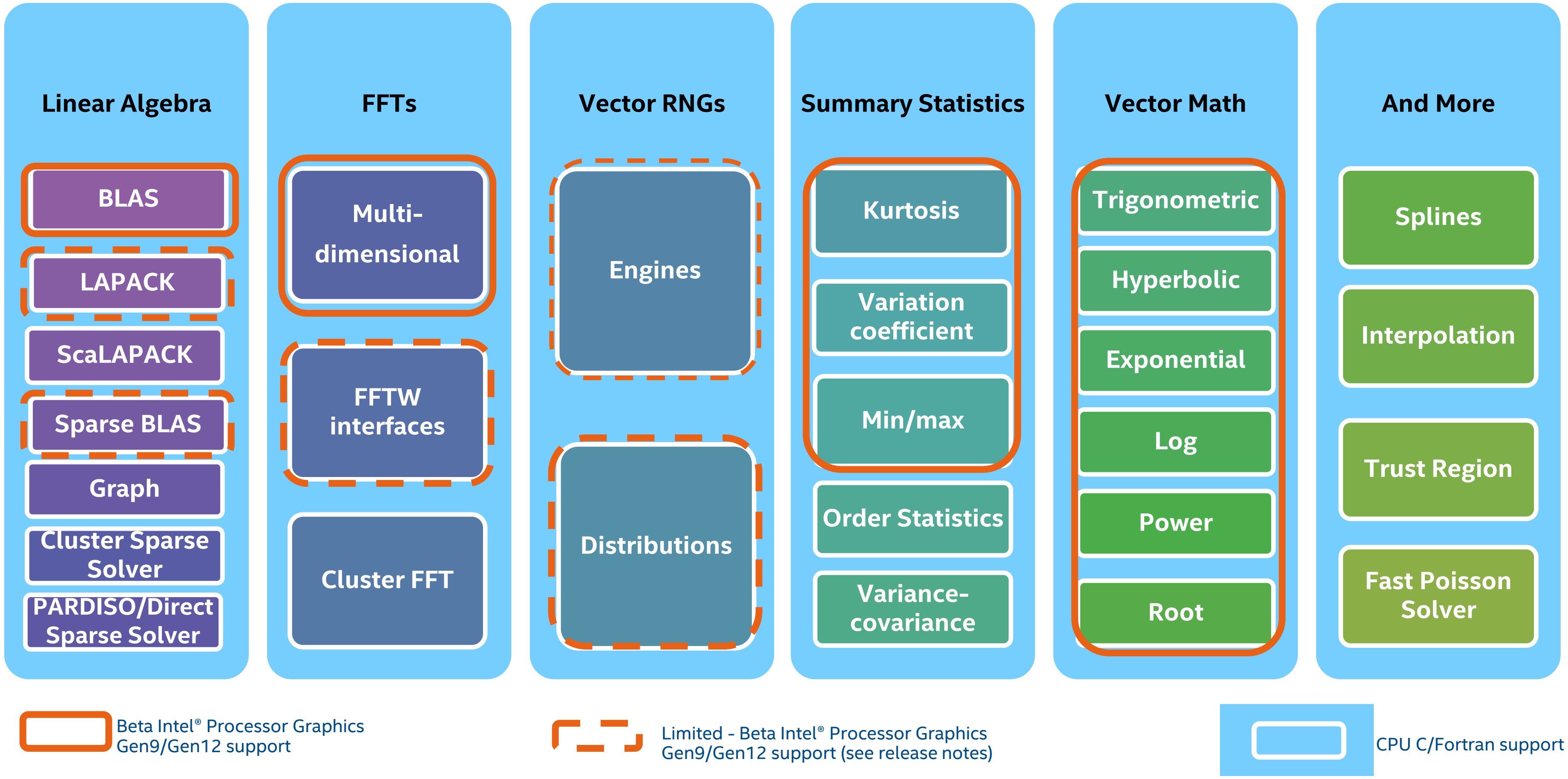
Vector RNGs
- Engines
- Distributions

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

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

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

Intel® oneAPI Math Kernel Library (oneMKL)



Intel® oneAPI MKL, Domain areas

Domain	CPU APIs			Intel GPU APIs		
	DPC++	C	Fortran	DPC++	C OpenMP* Offload	Fortran OpenMP* Offload
BLAS and BLAS-like Extensions	Yes	Yes	Yes	Yes	Yes	Yes
LAPACK and LAPACK-like Extensions	Yes1	Yes	Yes	Yes1	Yes2	Yes2
ScaLAPACK	No	Yes	Yes	No	No	No
Vector Math	Yes	Yes	Yes	Yes	Yes	Yes
Vector Statistics (Random Number Generators)	Yes	Yes	Yes	Yes1	Yes2	Yes2
Vector Statistics (Summary Statistics)	Yes1	Yes	Yes	Yes1	Yes2	Yes2
Data Fitting	No	Yes	Yes	No	No	No
FFT/DFT	Yes	Yes	Yes	Yes	Yes	Yes
Sparse BLAS	Yes1	Yes	Yes	Yes1	Yes2	No
Sparse Solvers	No	Yes	Yes	No	No	No

1: Subset of the full functionality available. Refer to the DPC++ developer reference for full list of DPC++ functionality supported.

2: Subset of the full functionality available. For the list of functionality, refer to the developer reference (C and Fortran)

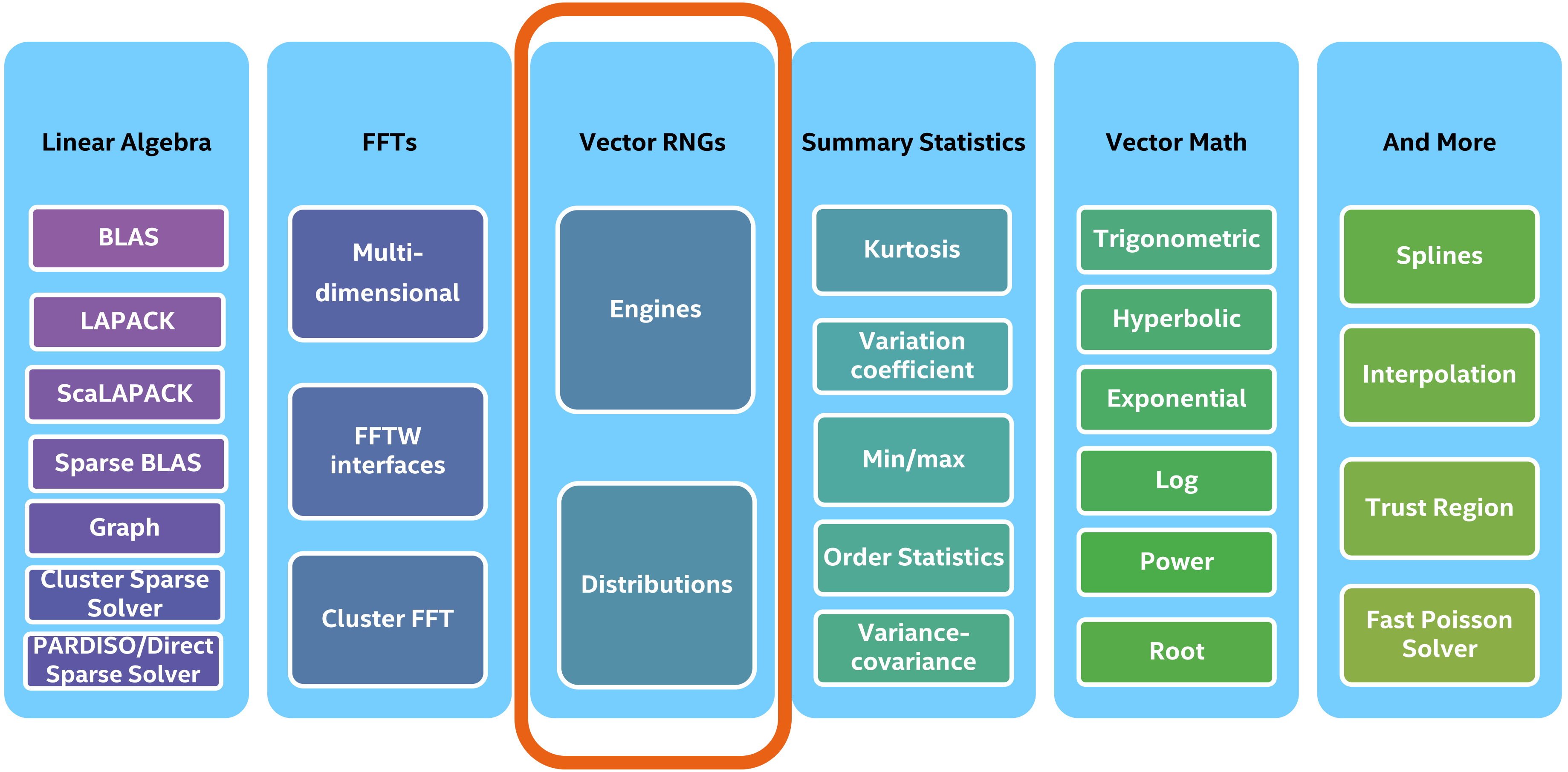
BLAS_64/Lapack_64 API Extensions

- Using BLAS and LAPACK with the 32-bit and 64-bit interface (lp64 / ilp64) at the same time
- BLAS_64 and LAPACK_64 - NetLib interfaces
- Declaration: mkl_blas_64.h, mkl_lapack.h,
- Limitations :
  - Intel64
  - C API
  - BLAS extensions - mkl_trans.h (?imatcopy,?omatcopy)
  - LAPACKE (C API for LAPACK)
  - CPU only

BLAS_64/Lapack_64 API Extensions, cont.

- BLAS:
  - void **sgemm**(const char *transa, const char *transb, const **MKL_INT** *m, const **MKL_INT** *n, const **MKL_INT** *k, const float *alpha, const float *a, const **MKL_INT** *lda, const float *b, const **MKL_INT** *ldb, const float *beta, float *c, const **MKL_INT** *ldc) NOTHROW;
  - void **sgemm_64**(const char *trans, const MKL_INT64 *m, const MKL_INT64 *n, const float *alpha, const float *a, const MKL_INT64 *lda, const float *x, const MKL_INT64 *incx, const float *beta, float *y, const MKL_INT64 *incy) NOTHROW;
- LAPACK:
  - void sgetrf(const **MKL_INT*** m, const **MKL_INT*** n, float* a, const **MKL_INT*** lda, **MKL_INT*** ipiv, **MKL_INT*** info) NOTHROW;
  - void sgetrf_64(const MKL_INT64* m, const MKL_INT64 * n, float* a, const MKL_INT64 * lda, MKL_INT64 * ipiv, MKL_INT64 * info) NOTHROW;

Intel® oneAPI Math Kernel Library, RNG



Random Number Generators (RNG) , Intro

- 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

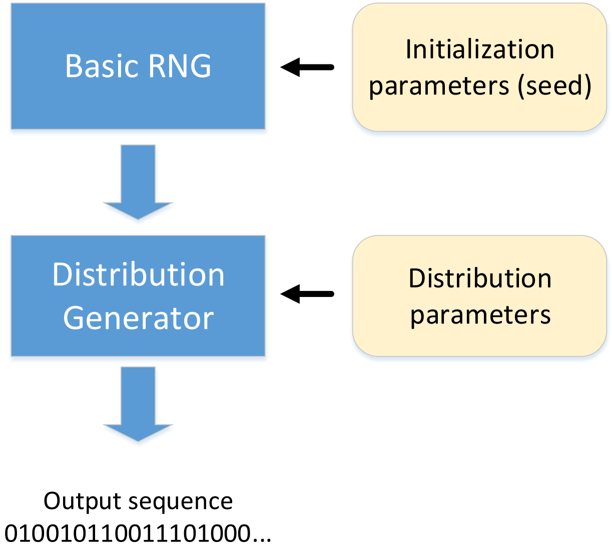
Basic Random Number Generators			
Pseudorandom		Quasi-random	Non-deterministic
Multiplicative Congruential 59-bit	Multiplicative Congruential 31-bit	Sobol	RDRAND based (HW dependent)
Multiple Recursive	Wichmann-Hill	Niederreiter	
Mersenne Twister 19937	Mersenne Twister 2203		
SIMD-oriented Fast Mersenne 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

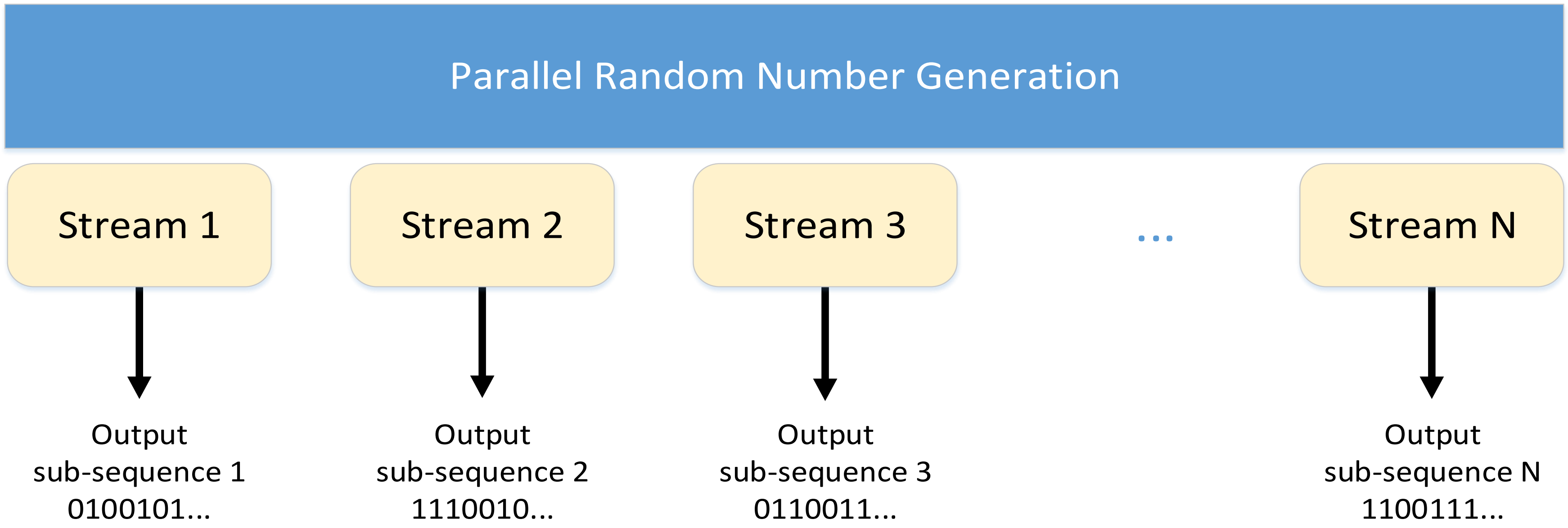
- A typical algorithm for VS random number generation is as follows:
  - Create and initialize stream
  - Call RNG and process the output
  - Delete the stream

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 - 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

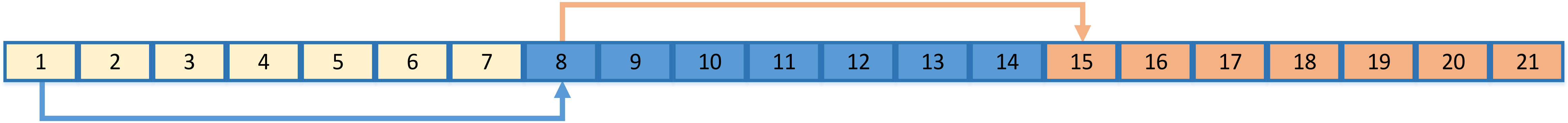


RNG - Parallel Computing. BRNG Set

- 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

RNG - Parallel Computing. Skip-Ahead

- The original sequence is split into k non-overlapping blocks
  - where k - 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

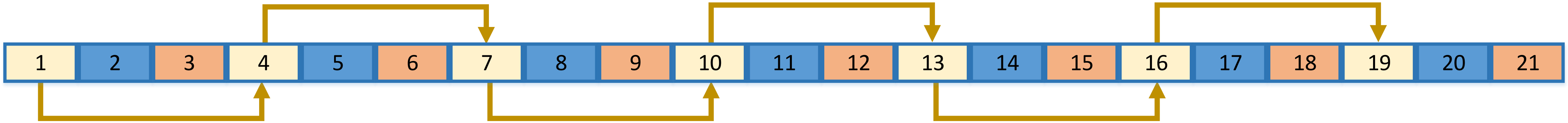
At the 1st node the stream contains 1, 2, 3, 4, 5, 6, 7.

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.

RNG - Parallel Computing. Leapfrog

- The original sequence is split into k disjoint sub-sequences
  - where k - the number of independent streams
- Each of the streams generates random numbers only from the corresponding subsequence



1st node stream

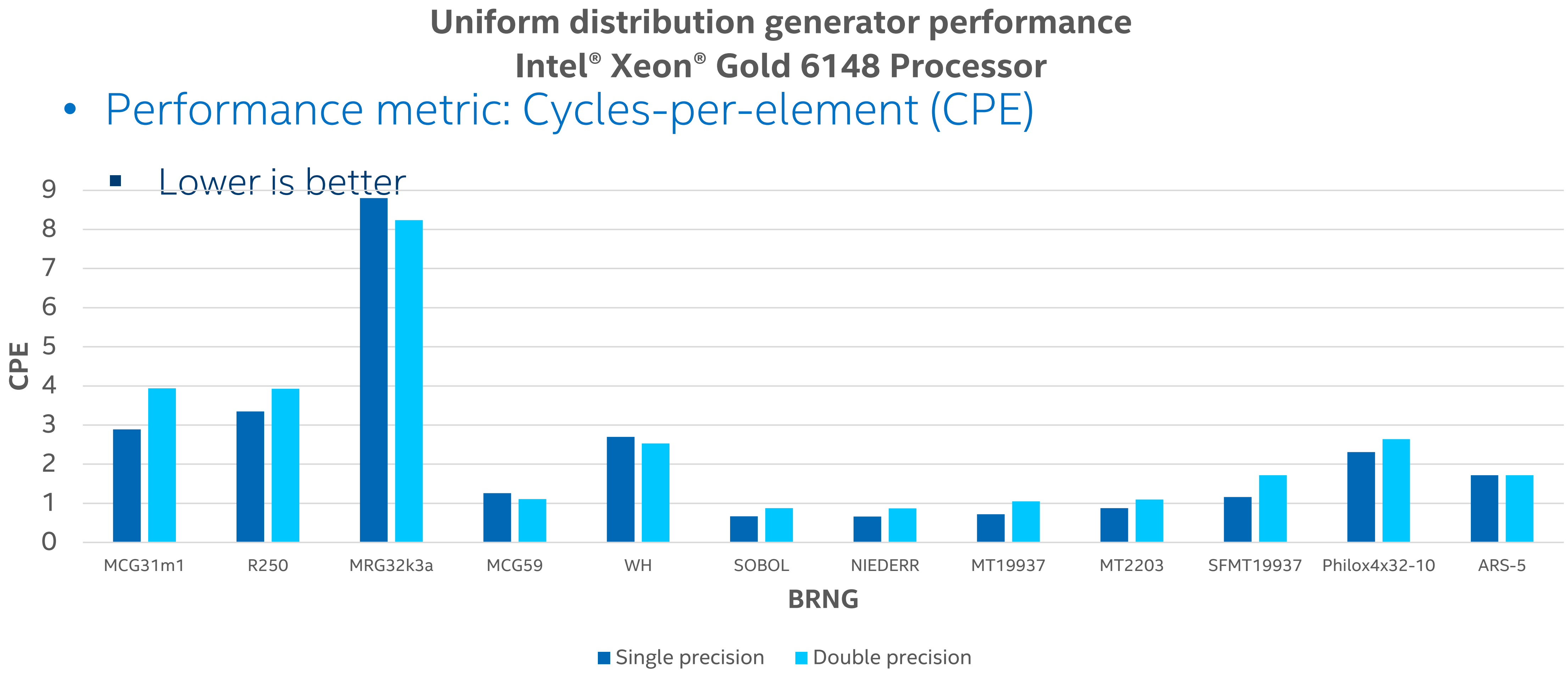
2nd node stream

3rd node stream

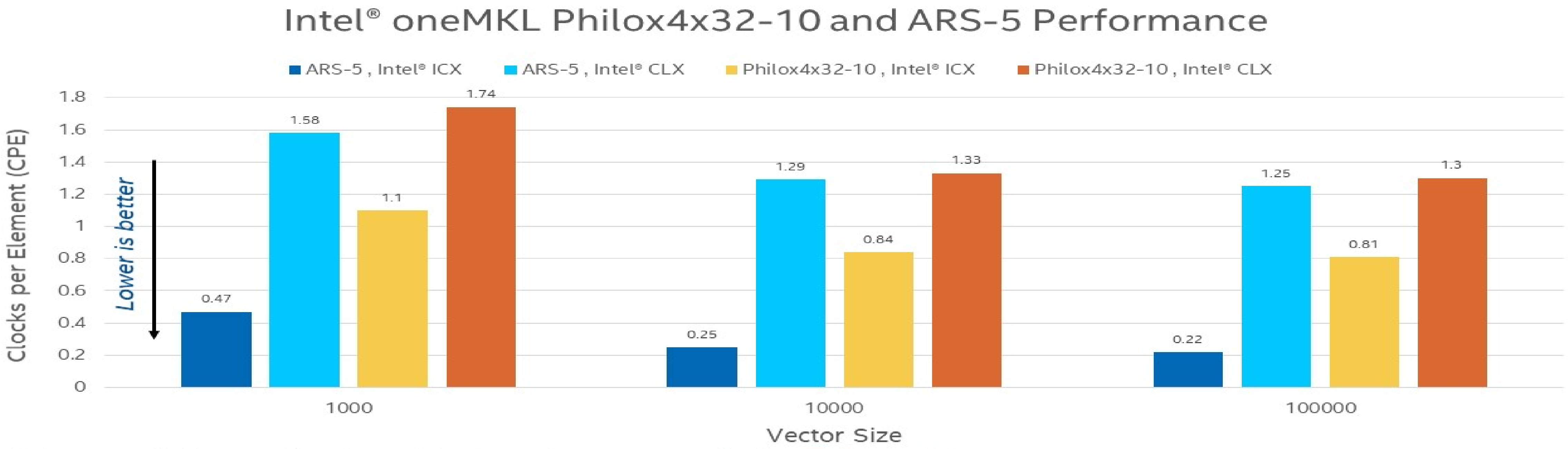
At the 1st node the stream contains 1, 4, 7, 10, 13, 16, 19.
At the 2nd node the stream contains 2, 5, 8, 11, 14, 17, 20.
At the 3rd node the stream contains 3, 6, 9, 12, 15, 18, 21.

MKL RNG - Performance

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



MKL RNG – Performance, cont



Performance varies by use, configuration, and other factors. Learn more at www.intel.com/PerformanceIndex.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration disclosure for details. No product or component can be absolutely secure.

Your costs and results may vary. Intel technologies may require enabled hardware, software, or service activation.

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. For more complete information visit www.intel.com/benchmarks.

Configuration: Testing by Intel as of 04/17/2021 Intel® oneAPI Math Kernel Library 2021.2; Intel(R) Xeon(R) Platinum 8280L CPU @ 2.70GHz, 2 sockets, 28 cores per socket; Intel(R) Xeon(R) Platinum 8380 CPU @ 2.30GHz, 2 sockets, 40 cores per socket;

Intel® oneAPI MKL, RNG, GPU

Engines (Basic Random Number Generators)			
Pseudorandom		Quasi-random	Non-deterministic
Multiplicative Congruential 59-bit (mcg59)	Multiplicative Congruential 31-bit (mcg31m1)	Sobol	RDRAND based (HW dependent)
Multiple Recursive (mrg32k3a)	Wichmann-Hill (wichmann_hill)	Niederreiter	
Mersenne Twister 19937 (mt19937)	Mersenne Twister 2203 (mt2203)		
SIMD-oriented Fast Mersenne Twister 19937 (sfmt19937)	Philox4x32-10 Counter-Based (philox4x32x10)		
ARS-5 Counter-Based (HW dependent) (ars5)	R250 Shift-Register (r250)		

Distributions			
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			

Available in C, Fortran, DPC++ and OpenMP offload for all devices

Available in C, Fortran and DPC++ for CPU only

Intel® oneMKL Resources

Intel® oneMKL Product Page	https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl.html
Get Started with Intel® oneMKL	https://www.intel.com/content/www/us/en/develop/documentation/get-started-with-mkl-for-dpcpp/top.html
Intel® oneMKL Developer Reference	https://www.intel.com/content/www/us/en/develop/documentation/onemkl-developer-reference-c/top.html
Intel® oneMKL Developer Guide	https://www.intel.com/content/www/us/en/develop/documentation/onemkl-windows-developer-guide/top.html
Intel® oneMKL Specification	https://spec.oneapi.io/versions/latest/elements/oneMKL/source/index.html
Intel® oneMKL Open-Source Interfac	https://github.com/oneapi-src/oneMKL
Intel® oneMKL Release Notes	https://cqpreview.intel.com/content/www/us/en/developer/articles/release-notes/onemkl-release-notes.html
Intel® oneMKL Forum	https://community.intel.com/t5/Intel-oneAPI-Math-Kernel-Library/bd-p/oneapi-math-kernel-library

Notices & Disclaimers

Intel technologies may require enabled hardware, software or service activation. Learn more at intel.com or from the OEM or retailer.

Your costs and results may vary.

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

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. https://software.intel.com/en-us/articles/optimization-notice

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. See backup for configuration details. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration disclosure for details. No product or component can be absolutely secure.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

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);
```

Returned value: 1 (if the hardware returns random value) and 0 – if failed