Math Special Interest Group

Session 4

September 20, 2023

Agenda

- Welcoming remarks – 3 minutes
- Updates from last meeting – 5 minutes
- Discussion on sparse BLAS APIs – Romain Biessy (30 minutes)
- value_or_pointer type for scalar parameters in BLAS APIs – Andrew Barker (20 minutes)
- Wrap-up and next steps – 2 minutes

Updates from last meeting

- Unified Acceleration Foundation was created!
  - Join the Math SIG Mailing List: https://lists.uxlfoundation.org/g/Math-SIG
- Open source oneMKL interfaces updates:
  - SYCL-BLAS renamed to portBLAS
  - Additional functionality supported and bugs fixed for various backends
  - In progress: sparse BLAS domain with MKLCPU backend

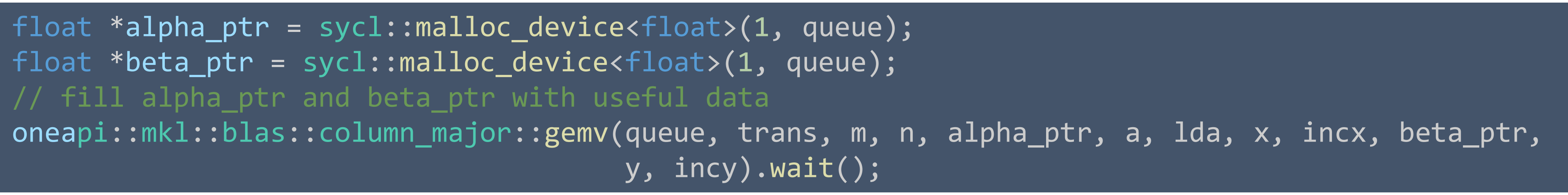
	x86 CPU	Intel GPU	NVIDIA GPU	AMD GPU
BLAS	Intel® oneMKL NETLIB LAPACK portBLAS	Intel® oneMKL portBLAS	NVIDIA cuBLAS portBLAS	AMD rocBLAS portBLAS
LAPACK	Intel® oneMKL	Intel® oneMKL	NVIDIA cuSOLVER	AMD rocSOLVER
RNG	Intel® oneMKL	Intel® oneMKL	NVIDIA cuRAND	AMD rocRAND
DFT	Intel® oneMKL	Intel® oneMKL	NVIDIA cuFFT	AMD rocFFT

Discussion on sparse BLAS APIs

value_or_pointer type for scalar parameters in BLAS APIs

Pointer/scalar flexibility in BLAS

- Allow users to pass either scalars (eg “float”) or pointers (eg “float*”) as scalar arguments (usually named “alpha” or “beta”) to BLAS USM functions in the oneMKL interfaces.



```
float *alpha_ptr = sycl::malloc_device<float>(1, queue);
float *beta_ptr = sycl::malloc_device<float>(1, queue);
// fill alpha_ptr and beta_ptr with useful data
oneapi::mkl::blas::column_major::gemv(queue, trans, m, n, alpha_ptr, a, lda, x, incx, beta_ptr,
                                      y, incy).wait();
```

Motivation

- Use data on the device as a scaling parameter without data transfers.
- Typical use case: use the result of a dot product to scale a gemv or an axpy in a conjugate gradient iteration.
- Ease transition from cuBLAS, which uses pointers for all its parameters.

Motivation

```
// using result of dot product on device
auto alpha = sycl::malloc_device<float>(1, queue);
auto dot_ev = oneapi::mkl::blas::dot(queue, n, x, 1, y, 1, alpha);
oneapi::mkl::blas::axpy(queue, n, alpha, x, 1, y, 1, {dot_ev}).wait();
sycl::free(alpha, queue);
```

```
// cublas prototype
cublasStatus_t cublasSgemv(cublasHandle_t handle, cublasOperation_t trans,
                           int m, int n,
                           const float    *alpha,
                           const float    *A, int lda,
                           const float    *x, int incx,
                           const float    *beta,
                           float          *y, int incy);
```

Minimize changes for existing users

- Existing code with alpha/beta scaling parameters passed by value will still be supported, and all existing calls to SYCL BLAS functions will work as expected.
- Users will be able to mix scalar and pointer parameters, for example a scalar for alpha and a pointer for beta.
- The current proposal is only for USM interfaces, not buffer interfaces.
- The current proposal is only for BLAS functions, but is easily extensible to other domains.

Proposed implementation

- The proposed implementation is a wrapper class called "value_or_pointer"
- This class can be (implicitly) constructed from either value types or pointer types.
- Users are not expected to explicitly use the class in any way.
- Casual users will not even need to know it exists.
- The new class will be visible in header files.
- Pointers used in the API may be raw pointers or USM-managed pointers, but raw pointers will not respect SYCL dependencies.

Example declaration

```
// proposed gemv declaration
sycl::event gemv(sycl::queue &queue, transpose trans, std::int64_t m, std::int64_t n,
                 value_or_pointer<float> alpha, const float *a, std::int64_t lda,
                 const float *x, std::int64_t incx, value_or_pointer<float> beta,
                 float *y, std::int64_t incy,
                 const std::vector<sycl::event> &dependencies = {});
```

Wrapper class

```cpp
// partial wrapper implementation
template <typename T>
class value_or_pointer {
    T value_;
    const T *ptr_;

public:
    // Constructor from value. Accepts not only type T but anything convertible to T.
    template <typename U, std::enable_if_t<std::is_convertible_v<U, T>, int> = 0>
    value_or_pointer(U value) : value_(value), ptr_(nullptr) {}

    // Constructor from pointer, assumed to be device-accessible.
    value_or_pointer(const T *ptr): value_(T(0)), ptr_(ptr) {}
};
```

Summary

- Proposal would allow users to pass either values or pointers as scalar parameters to BLAS functions in the USM API.
- Primary motivation is to allow device-side data for these parameters.
- Major changes to specification, touches almost every function.
- Relatively minor impact to users, existing code will continue to work.
- These kind of changes have been implemented for the Intel oneMKL 2024.0 release.
- PR into the spec: https://github.com/oneapi-src/oneAPI-spec/pull/503

Wrap-up

Next Steps

- Focuses for next meeting(s):
  - Discussion on discrete Fourier transform APIs
  - Any topics from Math SIG members?

- Please feel free to extend invitations to others to join Math SIG
- If anyone has content that they would like posted on oneAPI.io, please let us know

Resources

- Unified Acceleration Foundation: https://uxlfoundation.org/
- oneAPI Initiative: https://www.oneapi.io/
- Latest release of oneMKL Spec (currently v. 1.2; 1.3 provisional also available): 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 Community Forum: https://github.com/oneapi-src/oneAPI-tab
- GitHub for open source oneMKL interfaces (currently BLAS, RNG, LAPACK, and DFT domains): https://github.com/oneapi-src/oneMKL