codeplay®

New oneMKL sparse specification

Romain BIESSY

24/04/2024

New oneMKL sparse specification

- Follow up from the Math SIG on 20th Sep
- Specification PR #522 is in review
- The main motivation for updating the sparse specification was to better align with cuSPARSE and rocSPARSE APIs
    - Allows to support these backends with less overhead
    - Makes it easier to transition existing applications to oneMKL

Operations renaming

Previous API	New API	Description	Comment
gemv symv trmv	spmv	Sparse matrix by dense vector multiplication	Matrix views are used to differentiate between matrices that are general, symmetric, triangular, etc.
gemm	spmm	Sparse matrix by dense matrix multiplication	
trsv	spsv	Solves a system of linear equations where the coefficients are described by a triangular sparse matrix	
gemvdot	N/A	Fused sparse matrix by dense vector multiplication followed by dot product of resulting dense vector against input dense vector	Dropped support for gemvdot as there didn't seem to be enough use-cases nor support in other backends

SPMV example – changes to handle types

Previous API

```
matrix_handle_t A = nullptr;
init_matrix_handle(q, &A);
auto ev_set = set_csr_data(q, A, nrows, ncols, nnz, index_base::zero, ia_ptr, ja_ptr, a_ptr);




[…]


auto ev_release_mat = release_matrix_handle(q, &A, { ev_gemv });
```

New API

```
matrix_handle_t A = nullptr;
init_csr_matrix(q, A, nrows, ncols, nnz, index_base::zero, ia_ptr, ja_ptr, a_ptr);

dense_vector_handle_t x = nullptr, y = nullptr;
init_dense_vector(q, &x, ncols, x_ptr);
init_dense_vector(q, &y, nrows, y_ptr);
[…]
auto ev_release_x = release_dense_vector(q, x, {ev_spmv});
auto ev_release_y = release_dense_vector(q, y, {ev_spmv});
auto ev_release_mat = release_matrix_handle(q, &A, {ev_spmv});
```

Introducing *init_<sparse_format>_matrix* and *set_<sparse_format>_matrix*

Introducing *dense_vector_handle* and *dense_matrix_handle*

SPMV example – changes to handle types

Motivation

- The dense handle types are needed:
  - to avoid overheads for the cuSPARSE and rocSPARSE backends.
  - to make it easier to transition existing applications to oneMKL.
- The sparse matrices can now be initialized in a single function for easier use.

New API

```
matrix_handle_t A = nullptr;
init_csr_matrix(q, A, nrows, ncols, nnz, index_base::zero, ia_pt
r, ja_ptr, a_ptr);



dense_vector_handle_t x = nullptr, y = nullptr;
init_dense_vector(q, &x, ncols, x_ptr);
init_dense_vector(q, &y, nrows, y_ptr);
[…]
auto ev_release_x = release_dense_vector(q, x, {ev_spmv});
auto ev_release_y = release_dense_vector(q, y, {ev_spmv});
auto ev_release_mat = release_matrix_handle(q, &A, {ev_spmv});
```

SPMV example – operation descriptor

Previous API

```
N/A
```

New API

```
spmv_descr_t spmv_descr = nullptr;
init_spmv_descr(q, &spmv_descr);
[…]
auto ev_opt = spmv_optimize(q, transA, alpha, A_view, A, x,
beta, y, alg, spmv_descr, workspace_ptr);
auto ev_spmv = spmv(q, transA, alpha, A_view, A, x, beta, y,
alg, spmv_descr, {ev_opt});
auto ev_release_descr = release_spmv_descr(q, spmv_descr,
{ev_spmv});
```

- Each operation has its own descriptor type that needs to be initialized and released
- It is used to store data across the different functions of the operation
- It does not use a C++ object constructor and destructor as we need to have an asynchronous destruction that waits on input events and return an event.

SPMV example – algorithm enum

Previous API

```
N/A
```

New API

```
spmv_alg alg = spmv_alg::default_alg;
auto ev_opt = spmv_optimize(q, transA, alpha, A_view, A, x,
beta, y, alg, spmv_descr, workspace_ptr);
auto ev_spmv = spmv(q, transA, alpha, A_view, A, x, beta, y,
alg, spmv_descr, {ev_opt});
```

- Each operation has its own algorithm enum type
- Lets the user tune operations to have different properties (i.e. determinism), better performance for some layouts or to disable the optimization step depending on what the backends support.

Example spmv_alg

```
enum class spmv_alg {
    default_alg, no_optimize_alg,
    coo_alg1, coo_alg2,
    csr_alg1, csr_alg2, csr_alg3
};
```

SPMV example – Matrix views

Previous API

```
N/A
```

New API

```
matrix_view A_view;
auto ev_opt = spmv_optimize(q, transA, alpha, A_view, A, x, beta, y, alg, spmv_descr, workspace_ptr);
auto ev_spmv = spmv(q, transA, alpha, A_view, A, x, beta, y, alg, spmv_descr, {ev_opt});
```

- The matrix view is used to specify which part of the matrix should be read.
- By default, the matrix is assumed to be "general".
- 2 use cases:
  - Change the definition of the operation if the matrix handle is a full matrix but is viewed as triangular.
  - Optimization hint if the matrix handle is already triangular.

Matrix view type

```
enum class matrix_descr {
    general,
    symmetric,
    hermitian,
    triangular,
    diagonal,
};
```

```
struct matrix_view {
    matrix_descr type_view;
    uplo uplo_view;
    diag diag_view;
    […]
};
```

SPMV example – Matrix properties

Previous API

```
N/A
```

New API

```
bool is_supported = set_matrix_properties(q, A,
matrix_property::symmetric);
```

- Matrix properties are strong guarantees on the matrix's data.
- The user must ensure that the underlying data follow the properties that are set.
- Used as an optimization hint.

Matrix properties

```
enum matrix_property : std::int32_t {
   symmetric = 1 << 0,
   sorted    = 1 << 1,
};
```

SPMV example – External workspace

Previous API

```
N/A
```

New API

```
std::size_t workspace_size;

spmv_buffer_size(q, transA, alpha, A_view, A, x, beta, y, alg,
spmv_descr, workspace_size);

auto workspace_ptr = sycl::malloc_device(workspace_size, q);

auto ev_opt = spmv_optimize(q, transA, alpha, A_view, A, x,
beta, y, alg, spmv_descr, workspace_ptr);

auto ev_spmv = spmv(q, transA, alpha, A_view, A, x, beta, y,
alg, spmv_descr, {ev_opt});
```

- The user must query the workspace size and allocate a buffer or USM pointer of at least that size for every operation.
- This gives better control of the memory to the user.
- The functions *<operation>_buffer_size* are synchronous to allow the host to use the *workspace_size.*

SPMV example – putting it all together

Previous API

```
matrix_handle_t A = nullptr;
init_matrix_handle(q, &A);
auto ev_set = set_csr_data(q, A, nrows, ncols, nnz, index_base::zero,
ia_ptr, ja_ptr, a_ptr);









auto ev_opt = optimize_gemv(q, transA, A, { ev_set });

auto ev_gemv = gemv(q, transA, alpha, A, x_ptr, beta, y_ptr, { ev_opt });



auto ev_release = release_matrix_handle(q, &A, { ev_gemv });
```

New API

```
matrix_handle_t A = nullptr;
init_csr_matrix(q, A, nrows, ncols, nnz, index_base::zero, ia_ptr, ja_ptr, a_ptr);
dense_vector_handle_t x = nullptr, y = nullptr;
init_dense_vector(q, &x, ncols, x_ptr);
init_dense_vector(q, &y, nrows, y_ptr);
spmv_descr_t spmv_descr = nullptr;
init_spmv_descr(q, &spmv_descr);
matrix_view A_view;
spmv_alg alg = spmv_alg::default_alg;
std::size_t workspace_size;
spmv_buffer_size(q, transA, alpha, A_view, A, x, beta, y, alg, spmv_descr, workspace_size);
auto workspace_ptr = sycl::malloc_device(workspace_size, q);
auto ev_opt = spmv_optimize(q, transA, alpha, A_view, A, x, beta, y, alg, spmv_descr,
workspace_ptr);
auto ev_spmv = spmv(q, transA, alpha, A_view, A, x, beta, y, alg, spmv_descr, {ev_opt});
auto ev_release_descr = release_spmv_descr(q, spmv_descr, {ev_spmv});
auto ev_release_x = release_dense_vector(q, x, {ev_spmv});
auto ev_release_y = release_dense_vector(q, y, {ev_spmv});
auto ev_release_mat = release_matrix_handle(q, &A, {ev_spmv});
```

Summary of the changes

- Renamed the operations and dropped support for *gemvdot*.
- Small changes to handle type and new dense handle types.
- Introduce operation descriptor.
- Introduce algorithm enums.
- Introduce matrix properties and views.
- Add support for external workspace.

codeplay®

Disclaimers

A wee bit of legal

No product or component can be absolutely secure.

Your costs and results may vary.

Intel technologies may require enabled hardware, software or service activation.