oneMKL Technical Advisory Board

Session 10

January 27, 2021

Agenda

- Welcoming remarks – 5 minutes
- Updates from last meeting – 5 minutes
- Overview of oneMKL Batched Linear Algebra - Louise Huot and Rachel Ertl (30 minutes)
- Wrap-up and next steps – 5 minutes

Updates from last meeting

- oneAPI Math Kernel Library (oneMKL) Interfaces Project
  - Enabled building/testing for selected domains (currently BLAS and RNG)
- Preview of oneDNN Graph specification
- Intel® oneAPI Math Kernel Library (oneMKL) version 2021.1 released:
  - Part of the Intel® oneAPI Base Toolkit
- oneAPI GPU Optimization Guide available

Overview of oneMKL Batched Linear Algebra

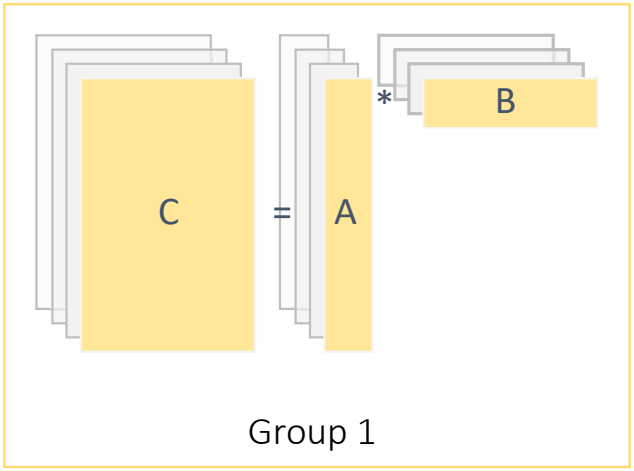
Batching Overview

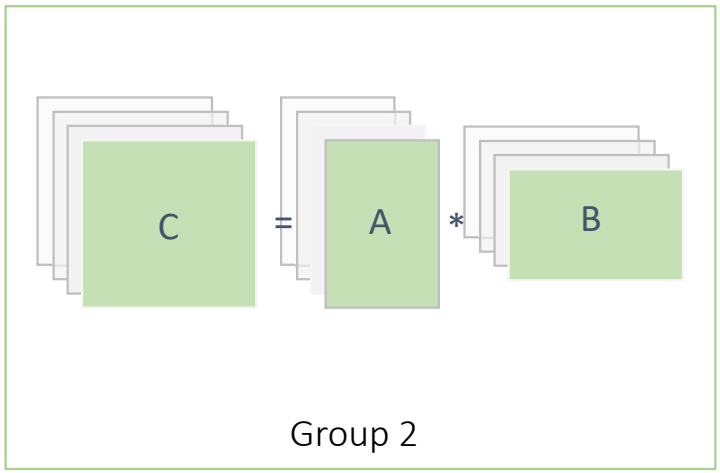
- Execute multiple **independent** operations of the same type in a single call
  - e.g. invert 100 different 8x8 matrices
- Benefits: increased parallelism, reduced overhead
- Batch functionality in the oneMKL DPC++ specification:
  - BLAS: gemm, trsm, axpy
  - LAPACK*: LU (getrf, getri, getrs), Cholesky (potrf, potrs), QR (geqrf, orgqr, ungqr)
  - FFT: all DFTs
- BLAS/LAPACK have two batching options: group API (independent pointers) vs strided API (one buffer with constant stride)

BLAS/LAPACK Group APIs

Group = set of operations with identical parameters (size, transpose…)
but different matrix/vector data

Group batch APIs process one or more groups simultaneously.





BLAS/LAPACK Group APIs

Group = set of operations with identical parameters (size, transpose…)
but different matrix/vector data

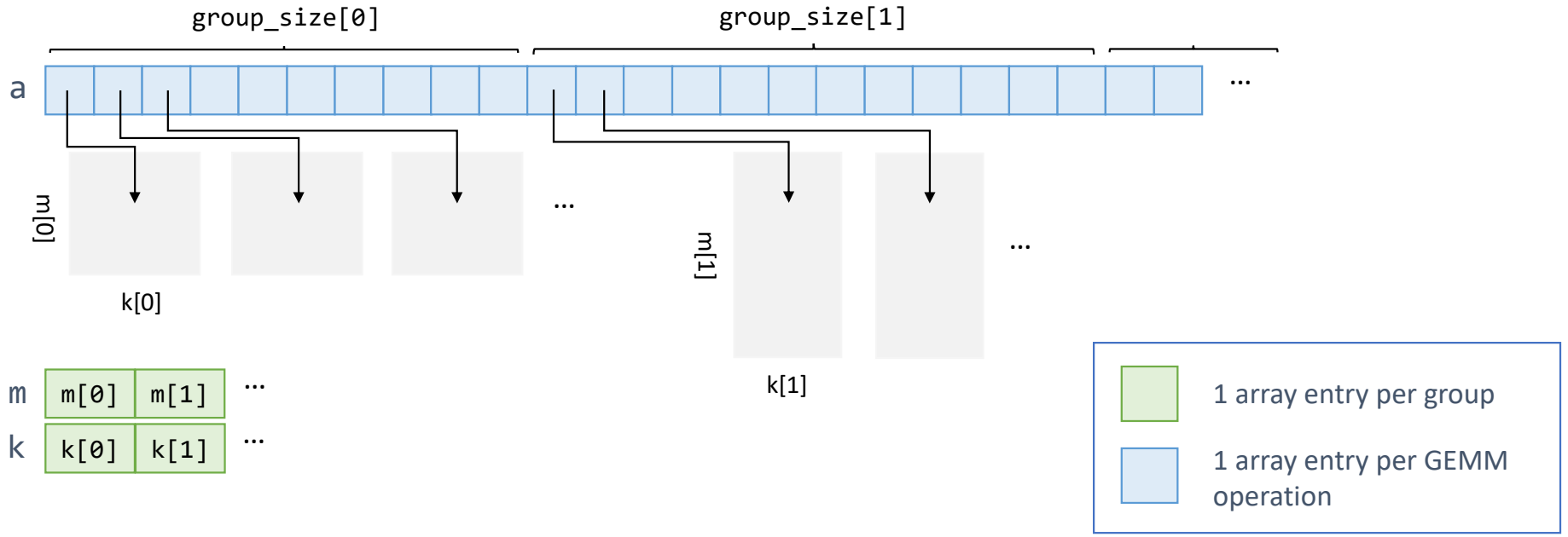
Group batch APIs process one or more groups simultaneously.

Examples:

- *n* operations, 1 group: all parameters identical
- *n* operations, *n* groups: each operation has different parameters

Example: Batch GEMM – Group API USM only

```
sycl::event oneapi::mkl::blas::{column,row}_major::gemm_batch(sycl::queue &queue,
                                                              onemkl::transpose *transa, onemkl::transpose *transB,
                                                              std::int64_t *m, std::int64_t *n, std::int64_t *k,
                                                              fp_type *alpha, const fp_type **a, std::int64_t *lda,
                                                              const fp_type **b, std::int64_t *ldb,
                                                              fp_type *beta, fp_type **c, std::int64_t *ldc,
                                                              std::int64_t num_groups, std::int64_t *group_sizes,
                                                              const sycl::vector_class<sycl::event> &dependencies = {});
```



BLAS/LAPACK Strided DPC++ APIs

- DPC++ oneMKL adds strided APIs for simple batch cases (similar to batch DFT)
  - **Single group**: all matrix/vector sizes, parameters are homogeneous
  - **Fixed stride** between successive matrices/vectors in batch
    - Base address + stride replaces array of pointers
    - Strides on inputs may be zero to reuse an input for all operations in the batch
    - Zero stride are not valid for the output matrices/vectors
  - Support for both **DPC++ buffer** and **USM pointers**
  - Same API as non-batch function + matrices/vectors stride and batch size integer inputs

```
void oneapi::mkl::lapack::getrf      (cl::sycl::queue &queue, std::int64_t m, std::int64_t n,
    cl::sycl::buffer<T> &a, std::int64_t lda,
    cl::sycl::buffer<std::int64_t> &ipiv,
    cl::sycl::buffer<T> &scratchpad, std::int64_t scratchpad_size)
void oneapi::mkl::lapack::getrf_batch(cl::sycl::queue &queue, std::int64_t m, std::int64_t n,
    cl::sycl::buffer<T> &a, std::int64_t lda, std::int64_t stride_a,
    cl::sycl::buffer<std::int64_t> &ipiv, std::int64_t stride_ipiv, std::int64_t batch_size,
    cl::sycl::buffer<T> &scratchpad, std::int64_t scratchpad_size)
```

Strided Batch Snippet – LU USM API

```
#include “oneapi/mkl.hpp”
using namespace oneapi::mkl;
…
int64_t batch = 100;                // 100 matrices
int64_t n = 10;                     // Matrix size – square in this example
int64_t stride_a   = n * n;         // 10x10 matrices are contiguous in memory
int64_t stride_piv = n;             // Pivot entries also contiguous in memory

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

// Allocate memory for matrices and pivot indices, as well as scratch space.
auto a_array      = sycl::malloc_shared<double>(stride_a   * batch, Q);
auto pivot_array  = sycl::malloc_shared<double>(stride_piv * batch, Q);

auto scratch_size = lapack::getrf_batch_scratchpad_size(Q, n, n, n, stride_a, stride_piv, batch);
auto scratch      = sycl::malloc_shared<double>(scratch_size, Q);
…
// [Initialize A_array here]
…
// Batch computation
try {
   auto done = lapack::getrf_batch(Q, n, n, a_array, n, stride_a, pivot_array, stride_piv, scratch, scratch_size)
   done.wait();
} catch (const lapack::exception& e) { /* */ }
…
```

LAPACK Batch Exception

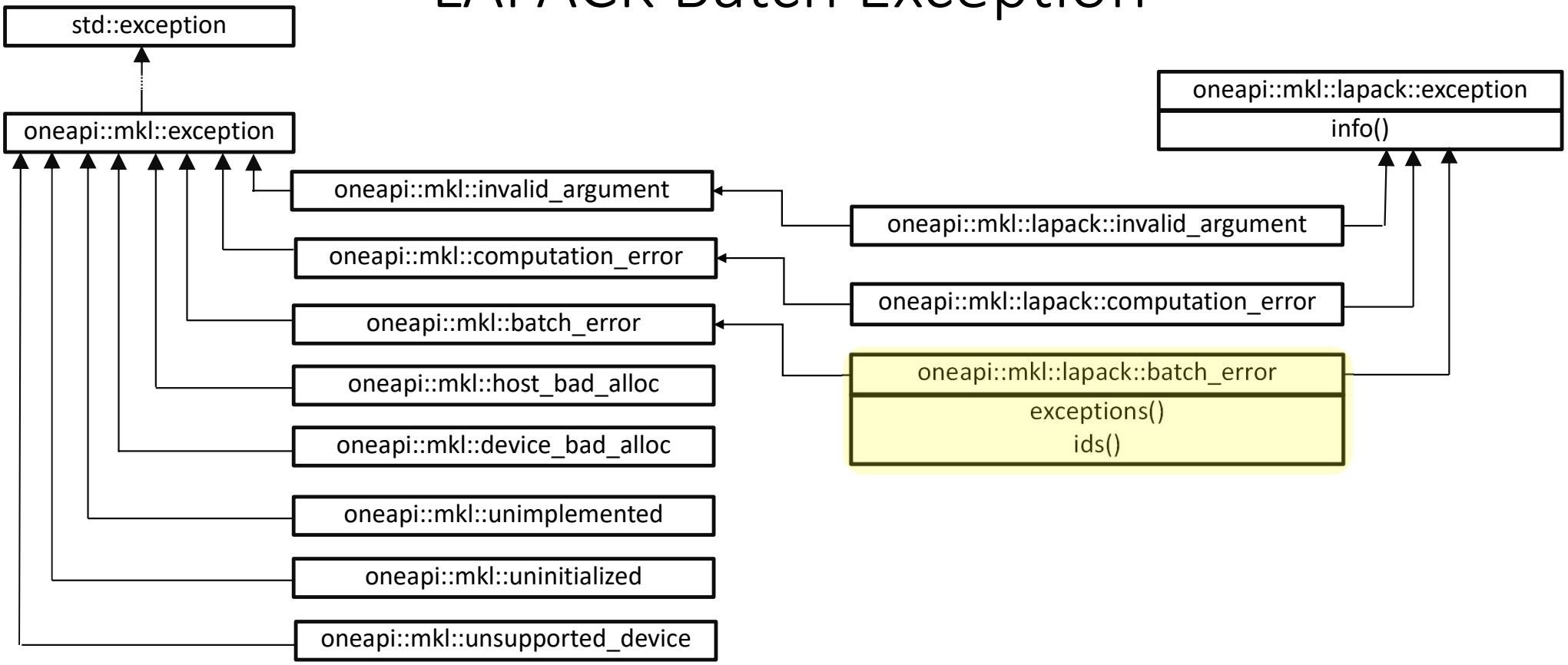
Common exceptions

Exception class	Description
`oneapi::mkl::exception`	Reports general unspecified problem
`oneapi::mkl::unsupported_device`	Reports a problem when the routine is not supported on a specific device
`oneapi::mkl::host_bad_alloc`	Reports a problem that occurred during memory allocation on the host
`oneapi::mkl::device_bad_alloc`	Reports a problem that occurred during memory allocation on a specific device
`oneapi::mkl::unimplemented`	Reports a problem when a specific routine has not been implemented for the specified parameters
`oneapi::mkl::invalid_argument`	Reports problem when arguments to the routine were rejected
`oneapi::mkl::uninitialized`	Reports problem when a handle (descriptor) has not been initialized
`oneapi::mkl::computation_error`	Reports any computation errors that have ccurred inside a oneMKL routine
`oneapi::mkl::batch_error`	Reports errors that have occurred inside a batch oneMKL routine

LAPACK specific exceptions

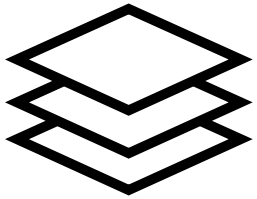
Exception class	Description
`oneapi::mkl::lapack::exception`	Base class for all LAPACK exceptions providing access to info code familiar to users of conventional LAPACK API. All LAPACK related exceptions can be handled with catch block for this class.
`oneapi::mkl::lapack::invalid_argument`	Reports errors when arguments provided to the LAPACK subroutine are inconsistent or do not match expected values. Class extends base `oneapi::mkl::invalid_argument` with ability to access conventional status info code.
`oneapi::mkl::lapack::computation_error`	Reports computation errors that have occurred during call to LAPACK subroutine. Class extends base `oneapi::mkl::computation_error` with ability to access conventional status info code familiar to LAPACK users.
`oneapi::mkl::lapack::batch_error`	Reports errors that have occurred during batch LAPACK computations. Class extends base `oneapi::mkl::batch_error` with ability to access individual exception objects for each of the issues observed in a batch and an info code. The info code contains the number of errors that occurred in a batch. Positions of problems in a supplied batch that experienced issues during computations can be retrieved with *ids()* method, and list of particular exceptions can be obtained with *exceptions()* method of the exception object. Possible exceptions for a batch are documented for corresponding non-batch API.

LAPACK Batch Exception

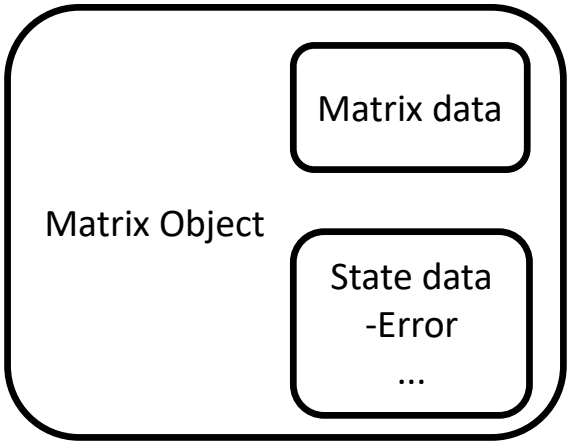


Matrix Object Encapsulation

Future possibilities – no concrete proposal



C++23 style mdspan interface
USM pointer interface
Actual memory layout



✘ Memory layout requirements may not be supported

✓ Utilizes support in C++ standard

✘ Errors may be associated with operation, not matrix

✓ Easier to keep track of info parameter

Batch API Extension Requests

Example axpy current API:

- strided API: `axpy_batch(sycl::queue&, std::int64_t, fp_type alpha, const fp_type*, std::int64_t, std::int64_t, fp_type*, std::int64_t, std::int64_t, std::int64_t, const sycl::vector_class<sycl::event>&)`
- group API: `axpy_batch(sycl::queue&, std::int64_t*, fp_type* alpha, const fp_type**, std::int64_t*, fp_type**, std::int64_t*, std::int64_t, std::int64_t*, const sycl::vector_class<sycl::event>&)`
  ➔ only group_count scalars stored in alpha

Interest in extending the batch API with

- Scalars passed by reference rather than value.
  - `axpy_batch(sycl::queue&, std::int64_t, fp_type* alpha, const fp_type*, std::int64_t, std::int64_t, fp_type*, std::int64_t, std::int64_t, std::int64_t, const sycl::vector_class<sycl::event>&)`
    ➔ only one scalar stored in alpha
  - `axpy_batch(sycl::queue&, std::int64_t*, fp_type** alpha, const fp_type**, std::int64_t*, fp_type**, std::int64_t*, std::int64_t, std::int64_t*, const sycl::vector_class<sycl::event>&)`
    ➔ only group_count pointers to scalar stored in alpha
- Scalars different for each computation.
  - `axpy_batch(sycl::queue&, std::int64_t, fp_type* alpha, const fp_type*, std::int64_t, std::int64_t, fp_type*, std::int64_t, std::int64_t, std::int64_t, const sycl::vector_class<sycl::event>&)`
    ➔ batch_size scalars stored in alpha
  - `axpy_batch(sycl::queue&, std::int64_t*, fp_type* alpha, const fp_type**, std::int64_t*, fp_type**, std::int64_t*, std::int64_t, std::int64_t*, const sycl::vector_class<sycl::event>&)`
    ➔ total batch size scalars stored in alpha
  - `axpy_batch(sycl::queue&, std::int64_t*, fp_type** alpha, const fp_type**, std::int64_t*, fp_type**, std::int64_t*, std::int64_t, std::int64_t*, const sycl::vector_class<sycl::event>&)`
    ➔ total batch size pointers to scalar stored in alpha

Current APIs do not allow such extensions as different semantics are conflicting with the same declarations

Batch API Extension Considerations

1. Additional input parameters to specify size of arrays
2. Additional input to specify stride for each array (could be 0 to have fixed batch)
3. New entry points
4. Use vector instead of pointers (similar to the proposal for SLATE) so the size can be queried and check
   - `axpy_batch(sycl::queue&, std::int64_t, const fp_type alpha, const fp_type*, std::int64_t, std::int64_t, fp_type*, std::int64_t, std::int64_t, std::int64_t, const std::vector<sycl::event>&)`
     Fixed strided batch
   - `axpy_batch(sycl::queue&, std::int64_t, const std::vector<fp_type> alpha, const fp_type*, std::int64_t, std::int64_t, fp_type*, std::int64_t, std::int64_t, std::int64_t, const std::vector<sycl::event>&)`
     Fixed strided batch with variable scalar, vector alpha must be of size 1 or batch_size
   - `axpy_batch(sycl::queue&, const std::vector<std::int64_t>, const std::vector<fp_type> alpha, const std::vector<fp_type*>, const std::vector<std::int64_t>, std::vector<fp_type*>, const std::vector<std::int64_t>, std::int64_t, const std::vector<std::int64_t>, const std::vector<sycl::event>&)`
     Variable group batch, vector size must be 1 or group_count or total_batch_size (except for the output vectors/matrices)
   - ***Additional overloads for alpha pointer or vector of pointers***
5. Matrix/tensor object encapsulation

Any other suggestions? Preference among the solutions for variable batch?
Currently leaning toward solution 4 or similar

Dense Linear Algebra Batch Support in Libraries

Library	Function	oneMKL Specification coverage
Intel® oneMKL	Group and strided API • C/Fortran/DPC++: gemm, axpy • DPC++: trsm, getrf, getri, getrs, potrf, potrs, geqrf, orgqr, ungqr	DPC++ group or strided API
cuBLAS	cublas?{gemm,trsm,getrf,getrs,getri,geqrf}Batched cublas?{gemm}StrideBatched	DPC++ group API with group_count = 1 or DPC++ strided API
	cublas?{matinv}Batched	getrf + getri DPC++ group API with group_count = 1
	cublas?gemm{Strided}BatchedEx, cublasHgemmBatched, cublas?{gels}Batched	None
MAGMA	magmablas_?{gemm,trsm,getrf,getrs,potrf,potrs,geqrf}_batched	DPC++ group API with group_count = 1
	Magma_?getri_outofplace_batched	DPC++ getri group API with group_count = 1 (in-place)
	magma_?{gemm,trsm,potrf,potrs}_vbatched	DPC++ group API, one computation per group
	magmablas_?{hemm,herk,her2k,symm,syrk,syr2k,trmm,geadd,gemv,hemv,symv,trsv,transpose,lacpy,gesv,get2f,posv}_{v}batched, magma_?{gesv,getf,getrf,getrs}_nopiv_batched	None
rocBLAS	rocblas_?{gemm, trsm, axpy}_{strided}_batched	DPC++ group API with group_count = 1 or DPC++ strided API
	rocblas_?{all other BLAS}_{strided}_batched	None

Dense Linear Algebra Batch Support in Libraries

Library	Function	oneMKL Specification coverage
Intel® oneMKL	Group and strided API • C/Fortran/DPC++: gemm, axpy • DPC++: trsm, getrf, getri, getrs, potrf, potrs, geqrf, orgqr, ungqr	DPC++ group or strided API
cuBLAS	cublas?{gemm,trsm,getrf,ge… cublas?{gemm}StrideBatch…	…with group_count = 1 or DPC++ strided API
	cublas?{matinv}Batched	…group API with group_count = 1
	cublas?gemm{Strided}Batc…	
MAGMA	magmablas_?{gemm,trsm,…	…th group_count = 1
	Magma_?getri_outofplace…	…API with group_count = 1 (in-place)
	magma_?{gemm,trsm,potrf,potrs}_vbatched	…one computation per group
	magmablas_?{hemm,herk,her2k,symm,syrk,syr2k,trmm,geadd,gemv,hemv,symv,trsv,transpose,lacpy,gesv,get2f,posv}_{v}batched, magma_?{gesv,getf,getrf,getrs}_nopiv_batched	None
rocBLAS	rocblas_?{gemm, trsm, axpy}_{strided}_batched	DPC++ group API with group_count = 1 or DPC++ strided API
	rocblas_?{all other BLAS}_{strided}_batched	None

Main gaps in the oneMKL specification

- Low/mixed precision GEMM
- All level3 BLAS
- Level2 BLAS: GEMV, SYMV/HEMV
- Matrix copy with transposition
- Non-pivoting LU (factorization, and solve)

Batch API future considerations summary

- Received several requests to extend batch support for BLAS and LAPACK in the Intel® oneAPI Math Kernel Library.

  Might consider extending the batch support in the DPC++ oneMKL specification to all relevant API ie most used functions: GEMM with lower/mixed precision, BLAS Level3, GEMV, SYMV, HEMV, DOT, COPY, SCAL, matrix copy with transpose, LQ factorization, GELS, select LAPACK (LU) non-pivoting version
- Extend variable batch support
- Matrix/Tensor object encapsulation (simplify API, improve error handling)

Next Steps

- Focuses for next meeting(s):
  - Sparse linear algebra
  - Discrete Fourier transforms
  - Any topics from oneMKL TAB members?

Resources

- oneAPI Main Page: https://www.oneapi.com/
- Latest release of oneMKL Spec (currently v. 1.0): https://spec.oneapi.com/versions/latest/elements/oneMKL/source/index.html
- GitHub for oneAPI Spec: https://github.com/oneapi-src/oneAPI-spec
- GitHub for oneAPI TAB: https://github.com/oneapi-src/oneAPI-tab

- GitHub for oneAPI Math Kernel Library (oneMKL) Interfaces (currently BLAS and RNG domains): https://github.com/oneapi-src/oneMKL