OneMKL Sparse specification

Romain Biessy

20/09/2023

Plan

- Present changes to the oneMKL SPARSE that affect all backends
- Present differences between the oneMKL SPARSE API and cuSPARSE / rocSPARSE API
  - Make a decision on how close should the 2 APIs be

General specification changes

Missing sycl::queue in init_matrix_handle

- Function used to allocate an opaque type on the host
- Every function in oneMKL interface need a SYCL queue to perform runtime dispatching across backends.
- *init_matrix_handle* does not have a SYCL queue as a first argument
  - This prevents oneMKL interface from dispatching to the backend
  - This was done on purpose with the assumption that oneMKL interface would have its own matrix handle type and would not dispatch to the backend
  - Backend's *init_matrix_handle* would need to be called at a later stage, maybe *set_csr_data*?
- Our suggestion is to add the SYCL queue to be more consistent with other oneMKL function and allow dispatching to the backend.

Buffer API does not use sycl::event

- Buffer API does not match with the USM API
  - Buffer API does not return sycl::event which can be used for profiling or benchmarking
  - Buffer API does not accept event dependencies. It is not required with buffers but users may want this freedom still.
- It is easier to switch between buffers and USM if the API is the same
- Same issue in all other oneMKL domains
- May be good to consider this change if we change the sparse specification

cuSPARSE / rocSPARSE specification differences

cuSPARSE / rocSPARSE specification differences

- oneMKL interface is meant to allow users to transition from other vendor's libraries to oneMKL
  - oneMKL interface should be as close as possible to existing libraries to achieve this
- Rated the differences with a severity that represents the impact on a user that wants to transition from cuSPARSE / rocSPARSE to oneMKL SPARSE
- There are no consensus on sparse blas APIs like there is for blas

No equivalent to the global cuSPARSE handle

- A cuSPARSE handle is an opaque pointer that needs to be initialised and is used for every cuSPARSE functions
  - No equivalent in oneMKL, the closest would be the sycl::queue
- OneMKL blas has a similar issue with cuBLAS
  - OneMKL blas keeps a static map of the global handle for each pi_context and threads
  - **The user loses the ability to choose when to free these resources**

High severity

No equivalent to the global cuSPARSE handle – suggested solution

- Add equivalent for cusparseCreate and cusparseDestroy
- Structure would include the SYCL queue
- Structure would replace the SYCL queue in other SPARSE functions
- No impact on the oneMKL product

```
namespace oneapi::mkl::sparse {
  struct handle;
  using handle_t = *handle;

  void init_handle(handle_t *p_handle,
sycl::queue &queue);

  void release_handle(handle_t handle);
}
```

High severity

Optimize_* mismatch

- cuSPARSE gives more freedom to optimize sparse operations
  - cuSPARSE lets the user query the scratch buffer size
  - **It is the user's responsibility to allocate and free the scratch buffer**
- Will be difficult for users transitioning from cuSPARSE to oneMKL
- Workaround requires us to store the scratch buffer size and scratch pointer inside the matrix handle
  - The matrix handle can be re-used in multiple operations so we need to store this information for every operation and every operation's configuration possible
  - **The user loses the ability to choose when to free these resources**
- Calling the optimization functions is mandatory in cuSPARSE
  - It should also be mandatory in oneMKL SPARSE to guarantee an application works with all backends
- cuSPARSE guarantees no internal allocations
  - oneMKL interface could allow internal allocations and require the user to allocate external memory of the size requested by the backend

High severity

Optimize_* mismatch – suggested solution

- Modify optimize_* to output the buffer_size
- Also impact the operation itself to accept a pointer to the user-allocated scratch buffer
- Add all of the operation's parameters to the optimize_* functions to match cuSPARSE
- **Need to measure the impact on oneMKL product**

```
namespace oneapi::mkl::sparse {
  sycl::event optimize_gemv(
    sycl::queue &queue,
    transpose transpose_val,
    const fp alpha,
    matrix_handle_t A_handle,
    const *fp x,
    const fp beta,
    *fp y,
    intType &buffer_size,
    const
std::vector<sycl::event>                    &de
pendencies = {});
}
```

High severity

optimize_trsv does not match cuSPARSE

- cuSPARSE uses a separate descriptor to store information across *cusparseSpSV* functions: *cusparseSpSVDescr_t*
  - This separate descriptors needs to be created and destroyed by the user
- cuSPARSE splits the pre-processing functions in 2: *cusparseSpSV_bufferSize* and *cusparseSpSV_analysis*
- Equivalent of uplo_val and diag_val are stored in the matrix handle
- Transition may be difficult for some user
- oneMKL interface could match cuSPARSE API without affecting oneMKL product

High severity

cuSPARSE matrix handle getters and setters

- cuSPARSE matrix handle have getter and setter functions:
  - *cusparseSpMatGetSize*, *cusparseSpMatGetIndexBase*, *cusparseSpMat(Get|Set)Values*,
  - *cusparseSpMatGetFormat* may not be needed while oneMKL only supports CSR
  - *cusparseSpMatGetStridedBatch* may not be needed while oneMKL does not support batched CSR
  - *cusparseSpMat(Get|Set)Attribute*, used for trsv (i.e. *cusparseSpSV*)
- Users may rely on these functions in their application
- Suggested solution is to match cuSPARSE
  - oneMKL interface could support this for MKLCPU and MKLGPU backends without changing oneMKL product

High severity

cuSPARSE matrix handle can be const

- cuSPARSE can create const and non-const matrix handles
- Non-const handles can be implicitly casted to const ones
- Most likely only useful for user readibility
- Suggested solution is to match cuSPARSE by adding a *const_matrix_handle_t* type
  - Ideally the same should be applied to oneMKL product
  - Most functions should use the const handle when possible

Medium severity

Operations mismatch - naming

- oneMKL and cuSPARSE use different names for the operations
  - **gemv maps to cusparseSpMV**
    - cuSPARSE has a gemvi operation but this is used for dense matrix A and sparse vector x
  - **gemm maps to cusparseSpMM**
    - cuSPARSE has a cusparseSpGEMM operation but this is used for 2 sparse matrices A and B
      - We will add matmat to map with cusparseSpGEMM
  - **trsv maps to cusparseSpSV**
- Consider renaming the operations to match cuSPARSE
- Need to document how each operation map for each backend
- No impact on oneMKL product

Low severity

Operations mismatch – different algorithms

- cuSPARSE gives the ability to choose different algorithms for some operations
  - Algorithm can impact performance, memory usage, whether the operation is deterministic
  - Users would lose this ability with the current oneMKL API
- Suggested solution is to add a parameter to the operations and their optimize function to support this
  - oneMKL should have its own enum for the algorithms and document how they map to the backends' algorithms
  - This is only a hint that oneMKL could ignore
  - No impact on oneMKL product

Medium severity

Operations mismatch – alpha/beta can be on host or device

- cuSPARSE supports alpha and beta parameters as values on the host or on the device
  - oneMKL requires them to be on the device
- May be a large impact if the user needs to add a synchronization point to copy the values to the host and launch the following kernels
- No proper solution is possible in oneMKL interface if the backends do not support this
- Unclear if this is useful in real world applications. May be revisited later.

Medium severity

Operations mismatch – not supported by cuSPARSE

- cuSPARSE does not support symv, trmv and gemvdot
  - symv and trmv used to be supported with their legacy API but were removed with their new generic API
- Not clear if there is value in supporting them in oneMKL interface
- We suggest not supporting these operations for now
  - Can be revisited if users request them

Low severity

cuSPARSE remaining differences

- cuSPARSE is more like a C API
  - cuSPARSE returns error code as an enum, oneMKL relies on exceptions
  - cuSPARSE uses void* for data and enums for the underlying types, oneMKL uses typed pointers
- These are acceptable if we decide that oneMKL should be a C++ API
  - Can impact the cuSPARSE users
- oneMKL has 2 functions to initialize the matrix handle and set the data (*init_matrix_handle* and *set_csr_data*) while cuSPARSE merges the 2 (*cusparseCreateCsr*)
  - Probably no impact for the user

Conclusion

- oneMKL SPARSE API should move closer to cuSPARSE API if we want to allow more users to transition
- Overall cuSPARSE is a slightly lower level API than oneMKL SPARSE
  - Easier to support MKLCPU and MKLGPU backends with a cuSPARSE like API than the other way around
- We are already working on supporting the MKLCPU and MKLGPU backends. These changes will impact this work.