oneMKL Technical Advisory Board

Session 19

July 27, 2022

Agenda

- Welcoming remarks – 5 minutes
- Updates from last meeting – 5 minutes
- Overview of matrix transposition and copy routines – Andrew Barker (20 minutes)
- Wrap-up and next steps – 5 minutes

Updates from last meeting

- Will schedule a oneMKL TAB meeting on August 24 to cover new proposed DFT APIs to support user-allocated workspace

Overview of matrix transposition and copy routines

Matrix transposition and copy

- Matrix transposition is very common in applications but not explicitly supported in standard BLAS.
    - Out-of-place scaling and transposition ($B = \alpha A^T$),
    - In-place scaling and transposition ($A = \alpha A^T$),
    - Matrix addition/scaling ($C = \alpha \, op(A) + \beta \, op(B)$)
- The oneMKL product currently has several matrix transposition/copy APIs with scaling: *omatcopy*, *imatcopy*, *omatadd*, and batched variants.
- Similar functionality is provided in cuBLAS and rocBLAS by *geam*.

Matrix transposition and copy: motivation

- ($B = \alpha \, op(A)$, $A = \alpha \, op(A)$, $C = \alpha \, op(A) + \beta \, op(B)$)
- Functionality can be similar to BLAS scal, copy, and axpy, but it is convenient to have APIs that are aware of matrix size, row-major and column-major orderings, and leading dimension.
- Having matrix-based functions allows us to potentially optimize important use cases.
- At Intel we have had customer requests and interest in this functionality.

Vendor library interfaces and support

Use case	oneMKL product	cuBLAS	rocBLAS
$C = \alpha\, op(A) + \beta\, op(B)$	omatadd	geam(A, B, C, …)	geam(A, B, C, …)
$C = \alpha A^T$	omatcopy	geam(β=0)	geam(β=0)
$A = \alpha A^T$	imatcopy	Not supported	Not documented?
$C = \alpha C + \beta\, op(B)$	Not supported	geam(A=C)	Not documented?
Batched interfaces	Supported	Not supported	Supported

Interface comparison

```
sycl::event omatadd(sycl::queue &queue, transpose transa, transpose transb,
                    std::int64_t m, std::int64_t n,
                    float alpha, const float *a, std::int64_t lda,
                    float beta, const float *b, std::int64_t ldb,
                    float *c, std::int64_t ldc,
                    const std::vector<sycl::event> &dependencies = {});

cublasStatus_t cublasSgeam(cublasHandle_t handle,
                           cublasOperation_t transa, cublasOperation_t transb,
                           int m, int n,
                           const float *alpha, const float *A, int lda,
                           const float *beta, const float *B, int ldb,
                           float *C, int ldc);
```

Options for oneAPI spec

Use {i,o}matcopy/omatadd APIs

- Provide imatcopy, omatcopy, omatadd APIs with different signatures.
- Easy on-ramp for existing oneMKL CPU users.
- Quick implementation in oneMKL open source interfaces with oneMKL backend.

Use geam APIs

- Provide geam API with documented special cases when pointers to matrices are repeated as arguments.
- Easy on-ramp for existing cuBLAS and rocBLAS GPU users.

RFC, pull request, and implementation

- RFC is Issue #421 in oneAPI spec.
- Draft implementation using imatcopy, omatcopy, omatadd APIs in PR #202 in the oneMKL open source interfaces.
- PR #420 in the oneAPI spec for actual proposed changes to the spec.

Wrap-up

Next Steps

- Focuses for next meeting(s):
  - DFT APIs to support user-allocated workspace
  - Device APIs for BLAS
  - Any topics from oneMKL TAB members?

- If anyone has content that they would like posted on oneAPI.io, please let us know

Resources

- oneAPI Main Page: https://www.oneapi.io/
- Latest release of oneMKL Spec (currently v. 1.1): 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 open source oneMKL interfaces (currently BLAS, RNG, and LAPACK domains): https://github.com/oneapi-src/oneMKL

BACKUP

Existing oneMKL APIs (USM)

```
sycl::event imatcopy(sycl::queue &queue, transpose trans,
                     std::int64_t m, std::int64_t n, float alpha, float *ab,
                     std::int64_t lda, std::int64_t ldb,
                     const std::vector<cl::sycl::event> &dependencies = {});
sycl::event omatcopy(sycl::queue &queue, transpose trans,
                     std::int64_t m, std::int64_t n, float alpha, const float *a,
                     std::int64_t lda, float *b, std::int64_t ldb,
                     const std::vector<cl::sycl::event> &dependencies = {});
sycl::event omatadd(sycl::queue &queue, transpose transa, transpose transb,
                    std::int64_t m, std::int64_t n,
                    float alpha, const float *a, std::int64_t lda,
                    float beta, const float *b, std::int64_t ldb,
                    float *c, std::int64_t ldc,
                    const std::vector<cl::sycl::event> &dependencies = {});
```

Existing oneMKL APIs (buffer)

```
void imatcopy(sycl::queue &queue, transpose trans,
              std::int64_t m, std::int64_t n,
              float alpha, cl::sycl::buffer<float, 1> &ab,
              std::int64_t lda, std::int64_t ldb);
void omatcopy(sycl::queue &queue, transpose trans,
              std::int64_t m, std::int64_t n,
              float alpha, cl::sycl::buffer<float, 1> &a,
              std::int64_t lda, cl::sycl::buffer<float, 1> &b, std::int64_t ldb);
void omatadd(sycl::queue &queue, transpose transa, transpose transb,
             std::int64_t m, std::int64_t n,
             float alpha, cl::sycl::buffer<float, 1> &a, std::int64_t lda,
             float beta, cl::sycl::buffer<float, 1> &b, std::int64_t ldb,
             cl::sycl::buffer<float, 1> &c, std::int64_t ldc);
```

Existing oneMKL APIs (USM) (batched)

```
// strided
sycl::event imatcopy_batch(sycl::queue &queue, transpose trans,
                           std::int64_t m, std::int64_t n, float alpha, float *ab,
                           std::int64_t lda, std::int64_t ldb, std::int64_t stride,
                           std::int64_t batch_size,
                           const std::vector<sycl::event> &dependencies = {});
// group batch
sycl::event imatcopy_batch(sycl::queue &queue, const transpose *trans,
                           const std::int64_t *m, const std::int64_t *n, const float *alpha,
                           float **ab, const std::int64_t *lda, const std::int64_t *ldb,
                           std::int64_t group_count, const std::int64_t *groupsize,
                           const std::vector<sycl::event> &dependencies);
```