Intel® oneAPI Math Kernel Library (oneMKL)

HLRS oneAPI Workshop

September 13th–15th, 2023

matthias.kirchhart@intel.com

Introduction

What is “the” oneMKL?

Different meaning in different contexts, but one common theme:

A set of functions of commonly used mathematical operations, such as:

- Linear algebra (basic arithmetic, solvers & decompositions, etc.)
- Fast Fourier Transforms (FFT)
- Random number generators

Do not write these routines yourself, use oneMKL!

A “good” oneMKL implementation often outperforms naïvely written code by a factor of 100x and more. (Performance varies depending on implementation, machine, context, problem type and size, etc. pp.)

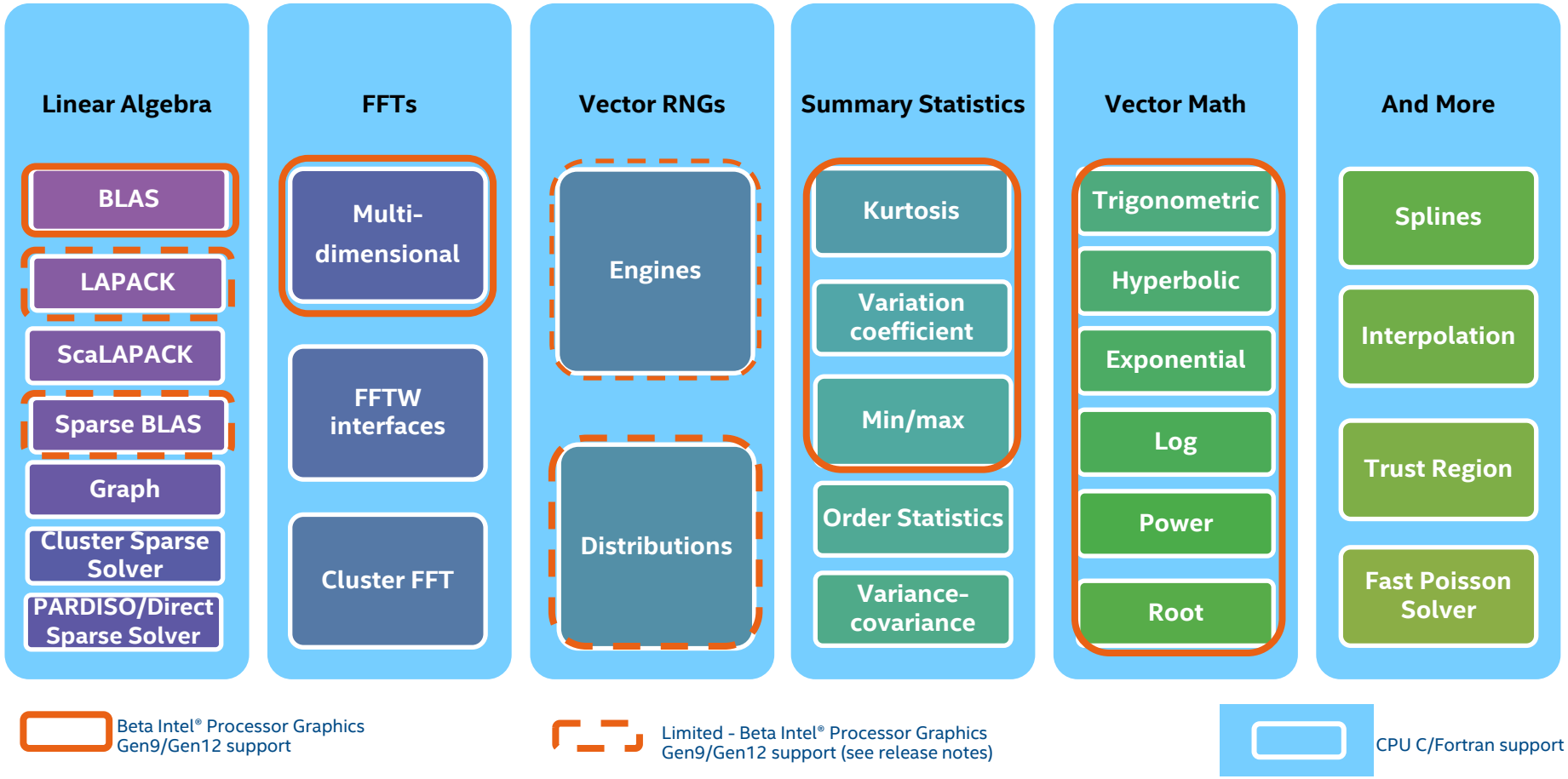
The different meanings of “oneMKL”

1. The SYCL* interface defined in the open oneAPI standard
2. Other vendors’ (partial) implementations of the standard

The different meanings of “oneMKL”

1. The SYCL* interface defined in the open oneAPI standard
2. Other vendors’ (partial) implementations of the standard
3. Intel’s implementation of it: Intel® oneAPI Math Kernel Library
   - Origin of the standard and the most complete implementation of it
   - Extremely optimised for Intel® hardware
   - Apart from the SYCL* interface, also provides interfaces for:
     - C++ and plain C, Fortran
     - Other de facto standard APIs: BLAS, LAPACK, FFTW, etc.
     - Support for distributed and heterogeneous computing via OpenMP and MPI.

Overview of Intel® oneMKL Functionality



oneMKL is part of the Intel® oneAPI Base Toolkit

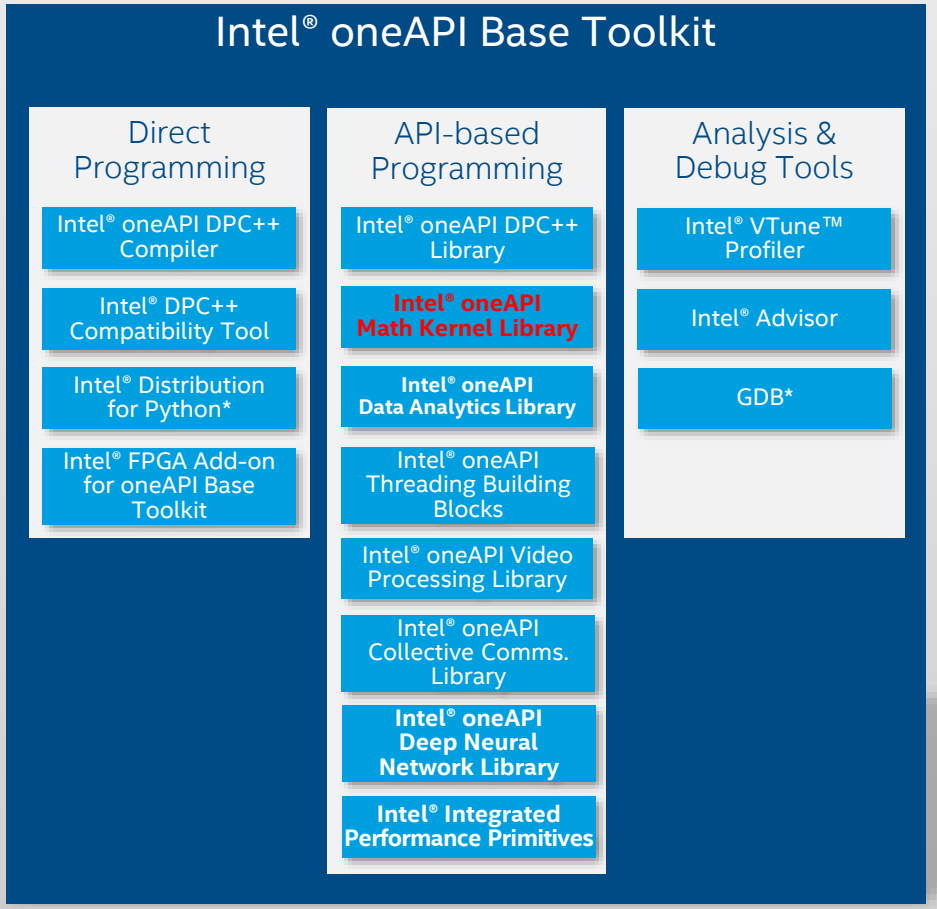
Core set of frequently used tools and libraries for developing high-performance applications across diverse architectures—CPU, GPU, FPGA.

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



Also available as a standalone Download

https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl-download.html

A Simple Example

A simple Matrix–Matrix product

Given two NxN matrices A and B, compute their product C = A*B.

A simple Matrix–Matrix product

Given two NxN (N=1024) matrices A and B, compute their product C = A*B.

Naïve C code:

```
for ( size_t i = 0; i < n; ++i )
for ( size_t j = 0; j < n; ++j )
for ( size_t k = 0; k < n; ++k )
    C[ i + j*n ] += A[ i + k*n ]*B[ k + j*n ];
```

A simple Matrix–Matrix product

Given two NxN (N=1024) matrices A and B, compute their product C = A*B.

Naïve C code:

```
for ( size_t i = 0; i < n; ++i )
for ( size_t j = 0; j < n; ++j )
for ( size_t k = 0; k < n; ++k )
    C[ i + j*n ] += A[ i + k*n ]*B[ k + j*n ];
```

Performance on my Laptop:

1.01 GFLOPs

A simple Matrix–Matrix product

Given two NxN (N=1024) matrices A and B, compute their product C = A*B.

Naïve C code:

```
for ( size_t i = 0; i < n; ++i )
for ( size_t j = 0; j < n; ++j )
for ( size_t k = 0; k < n; ++k )
    C[ i + j*n ] += A[ i + k*n ]*B[ k + j*n ];
```

Performance on my Laptop:

1.01 GFLOPs

Corresponding oneMKL Call:

```
cblas_dgemm( CblasColMajor, CblasNoTrans, CblasNoTrans,
             n, n, n, 1.0, A, n, B, n, 0.0, C, n );
```

A simple Matrix–Matrix product

Given two NxN (N=1024) matrices A and B, compute their product C = A*B.

Naïve C code:

```
for ( size_t i = 0; i < n; ++i )
for ( size_t j = 0; j < n; ++j )
for ( size_t k = 0; k < n; ++k )
    C[ i + j*n ] += A[ i + k*n ]*B[ k + j*n ];
```

Performance on my Laptop:

1.01 GFLOPs

Corresponding oneMKL Call:

```
cblas_dgemm( CblasColMajor, CblasNoTrans, CblasNoTrans,
             n, n, n, 1.0, A, n, B, n, 0.0, C, n );
```

193.3 GFLOPs

oneMKL, DGEMM, C API

$$C \leftarrow \alpha AB + \beta C$$

```
int main() {

    int64_t m = 10, n = 6, k = 8, lda = 12, ldb = 8, ldc = 10;
    int64_t sizea = lda * k, sizeb = ldb * n, sizec = ldc * n;
    double alpha = 1.0, beta = 0.0;


    // Allocate matrices
    double *A = (double *) mkl_malloc(sizeof(double) * sizea);
    double *B = (double *) mkl_malloc(sizeof(double) * sizeb);
    double *C = (double *) mkl_malloc(sizeof(double) * sizec);

    // Initialize matrices [...]
    ...
    cblas_dgemm(CblasColMajor, CblasNoTrans, CblasNoTrans, m, n, k,
            alpha, A, lda, B, ldb, beta, C, ldc);
    ...
}
```

oneMKL, DGEMM, SYCL API

$$C \leftarrow \alpha AB + \beta C$$

```
using namespace oneapi::mkl::blas
int main(....) {

   int64_t m = 10, n = 6, k = 8, lda = 12, ldb = 8, ldc = 10;
   int64_t sizea = lda * k, sizeb = ldb * n, sizec = ldc * n;
   double alpha = 1.0, beta = 0.0;

   sycl::queue Q{sycl::gpu_selector{}};

   // Allocate matrices
   double *A = malloc_shared<double>(sizea, Q);
   double *B = malloc_shared<double>(sizeb, Q);
   double *C = malloc_shared<double>(sizec, Q);

   // Initialize matrices [...]
   ...

   auto e = gemm(queue, transpose::N, transpose::N, m, n, k,

                        alpha, A, lda, B, ldb, beta, C, ldc));

   ...
}
```

Set up GPU queue

Allocate CPU/GPU-accessible shared memory

New oneMKL DPC++ API

Computation is performed on given queue

Output **e** is a sycl::event object representing command completion

Call **`e.wait()`** to wait for completion

Different Ways of using oneMKL

Interfaces to the Intel® oneAPI Math Kernel Library

Operating Systems: Windows and various Linux flavours

Intel CPUs:

- C, C++, and Fortran interfaces
- Support for 32- and 64-bit integers
- Multi-threaded and single-threaded versions
- Controlled through linker flags and environment variables
- Static and dynamic linking possible
- Can be used with other compilers as well

Intel GPUs:

- SYCL interface
- OpenMP 5.0 support for GPU offloading, can be used from both C and Fortran

Interfaces to the Intel® oneAPI Math Kernel Library

Operating Systems: Windows and various Linux flavours

Intel CPUs:

- C, C++, and Fortran interfaces
- Support for 32- and 64-bit integers
- Multi-threaded and single-threaded versions
- Controlled through linker flags and environment variables
- Static and dynamic linking possible
- Can be used with other compilers as well

Intel GPUs:

- SYCL interface
- OpenMP 5.0 support for GPU offloading, can be used from both C and Fortran

Many different combinations!
Use the link-line advisor for help!

The oneMKL Interfaces Project

What is it?

- Implements (parts of) the oneMKL SYCL* API for Nvidia* and AMD* GPUs
- Wrapper around the vendor-supplied libraries (e.g., cuBLAS*, ...)
- Allows you to use the oneMKL APIs on all common GPUs and with a single source code. Only compiler and linker flags change!

Where do I get it?

On GitHub

oneAPI MKL Interfaces Project

			Type	Compiler
BLAS	x86 CPU	Intel(R) oneAPI Math Kernel Library	Dynamic, Static	DPC++, LLVM*, hipSYCL
	Intel GPU		Dynamic, Static	DPC++
	NVIDIA GPU	NVIDIA cuBLAS	Dynamic, Static	LLVM*, hipSYCL
	x86 CPU	NETLIB LAPACK	Dynamic, Static	DPC++, LLVM*, hipSYCL
	AMD GPU	AMD rocBLAS	Dynamic, Static	LLVM*, hipSYCL
	x86 CPU, Intel GPU, NVIDIA GPU, AMD GPU	SYCL-BLAS	Dynamic, Static	DPC++, LLVM*
LAPACK	x86 CPU	Intel(R) oneAPI Math Kernel Library	Dynamic, Static	DPC++, LLVM*
	Intel GPU		Dynamic, Static	DPC++
	NVIDIA GPU	NVIDIA cuSOLVER	Dynamic, Static	LLVM*
	AMD GPU	AMD rocSOLVER	Dynamic, Static	LLVM*
RNG	x86 CPU	Intel(R) oneAPI Math Kernel Library	Dynamic, Static	DPC++, LLVM*, hipSYCL
	Intel GPU		Dynamic, Static	DPC++
	NVIDIA GPU	NVIDIA cuRAND	Dynamic, Static	LLVM*, hipSYCL
	AMD GPU	AMD rocRAND	Dynamic, Static	LLVM*, hipSYCL
DFT	Intel GPU	Intel(R) oneAPI Math Kernel Library	Dynamic, Static	DPC++
	x86 CPU		Dynamic, Static	DPC++
	NVIDIA GPU	NVIDIA cuFFT	Dynamic, Static	DPC++

What's new in the Intel® oneAPI Math Kernel Library (oneMKL)

oneMKL on 4th Gen Intel® Xeon® Scalable processors

Maximize performance with oneMKL, unleashing the power of built-in accelerators

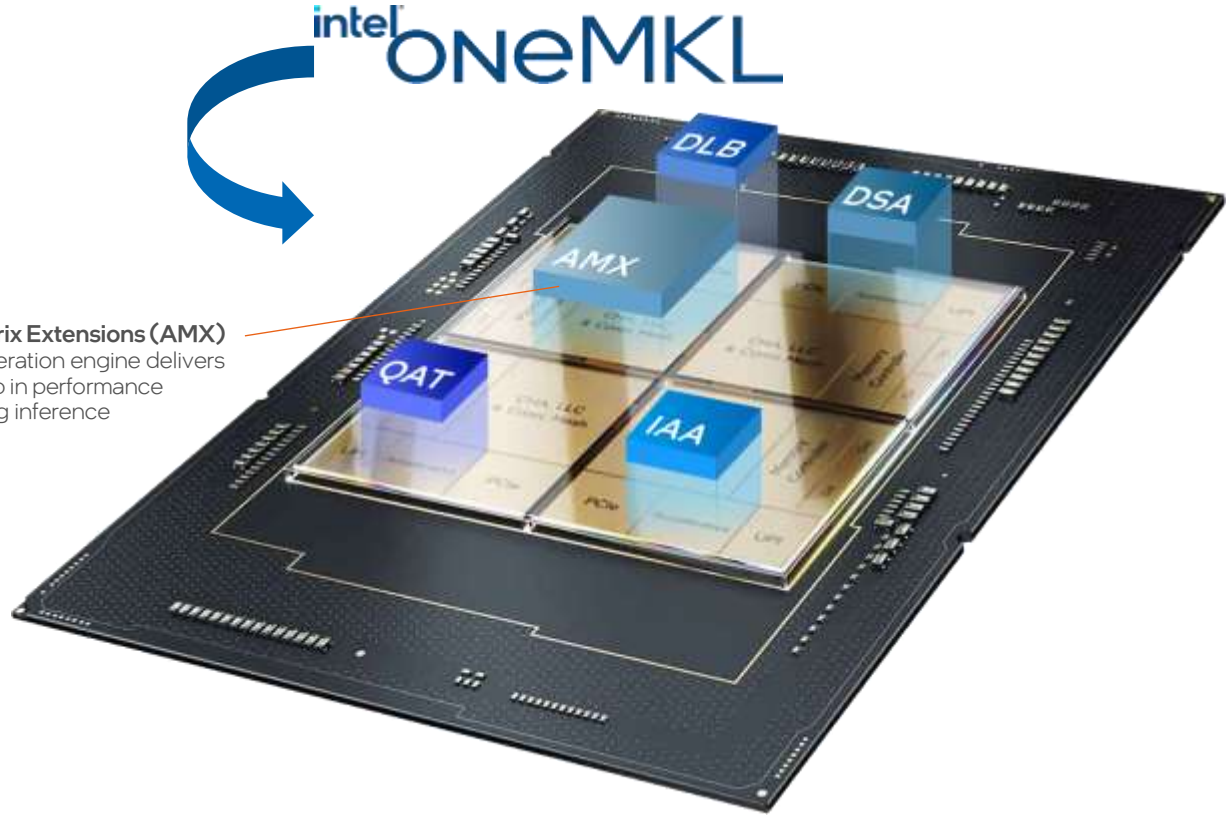
- The Intel® oneAPI Math Kernel Library (oneMKL) leverages Intel® AMX-Advanced Matrix eXtensions to optimize matrix computations for the BF16 and INT8 data types.
- oneMKL also leverages Intel ® AVX-512 -Advanced Vector Extensions for the FP16 data type on 4th Gen Intel® Xeon® Scalable processors.
- Most oneMKL memory- bound dense and sparse linear algebra (BLAS, LAPACK, sparse direct solvers), FFT, vector math, vector RNG, summary statistics, or spline computations, directly benefit from the onboard High Bandwidth Memory (HBM).

intel XEON SILVER

intel XEON GOLD

intel XEON PLATINUM

Advanced Matrix Extensions (AMX)
Built-in AI acceleration engine delivers a significant leap in performance for deep learning inference and training.

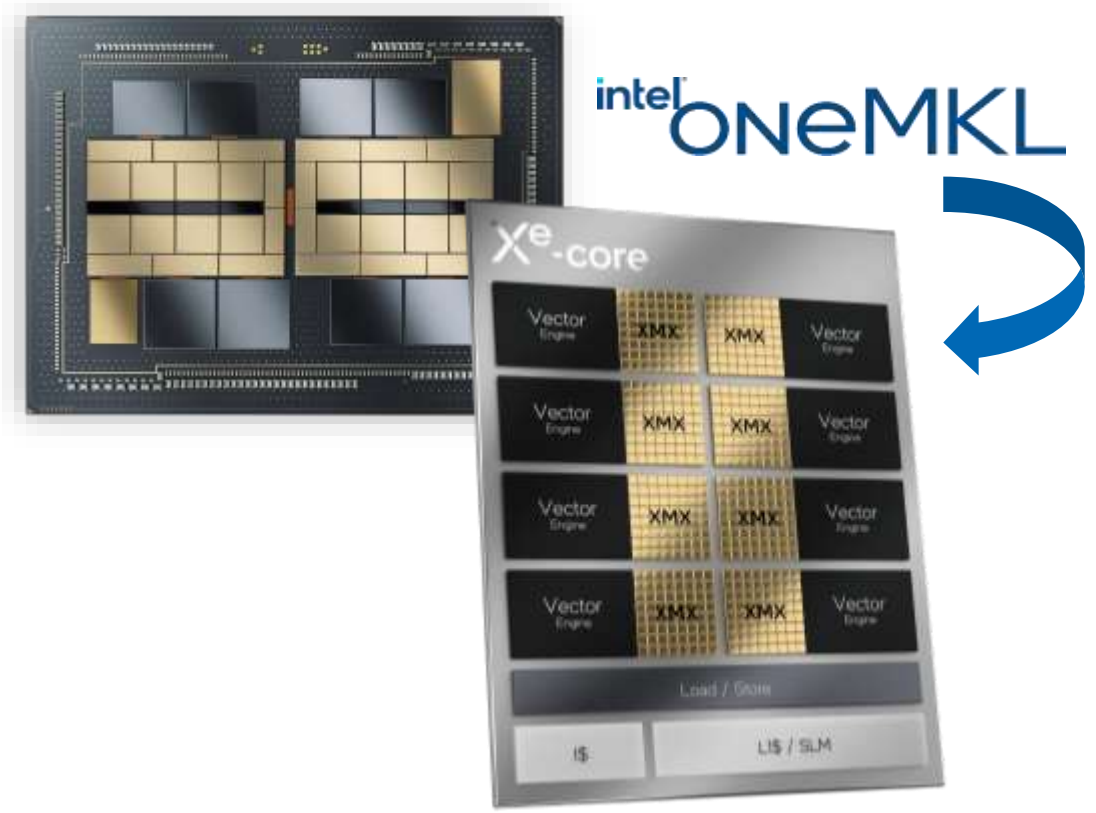


4th Gen Intel® Xeon® Scalable Processors with Intel® Advanced Matrix Extensions, Quick assist Technology, Intel® AVX-512, bfloat16, and more built-in accelerators

oneMKL on Intel® Data Center GPU Max Series

Breakthrough Performance for HPC and AI

- The Intel® oneAPI Math Kernel Library (oneMKL) leverages Intel® Xe Matrix Extensions (Intel® XMX) to optimize matrix computations for TF32, FP16, BF16 and INT8 data types on Intel® Data Center GPU Max Series (codenamed Ponte Vecchio).
- oneMKL provides a variety of dense and sparse linear algebra (BLAS, LAPACK, sparse BLAS), FFT, vector math, vector RNG, summary statistics, and spline interfaces both for the SYCL and C/Fortran OpenMP* offload programming models to enable applications targeting Intel® Data Center GPUs.



Intel® Data Center GPUs with hardware AV1 encode and Max with datatype flexibility, Intel® Xe Matrix Extensions, vector engine, XE-Link, and other features

oneMKL BLAS Compute Modes

- Core idea: use two or three FP16 numbers to obtain single precision (FP32)
  - “Concatenate” the mantissæ to achieve higher accuracy.
  - Warning: Not IEEE-754 conforming!

  However:
  - Can use hardware acceleration for FP16 to boost performance where high accuracy is not important (e.g., certain iterative algorithms)
  - Mostly benefits dense linear algebra operations
  - Supported in some BLAS routines

oneMKL, BLAS, GPU, the new compute modes

- The new MKL_BLAS_COMPUTE_MODE is intended for quickly evaluating whether alternate compute modes provide performance benefits and acceptable accuracy for an application. After initial testing, alternate mode settings can be permanently applied within the application using the per-call or per-source-file APIs.
- New alternate computation mode functionality for Level-3 routines
  - Better performance
  - Reduced accuracy
  - oneMKL does not enable any alternate compute modes by the default
  - The same or optional interfaces
  - Limitations: gemm, gemmt, syrk and syrk2 (MKL 2023)

oneMKL, BLAS, GPU, the new compute modes, cont.

```
from blas.hpp
#define ONEMKL_DECLARE_GEMM(Ta, Tb, Tc, Ts) \
DLL_EXPORT sycl::event gemm(sycl::queue &queue, transpose transa, transpose transb, \
                            std::int64_t m, std::int64_t n, std::int64_t k, \
                            Ts alpha, const Ta *a, std::int64_t lda, \
                            const Tb *b, std::int64_t ldb, \
                            Ts beta, Tc *c, std::int64_t ldc, \
                            compute_mode mode, 
                            const std::vector<sycl::event> &dependencies = {}); \
ONEMKL_INLINE_DECLARE sycl::event gemm(sycl::queue &queue, transpose transa, transpose transb, \
                              std::int64_t m, std::int64_t n, std::int64_t k, \
                              Ts alpha, const Ta *a, std::int64_t lda, \
                              const Tb *b, std::int64_t ldb, \
                              Ts beta, Tc *c, std::int64_t ldc, \
                              const std::vector<sycl::event> &dependencies = {}) \
{ \
   return gemm(queue, transa, transb, m, n, k, alpha, a, lda, b, ldb, beta, c, ldc, MKL_BLAS_COMPUTE_MODE, dependencies); \
}
```

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.

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


intel

Demo, MKL

MKL Verbose mode, cont.

Examples:

MKL_VERBOSE oneMKL 2023.0 Product build 20221128 for Intel(R) 64 architecture Intel(R) Advanced Vector Extensions 512 (Intel(R) AVX-512) with Intel(R) DL Boost, bfloat16 support, **Intel(R) AMX with bfloat16 and 8-bit integer support, and FP16 instructions, Lnx 3.00GHz ilp64** intel_threadMKL_VERBOSE

GEMM_BF16BF16F32(N,N,1024,1024,1024,0x7ffeb4e10028,0x7ff21d2b5080,1024,0x7ff21d4b6080,1024,0x7ffeb4e10030,0x7ff21ceb4080,1024) 69.71ms CNR:OFF Dyn:1 FastMM:1 TID:0

NThr:112FFT(dcbi5x13x7,tLim:22,desc:0xeb6d80) 93.11us CNR:OFF Dyn:1 FastMM:1 TID:0 NThr:44

MKL GPU Verbose mode, cont.

GPU Verbosing – extension of the existing env variable and run time functions

To change the verbose mode, do one of the following:

- set the environment variable *MKL_VERBOSE*

	CPU Targets	*GPU Targets*
(default) Set MKL_VERBOSE to 0	to disable verbose	to disable verbose
Set MKL_VERBOSE to 1	to enable verbose	to enable verbose without timing
Set MKL_VERBOSE to 2	to enable verbose	to enable verbose with synchronous timir

- Or call the support function mkl_verbose(int mode)

	CPU Targets	*GPU Targets*
(default) Call mkl_verbose(0)	to disable verbose	to disable verbose
Call mkl_verbose(1)	to enable verbose	to enable verbose without timing
Call mkl_verbose(2)	to enable verbose	to enable verbose with synchronous timir

MKL GPU Verbose mode, cont.

export MKL_VERBOSE=2

```
Running tests on GPU.
        Running with double precision complex-to-complex 1-D FFT:
        Using SYCL buffers
MKL_VERBOSE oneMKL 2022.0 Product build 20211022 for Intel(R) 64 architecture Intel(R)
MKL_VERBOSE Detected GPU0 Intel(R)_Gen9 Backend:Level_Zero VE:72 Stack:1 maxWGsize:256

MKL_VERBOSE FFT(dcfil6,bScale:0.0625) 88.88ms GPU0
                Verify the result, errthr = 4.44089e-15
                Verified, maximum error was 1.9405e-16
MKL_VERBOSE FFT(dcbil6,bScale:0.0625) 179.05ms GPU0
                Verify the result, errthr = 4.44089e-15
                Verified, maximum error was 6.93889e-18
        Using USM
MKL_VERBOSE FFT(dcfil6,bScale:0.0625) 191.91us GPU0
                Verify the result, errthr = 4.44089e-15
                Verified, maximum error was 1.9405e-16
MKL_VERBOSE FFT(dcbil6,bScale:0.0625) 204.91us GPU0
                Verify the result, errthr = 4.44089e-15
                Verified, maximum error was 6.93889e-18
        Test Passed
```

DEMO

DEMO, BLAS

..\2023\ww23_LRZ\DEMO:

PART I

- Reviewing the examples.
- Reviewing the makefiles. : -lmkl_sycl -lmkl_intel_ilp64 -lmkl_tbb_thread -lmkl_core -lsycl -lOpenCL -lpthread -lm -ldl

PART II

- make usm comp <-- Compilation
- Verbosing -- clarification

DEMO, BLAS, cont.

3. Small sizes, Conclusion
4. C/Fortran vs DPCPP performance
5. sequential Code: make seq
6. ZE_AFFINITY_MASK=0.0, 0.1

6.1 → BLAS, 2 sub-slises.

6.2 BLAS example – 2 slices….

6.3 BLAS. Splitting works. 2 tiles case.

7. Backends, comparisons
8. BLAS. New Compute Mode

DEMO, BLAS, newCompute mode.

./a.out 3000 0

size == 3000, GFlops == 4003.065 → host (SPR), default call

./a.out 3000 1

size == 3000, GFlops == 14863.280 -> PVC, default call, no Compute Mode

MKL_BLAS_COMPUTE_MODE=FLOAT_TO_BF16 ./a.out 3000 1

size == 3000, GFlops == 53650.547 -> PVC, Compute mode enabled by Env. Variables.

$ **./a1.**out 3000 1 1

size == 3000, GFlops == 53881.142 -> PVC, compute mode is enabled explicitly by dgemm(...oneapi::mkl::blas::compute_mode::float_to_bf16)

DEMO, BLAS, cont.

9. Offloading

Demo, tips, ZE_AFFINITY_MASK

- ZE_AFFINITY_MASK. Level_Zero sycl specification.
- sdp4451 – 1 device, 2 tiles.
-

- *0, 1*: all devices and sub-devices are reported (same as default)
- *0*: only device 0 is reported;with all its sub-devices
- *1*: only device 1 is reported as device 0; with all its sub-devices
- *0.0*: only device 0, sub-device 0 is reported as device 0
- *1.1, 1.2*: only device 1 is reported as device 0; with its sub-devices 1 and 2 reported as sub-devices 0 and 1, respectively
- *0.2, 1.3, 1.0, 0.3*: both device 0 and 1 are reported; device 0 reports sub-devices 2 and 3 as sub-devices 0 and 1, respectively; device 1 reports sub-devices 0 and 3 as sub-devices 0 and 1, respectively; the order is unchanged.

Environment Variables

The following table documents the supported knobs for overriding default functional behavior.

Category	Name	Values	Description
Device	ZE_AFFINITY_MASK	list	Forces driver to only report devices (and sub-devices) as specified by values
	ABLE_PCI_ID_DEVICE_ORDER	{**0**, 1}	Forces driver to report devices from lowest to highest PCI bus ID
	ARED_FORCE_DEVICE_ALLOC	{**0**, 1}	Forces all shared allocations into device memory

https://spec.oneapi.io/level-zero/latest/core/PROG.html

Demo, BLAS, multi-stack, explicit method

```
// #1 -- Create sub devices for multi-stack device
std::vector<sycl::device> subdev = {};
cl::sycl::queue subdev_queue0;
cl::sycl::queue subdev_queue1;

// # 2 -- Create context, execution queue, and buffers of matrix data
 subdev_queue0 = cl::sycl::queue(cxt, subdev[0], exception_handler);
 subdev_queue1 = cl::sycl::queue(cxt, subdev[1], exception_handler);
```

```
try {
    if ( multistack_mode == 1 && nb_device > 1 ){
        // Split B and C for subdevices
        n_subdev = n/2;
        sizeb_subdev = ldB * n_subdev;
        sizec_subdev = ldC * n/2;
        gemm_done0 = gemm(subdev_queue0, transA, transB, m, n_subdev, k, a
        gemm_done1 = gemm(subdev_queue1, transA, transB, m, n_subdev, k, a
                          A[1], ldA, B[1] + sizeb_subdev, ldB, beta, C[1] +
    } else {
        gemm_done0 = oneapi::mkl::blas::gemm(subdev_queue0, transA, transB
                          A[0], ldA, B[0], ldB, beta, C[0], ldC, gemm_depend
    }
}
```

Demo, BLAS, multi-stack, explicit method. cont.

./usm2stacks.x 4000 1

```
gfedorov@sdp4451:~/5temp/ww22_2023$ MKL_VERBOSE=2 ./usm2stacks.x 4000 1
 #of divices == 2
MKL_VERBOSE oneMKL 2023.0 Update 1 Product build 20230303 for Intel(R) 64 architecture Intel(R) Advanced Vector
 Extensions 512 (Intel(R) AVX-512) with Intel(R) DL Boost, bfloat16 support, and FP16 instructions, Lnx 1.80GHz
 ilp64 tbb_thread
MKL_VERBOSE Detected GPU0 Intel(R)_Xe_HPC Backend:Level_Zero VE:1024 Stack:2 maxWGsize:1024

MKL_VERBOSE oneapi::mkl::blas::column_major::gemm[double](0x7ffd64614010,NonTranspose,NonTranspose,4000,2000,40
00,2,0x14c053400000,4000,0x14c04b600000,4000,3,0x14c043800000,4000,unset,Vector<sycl::_V1::event>OfSize:0) mode
:standard host:545.88ms device:nan GPU0
MKL_VERBOSE oneapi::mkl::blas::column_major::gemm[double](0x7ffd64614040,NonTranspose,NonTranspose,4000,2000,40
00,2,0x14c03ba00000,4000,0x14c037909000,4000,3,0x14c02fb09000,4000,unset,Vector<sycl::_V1::event>OfSize:0) mode
:standard host:490.54ms device:nan GPU0
```

#of devices == 2

GPU Explicit, perf = 36252.8 GFlops

DEMO, tips, BACKEND

- SYCL_DEVICE_FILTER=level_zero MKL_VERBOSE=2 ./usm.x 4000 1
- MKL_VERBOSE Detected GPU0 Intel(R)_Xe_HPC Backend:Level_Zero VE:1024 Stack:2 maxWGsize:1024

- SYCL_DEVICE_FILTER=opencl MKL_VERBOSE=2 ./usm.x 4000 1
- MKL_VERBOSE Detected GPU0 Intel(R)_Xe_HPC Backend:OpenCL VE:10
- size == 4000, GFlops == 9634.553

Demo, BLAS, GPU, the new compute modes

Compute mode enum value	Environment variable	Description
compute_mode::float_to_bf16	FLOAT_TO_BF16	Convert single-precision inputs to bfloat16 format internally; Output is accumulated in single precision. Accuracy will be reduced, with possibly much higher performance.
compute_mode::float_to_bf16x2	FLOAT_TO_BF16X2	Convert each single-precision input value to a sum of two bfloat16 values internally; Output is accumulated in single precision. Expected accuracy is between that of standard single precision arithmetic and bfloat16 arithmetic.
compute_mode::float_to_bf16x3	FLOAT_TO_BF16X3	Convert each single-precision input value to a sum of three bfloat16 values internally; Output is accumulated in single precision. Expected accuracy is comparable to standard single precision arithmetic in most cases.
…….	…….	…….
compute_mode::complex_3m	COMPLEX_3M	Reduce the four real multiplications in a standard complex multiplication to three real multiplications. Expected accuracy is comparable to standard arithmetic in most cases.
compute_mode::standard	STANDARD	Do not allow any alternate compute modes

Demo, BLAS, GPU, the new compute modes, cont.

- Usage :
  - Environment Variables:

    set/export MKL_BLAS_COMPUTE_MODE=FLOAT_TO_BF16X2
  - Per-call or per-source-file mode:

```
using oneapi::mkl::blas; auto mode_settings = compute_mode::float_to_bf16x2 | compute_mode::float_to_tf32; /* allow either of these two modes */

// USM API, with dependencies :
    syrk(my_queue, n, k, uplo, trans, alpha, a_ptr, lda, beta, c_ptr, ldc, compute_mode::float_to_bf16, event );
// USM API, dependencies but no special compute_mode settings
    syrk(my_queue, n, k, uplo, trans, alpha, a_ptr, lda, beta, c_ptr, ldc, event );
```

  - Checking Which Mode Is Used:

```
MKL_VERBOSE oneapi::mkl::blas::column_major::gemm[float](0x7ffd39046350,...,float_to_bf16)
mode:float_to_bf16 host:nan device:nan GPU0
```

https://www.intel.com/content/www/us/en/develop/documentation/oneapi-mkl-dpcpp-developer-reference/top/blas-routines/blas-compute-modes.html

Demo, BLAS, Compute mode, cont

- Check sgemm_newcomputemode.cpp, makefile, run_newcompmode.sh
  - The same code
  - The same compiling
  - run_newcompmode.sh

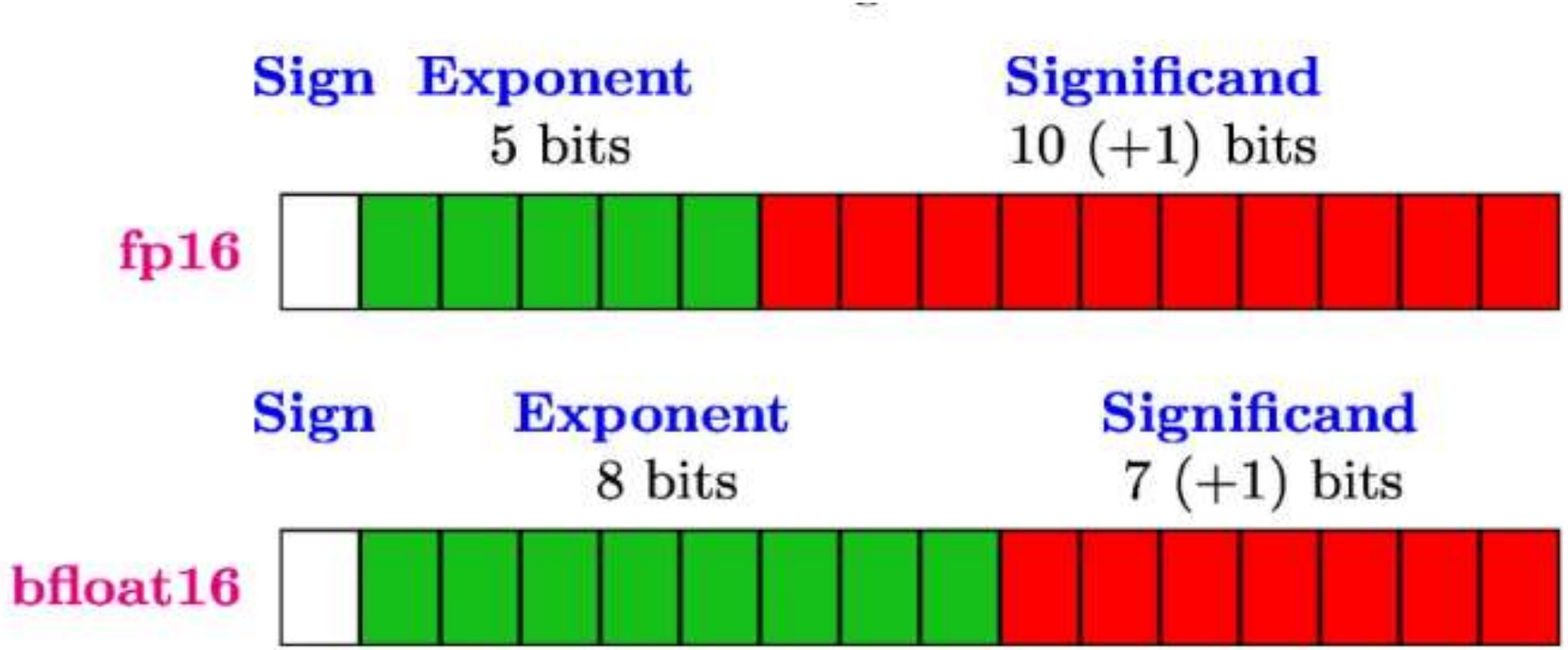
```
#!/bin/bash
size=$1
mode=$2

#echo " std :"
./a.out $size $mode
#echo " BF : "
MKL_BLAS_COMPUTE_MODE=FLOAT_TO_BF16 ./a.out $size $mode
#echo " BFX2 :"
MKL_BLAS_COMPUTE_MODE=FLOAT_TO_BF16X2 ./a.out $size $mode
#echo " BFX3 : "
MKL_BLAS_COMPUTE_MODE=FLOAT_TO_BF16X3 ./a.out $size $mode
```

Demo, BLAS, Compute mode, cont

```
gfedorov@sdp4451:~/5temp/ww22_2023/Compute_mode$ ./run_newcompmode.sh 2000 1
        size == 2000, GFlops  == 26092.908
        size == 2000, GFlops  == 30878.479
        size == 2000, GFlops  == 29908.475
        size == 2000, GFlops  == 27725.575
gfedorov@sdp4451:~/5temp/ww22_2023/Compute_mode$ ./run_newcompmode.sh 3000 1
        size == 3000, GFlops  == 14988.770
        size == 3000, GFlops  == 55078.302
        size == 3000, GFlops  == 28512.450
        size == 3000, GFlops  == 12226.104
```

Bfloat16 Arithmetic



Bfloat16 Arithmetic, cont.

	Bits (significand, exponent)	Unit roundoff	Min positive subnormal	Min positive normalized	Max finite
bfloat16	(8, 8)	3.91×10^{-3}	9.18×10^{-41}	1.18×10^{-38}	3.39×10^{38}
fp16	(11, 5)	4.88×10^{-4}	5.96×10^{-8}	6.10×10^{-5}	6.55×10^{4}
fp32	(24, 8)	5.96×10^{-8}	1.40×10^{-45}	1.18×10^{-38}	3.40×10^{38}
fp64	(53, 11)	1.11×10^{-16}	4.94×10^{-324}	2.22×10^{-308}	1.80×10^{308}

Bfloat16 Arithmetic, cont.

Dynamic range

- fp32
- fp16
- bf16