oneMKL Technical Advisory Board

Session 11

February 24, 2021

Agenda

- Welcoming remarks – 5 minutes
- Updates from last meeting – 5 minutes
- Overview of oneMKL Sparse BLAS domain – Spencer Patty (30 minutes)
- Wrap-up and next steps – 5 minutes

Updates from last meeting

- oneAPI Math Kernel Library (oneMKL) Interfaces Project
  - LBNL has a pull request for supporting the NVIDIA cuRAND backend for random number generators domain on NVIDIA GPUs
- Will transition from Webex to Microsoft Teams after this meeting
  - Will cancel current meeting series and set up a new series

Overview of oneMKL Sparse BLAS domain

Sparse Matrix Formats in Sparse BLAS Libraries

- Compressed Sparse Row (CSR) format
  - 3-array variant (oneMKL IE SpBLAS C/Fortran)(oneMKL DPC++ SpBLAS*)(cuSPARSE)(MAGMA)(rocSPARSE)(Ginkgo)(ViennaCL)
  - 4-array variant (oneMKL IE SpBLAS C/Fortran)(plan for oneMKL DPC++ SpBLAS*)
- Compressed Sparse Column (CSC) format
  - 3-array variant (oneMKL IE SpBLAS C/Fortran)(plan for oneMKL DPC++ SpBLAS*)(cuSPARSE)
  - 4-array variant (oneMKL IE SpBLAS C/Fortran)(plan for oneMKL DPC++ SpBLAS*)
- Coordinate (COO) Format
  - (COO) Structure of arrays (SoA) format (oneMKL IE SpBLAS C/Fortran)(cuSPARSE)(rocSPARSE)(Ginkgo)(ViennaCL)
  - (COO_AOS) Array of structures (AoS) format (cuSPARSE)(rocSPARSE)
- Block Compressed Sparse Row (BSR) format
  - 3-array variant (oneMKL IE SpBLAS C/Fortran)(plan for oneMKL DPC++ SpBLAS*)(cuSPARSE)(rocSPARSE)
  - 4-array variant (oneMKL IE SpBLAS C/Fortran)(plan for oneMKL DPC++ SpBLAS*)(cuSPARSE)
- ELLPACK format
  - ELL (MAGMA)(rocSPARSE)(Ginkgo)(ViennaCL)
  - Sliced-ELL or SELL-P (MAGMA)(Ginkgo)(ViennaCL)
  - Blocked ELL (cuSPARSE)
- Hybrid (ELL + COO) (rocSPARSE)(Ginkgo)(ViennaCL)
- Dense format (plan for oneMKL DPC++ SpBLAS*)(cuSPARSE)(MAGMA)(Ginkgo)(ViennaCL)
- Diagonal format

*oneMKL IE SpBLAS C/Fortran Openmp Offload matrix formats will align with oneMKL DPC++ SpBLAS

intType : {std::int32_t, std::int64_t}
fpType : {float, double, std::complex<float>, std::complex<double> }

Sparse CSR 3-array matrix format

- A in compressed sparse row matrix format:
  - **num_rows** – (intType) number of rows in A
  - **num_cols** – (intType) number of columns in A
  - **nnz** – (intType) number of non-zeros in A (= **row_ptr**[num_rows])
  - **index** – (intType) 0 (C/C++ style) or 1 (Fortran style) based indices in **row_ptr**, **col_ind** arrays
  - **row_ptr** – (intType[num_rows+1]) array of offsets for each row k in **col_ind** /**val** arrays.
    I.e. **row_ptr**[k] is the start of row k in **col_ind** and **val** arrays.
  - **col_ind** – (intType[nnz]) array of column indices
  - **val** – (fpType[nnz]) array of values

A:

a_0			a_1	a_2	
	a_3	a_4			
		a_5		a_6	a_7
				a_8	
				a_9	a_10

A in CSR format

num_rows=5, num_cols=6, nnz=11, index=0

row_ptr:	0	3	5	8	9	11					
val:	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_10
col_ind:	0	3	4	1	2	2	4	5	4	4	5

- Memory footprint of A is
  - sizeof(A) = [sizeof(intType) + sizeof(fpType)]*nnz + sizeof(intType)*(num_rows+1)

Sparse BLAS Analysis-Execution Strategy

Motivation:

- Sparse BLAS operations are often used in algorithms where the same operation is applied multiple times until some stopping criteria. Example: iterative solvers like conjugate gradient, preconditioners, power method for eigen-solvers, etc..
- Changing the internal format of data and/or analyzing the sparsity structure helps to exploit better optimized kernels and, in some cases like TRSV, extract more parallelism/enable a greater level of parallelism.

Analysis-Execution Strategy:

- Initialization Stage – create the opaque matrix handle and provide it user data
- Analysis Stage (Preprocessing step) – prepare internal structures, data for given matrices and operations. Only called once per operation to be optimized. Can be skipped at cost of possibly worse performance in execution stage.
- Execution Stage – performs the operation, can be called once or many times
- Release Stage – clean up matrix handle and release internally allocated data

Sparse BLAS Naming Strategy

- Namespace:

  oneapi::mkl::sparse

- API naming strategy:
  - Mirror BLAS names where it makes sense (GEMV, SYMV, GEMM, TRSV, etc.)
  - Prefer overloading APIs to templating for intType, fpType, and USM/buffer
    - intType: std::int32_t, std::int64_t
    - fpType: float, double, std::complex<float>, std::complex<double>
    - Return `void` type for sycl::buffer APIs
    - Return `cl::sycl::event` and last argument is a `sycl::vector_class<cl::sycl::event> &dependencies` for USM APIs.
  - Use lower camel case with action verbs when appropriate
    - make_transpose
    - set_csr_data
    - release_matrix_handle, etc.

Sparse BLAS Auxiliary (Init/Release) DPC++ APIs

```
enum class property : char {
    symmetric = 0,
    sorted    = 1,
};

struct matrix_handle;
typedef struct matrix_handle *matrix_handle_t;

void init_matrix_handle(matrix_handle_t *handle);

void release_matrix_handle(matrix_handle_t *handle,
                           const cl::sycl::vector_class<cl::sycl::event> &dependencies = {});

void set_matrix_property(matrix_handle_t handle, property property_value);

void set_csr_data(matrix_handle_t handle,
                  const std::int32_t num_rows,
                  const std::int32_t num_cols,
                  index_base index,
                  cl::sycl::buffer<std::int32_t, 1> &row_ptr,
                  cl::sycl::buffer<std::int32_t, 1> &col_ind,
                  cl::sycl::buffer<float, 1> &val);

void set_csr_data(matrix_handle_t handle,
                  const std::int32_t num_rows,
                  const std::int32_t num_cols,
                  index_base index,
                  std::int32_t *row_ptr,
                  std::int32_t *col_ind,
                  float *val);
```

Use opaque handle to store matrix data and any internal optimized data

Lightweight initialization of handle structure and pointers

Not yet defined in oneMKL Spec, but will be added, useful for communicating properties of matrix like, sorted, symmetric, etc.

Use overloading for all variants of intType:
`{std::int32_t, std::int64_t}`
and fpType:
`{float, double, std::complex<float>, std::complex<double>}`

$$\mathrm{op}(A) = \begin{cases} A \\ A^T \\ A^H \end{cases}$$

Sparse BLAS Level 2 and Level 3 DPC++ APIs

Execution Stage

- sparse::gemv
- sparse::trmv
- sparse::symv
- sparse::gemvdot
- sparse::trsv
- sparse::gemm

Analysis stage

- sparse::optimize_gemv
- sparse::optimize_trmv
- sparse::optimize_trsv

MV: A is a sparse matrix, x, y are dense vectors, α, β, d are scalars:

$$y = \alpha \cdot \mathrm{op}(A) \cdot x + \beta \cdot y$$
$$d = \mathrm{dot}(y, x)$$

TRSV: A is a sparse triangular matrix, x, y are dense vectors:

Solve for y:

$$\mathrm{op}(A) \cdot y = x$$

GEMM: A is a sparse matrix, X, Y are dense matrices, α, β are scalars:

$$Y = \alpha \cdot \mathrm{op}(A) \cdot X + \beta \cdot Y$$

Sparse BLAS GEMV DPC++ API

GEMV: A a general sparse matrix, x, y dense vectors, α, β scalars:

$$y = \alpha \cdot \mathrm{op}(A) \cdot x + \beta \cdot y$$

sycl::buffer APIs

```
void optimize_gemv(cl::sycl::queue &queue, transpose transpose_flag, matrix_handle_t handle);

void gemv(cl::sycl::queue &queue,
          transpose transpose_flag,
          const float alpha,
          matrix_handle_t handle,
          cl::sycl::buffer<float, 1> &x,
          const float beta,
          cl::sycl::buffer<float, 1> &y);
```

USM APIs

```
cl::sycl::event optimize_gemv(cl::sycl::queue &queue,
                              transpose transpose_flag,
                              matrix_handle_t handle,
                              const cl::sycl::vector_class<cl::sycl::event> &dependencies);

cl::sycl::event gemv(cl::sycl::queue &queue,
                     transpose transpose_flag,
                     const float alpha,
                     matrix_handle_t handle,
                     float *x,
                     const float beta,
                     float *y,
                     const cl::sycl::vector_class<cl::sycl::event> &dependencies = {});
```

Possible Sparse * Sparse Operation

$$\mathrm{op}(A) = \begin{cases} A \\ A^T \\ A^H \end{cases}$$

- Operations current supported in oneMKL IE SpBLAS C/Fortran APIs

Sparse Matrix Output

- SpGEMM: (A, B, C sparse)
  - $C = \mathrm{opA}(A) \cdot \mathrm{opB}(B)$
  - $C = \mathrm{opC}(\, \mathrm{opA}(A) \cdot \mathrm{opB}(B) \,)$
  - $C = \alpha \cdot \mathrm{opC}(\, \mathrm{opA}(A) \cdot \mathrm{opB}(B) \,) + \beta \cdot C$

Dense Matrix Output

- SpGEMM (A, B sparse, C dense)
  - $C = \mathrm{opA}(A) * \mathrm{opB}(B)$
  - $C = \mathrm{opC}(\, \mathrm{opA}(A) \cdot \mathrm{opB}(B) \,)$

Open Question: Do you see value in supporting extra opC() in SpGEMM operations?

- Ex: computing B^T * A^T requires transposing two sparse matrices vs computing (A*B)^T requires transposing only one.
- Can adds a lot more complexity to implementations, would people use it?

Sparse BLAS Other Possible DPC++ APIs

$$\mathrm{op}(A) = \begin{cases} A \\ A^T \\ A^H \end{cases}$$

- Operations current supported in oneMKL IE SpBLAS C/Fortran APIs

APIs with Sparse Output

- SYRK (A,C sparse)
  - $C = \mathrm{op}(A) \cdot \mathrm{op}(A)^H$
- SYPR (A,B,C sparse, B Hermitian) (symmetric triple product)
  - $C = \mathrm{op}(A) \cdot B \cdot \mathrm{op}(A)^H$
- Matrix_Add (A, B, C sparse)
  - $C = \alpha \cdot op(A) + B$
  - $C = \alpha \cdot \mathrm{opA}(A) + \beta \cdot \mathrm{opB}(B)$
- Matrix_Copy (A, C sparse)
  - $C = A$
  - $C = \alpha \cdot \mathrm{op}(A)$
- Sampled dense-dense matrix product (SDDMM) (A,B sparse, C,D dense)
  - $A = B .* (CD)$ or using index notation ($A_{ij} = B_{ij} \cdot C_{ik} \cdot D_{kj}$)

APIs with Dense Matrix or Vector Output

- TRSM (A sparse triangular, X, Y dense)
  - $Y = \alpha \cdot \mathrm{op}(A)^{-1} \cdot X$
- SYRK (A sparse, Y dense)
  - $Y = \mathrm{op}(A) \cdot \mathrm{op}(A)^H$
- SYPR (A,B sparse, B Hermitian, Y dense)
  - $Y = \mathrm{op}(A) \cdot B \cdot op(A)^H$
- GEMVShift (A sparse, x, y dense vectors, α, β, λ scalars)
  - $\mathrm{y} = \alpha \cdot \mathrm{op}(A - \lambda I) \cdot x + \beta \cdot y$
- ILU0 preconditioner step
- SYMGS preconditioner step

Do we need support for Sparse Vectors ?

Next Steps

- Focuses for next meeting(s):
  - Discrete Fourier transforms
  - Any topics from oneMKL TAB members?
- If anyone has content that they would like posted on oneAPI.com, please let us know

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 open source oneMKL interfaces (currently BLAS and RNG domains): https://github.com/oneapi-src/oneMKL