oneMKL Technical Advisory Board

Session 16

October 6, 2021

Agenda

- Welcoming remarks – 5 minutes
- Updates from last meeting – 5 minutes
- DPC++ APIs for oneMKL Data Fitting - Nikita Semin and Andrey Fedorov (30 minutes)
- Wrap-up and next steps – 5 minutes

oneMKL TAB Members

- Ye Luo, Argonne National Laboratory (ANL)
- *Nichols Romero, ANL* - stepped down

- Hartwig Anzt, Karlsruhe Institute of Technology (KIT)
- Romain Dolbeau, SiPearl
- Mehdi Goli, Codeplay
- Mark Hoemmen, Stellar Science
- Nevin Liber, Argonne National Laboratory (ANL)
- Piotr Luszczek, Innovative Computing Laboratory (ICL) at University of Tennessee, Knoxville (UTK)
- Vincent Pascuzzi, Brookhaven National Laboratory
- Pat Quillen, MathWorks
- Edward Smyth, Numerical Algorithms Group (NAG)

- Brief intro: your job; how you use math libraries

Updates from last meeting

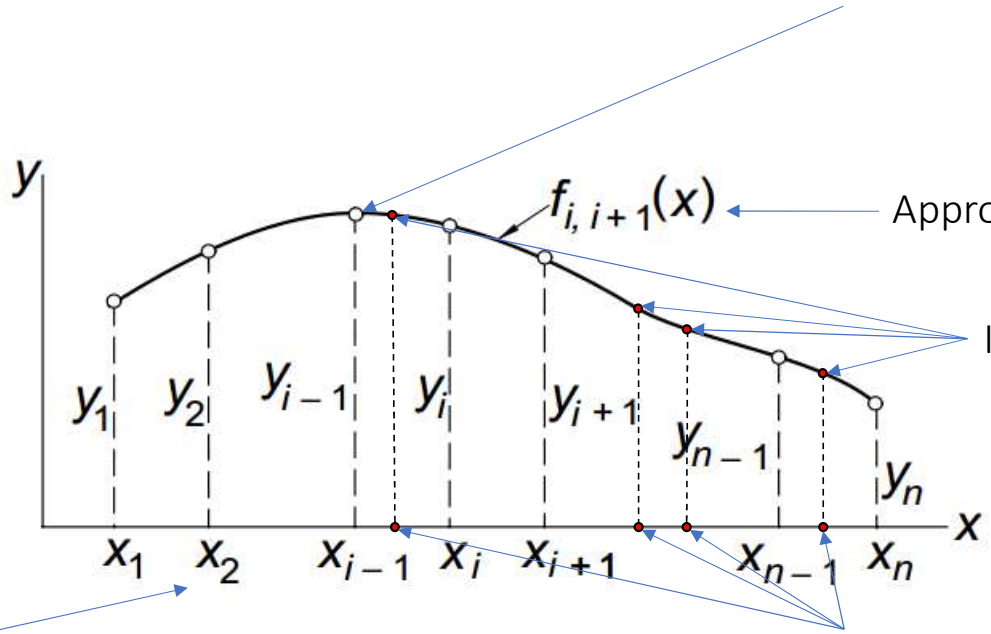
- oneMKL Spec: Issue/Request for Comments #364 for extending sparse * dense matrix product to support column major storage; Pull Request #363 for addressing this
- cl::sycl::vector_class is deprecated
  - Used in oneMKL USM APIs to pass list of input event dependencies
  - These oneMKL APIs will be deprecated and ones using std::vector added for v. 1.1
- oneAPI Math Kernel Library (oneMKL) Interfaces Project
  - oneMKL builds successfully with CUDA 11.4 (CUDA 10.2 is default/supported version)
- Intel oneMKL 2021.4 was released

DPC++ APIs for oneMKL Data Fitting

What Data Fitting is

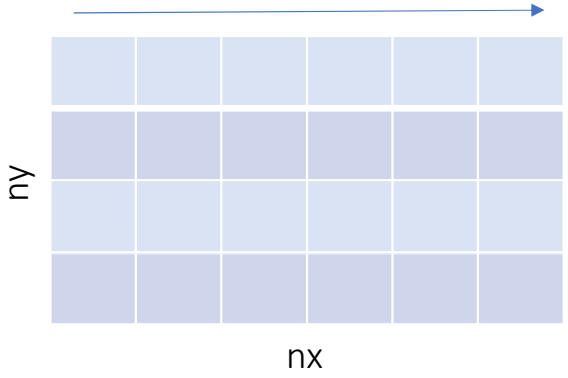
- Set of functions to perform
  - interpolation (function values and derivatives)
  - integration

N function values for each function (multiple functions may be used for one partition)

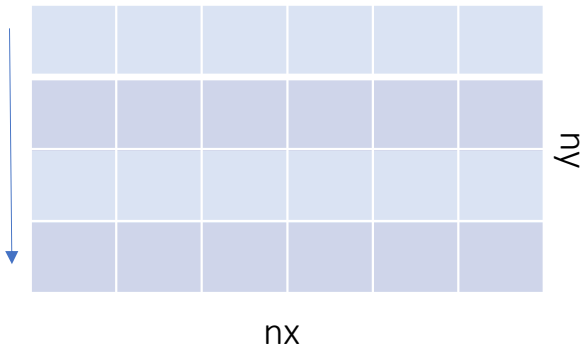


Data Layouts for coefficients and functions

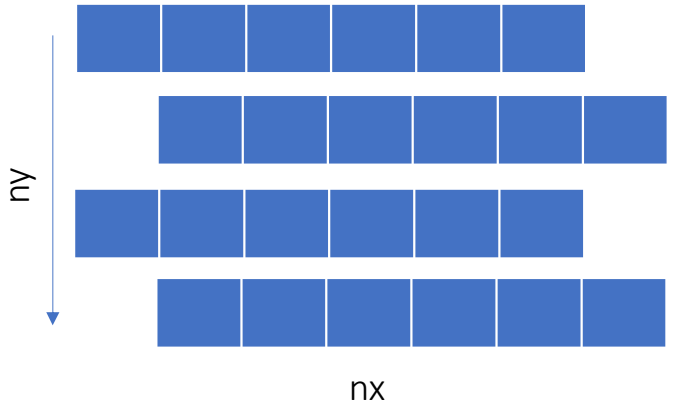
Row major



Column major



"First coordinate"



nx - quantity of partitions
ny - quantity of functions

Independent memory for each set of coefficients/functions

Usage of Data Fitting

- Libraries that contain interpolation functions:
  - Intel® oneAPI Math Kernel Library (oneMKL) (C)
  - IMSL (C)
  - GNU Scientific Library (C)
  - ArrayFire (C++)
  - Eigen (supported by community) (C++)
  - SciPy (Python)
  - NumPy (Python, only linear 1d)
  - cuPy (Python, only linear 1d)
- Applications that use interpolation:
  - GROMACS
  - NAMD

Proposal

Introduce Data Fitting spline-based functionality in oneMKL Spec

oneMKL Spec changes

It's better to have Data Fitting DPC++ APIs as an experimental feature for Intel® oneAPI Math Kernel Library to have a chance to change interfaces later based on feedback from the community.

We can open a PR for the oneMKL Specification repo to get feedbacks.

Since our interfaces are not clearly defined yet, let's use https://github.com/oneapi-src/oneAPI-spec/issues/379 for discussions.

What do you think about adding Data Fitting APIs to oneMKL Spec?

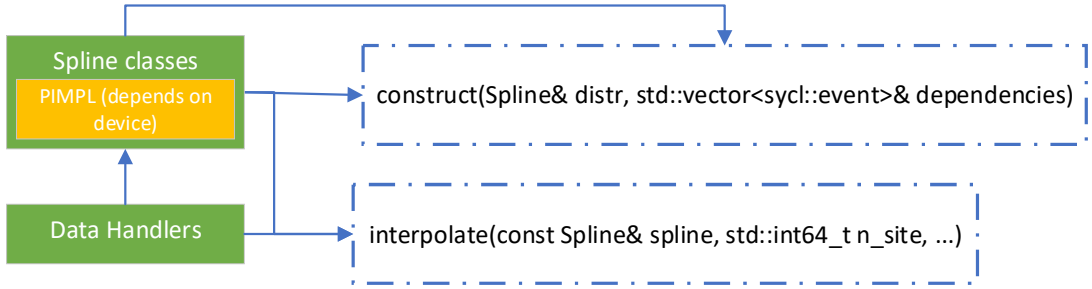
Is such process to add APIs good enough?

Proposed APIs

Proposed APIs* for spline-based Data Fitting

Support both Buffer and USM APIs

- Spline classes:
  - linear_spline
  - quadratic_spline
  - cubic_spline
- Free functions:
  - construct – computes coefficients of spline
  - interpolate – performs interpolation (including function and derivatives computation)
- Handlers to store data

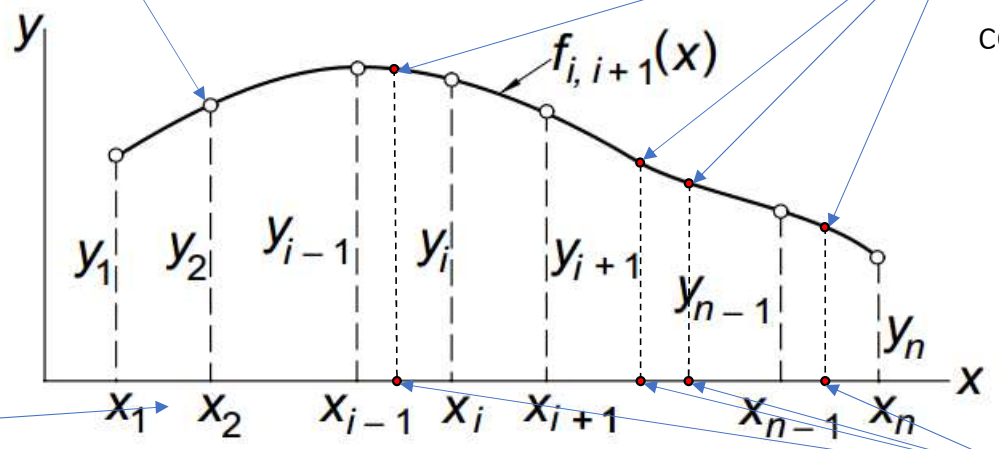


* APIs are experimental and may be modified in future

Glossary

N function values for each function (multiple functions may be used for one partition)

storage_hint: row_major, col_major, first_coordinate



Interpolation results

storage_hint: row_major, col_major, first_coordinate

N partitions of X

partition_hint: non_uniform, quasi_uniform, uniform

Interpolation sites

site_hint: non_uniform, uniform, sorted

Basic example

```
#include “oneapi/mkl/experimental/df.hpp”

namespace mkl_df = oneapi::mkl::experimental::df;

int main() {
      std::int64_t nx = 10000; // quantity of x points
      std::int64_t ny = 100; // quantity of functions
      float* coeffs = sycl::malloc_shared<float>(ny * (nx – 1) * 4, q);
      float* functions = sycl::malloc_shared<float>(ny * nx, q);
      // memory allocation for sites, results, partitions and initialization of partitions, functions
….
      // Data Fitting usage
      mkl_df::coeffs_funcs_handler cfh(coeffs, functions); // create handler
      mkl_df::partitions_handler part(partitions); // create handler → partitions_handler<partition_hint::non_uniform>
      mkl_df::cubic_spline cubic_spline(q, cfh, part, nx, ny); // create spline object to accumulate all data
      mkl_df::construct(cubic_spline); // compute spline coefficients
      // can create handlers on go
      mkl_df::interpolate(cubic_spline, {sites}, {results}, nsites); // interpolate data basing on computed spline coefficients
….
}
```

Advanced example

```
…
// Data Fitting usage
// using first_coordinate as layout for coefficients
std::vector<float*> coeffs;
coeffs.push_back(some_valid_coeff_ptr);
coeffs.push_back(another_valid_coeff_ptr);
…
// c++17 afford us to use deduction guides
mkl_df::coeffs_funcs_handler cfh(coeffs, functions);
mkl_df::partitions_handler<mkl_df::quasi_uniform> part(partitions);
mkl_df::cubic_spline cubic_spline(q, cfh, part, nx, ny);
mkl_df::construct(cubic_spline);
mkl_df::interpolate(cubic_spline, {sites}, {results}, nsites);
…
```

Spline object has the same type as in previous example. Multiple splines can be stored in a container (e.g., std::vector)

Spline Interface

```
template<typename FpStorage = float*,typename SplineType = cubic_spline_type::default_spline>
class cubic_spline {
public:
    using fp_storage = FpStorage;

    template <coeffs_computed_hint CoeffsComputedHint,
        storage_hint CoeffsHint,
        storage_hint FunctionsHint,
        partition_hint PartitionsHint,
        bc_type BCType = bc_type::no_bc,
        ic_type ICType = ic_type::no_ic>
    cubic_spline(const sycl::queue& q,
        coeff_func_handler< CoeffsComputedHint, CoeffsHint, FunctionsHint, FpStorage> cfh,
        partitions_handler<PartitionsHint, FpStorage> partitions,
        std::int64_t nx,
        std::int64_t ny = 1,
        bc_handler<BCType, FpStorage> bc = {0},
        ic_handler<ICType, FpStorage> ic = {0});

protected:
    std::unique_ptr<detail::spline_base<SplineType, FpStorage>> impl_;
}
```

Construct & Interpolate Interfaces

```
template<typename Spline>
sycl::event construct(Spline& spline, const std::vector<sycl::event>& dependencies = {});


template<typename Spline,
    site_hint SiteHint = site_hint::non_uniform,
    storage_hint ResHint = storage_hint::row_major>
sycl::event interpolate(const Spline& spline,
    sites_handler<SiteHint, typename Spline::fp_storage> sites,
    storage_hanlder<ResHint, typename Spline::fp_storage> results,
    std::int64_t n_sites,
    derivative_handler derivative = {},
    const std::vector<sycl::event>& dependencies = {});
```

Summary

We propose to add DPC++ Data Fitting interfaces to oneMKL Spec.

Waiting for feedback!

https://github.com/oneapi-src/oneAPI-spec/issues/379

Wrap-up

Next Steps

- Focuses for next meeting(s):
  - Any topics from oneMKL TAB members?

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

Version of oneAPI Specification	Date
1.1-provisional-rev-3	21 September 2021
1.1-rev-1	12 November 2021

Resources

- oneAPI Main Page: https://www.oneapi.io
- 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, RNG, and LAPACK domains): https://github.com/oneapi-src/oneMKL