oneMKL Technical Advisory Board

Session 17

March 23, 2022

Agenda

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

oneMKL TAB Members

- Terry Cojean, Karlsruhe Institute of Technology (KIT)

- Hartwig Anzt, Karlsruhe Institute of Technology (KIT)
- Romain Dolbeau, SiPearl
- Mehdi Goli, Codeplay
- Mark Hoemmen, Stellar Science
- Nevin Liber, Argonne National Laboratory (ANL)
- Ye Luo, 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 TAB meeting cadence changed to bi-monthly
- oneAPI Math Kernel Library (oneMKL) Interfaces Project: LAPACK supported on Nvidia GPUs via cuSolver
- Intel oneMKL 2022.0 was released; Intel oneMKL 2022.1 to be released soon

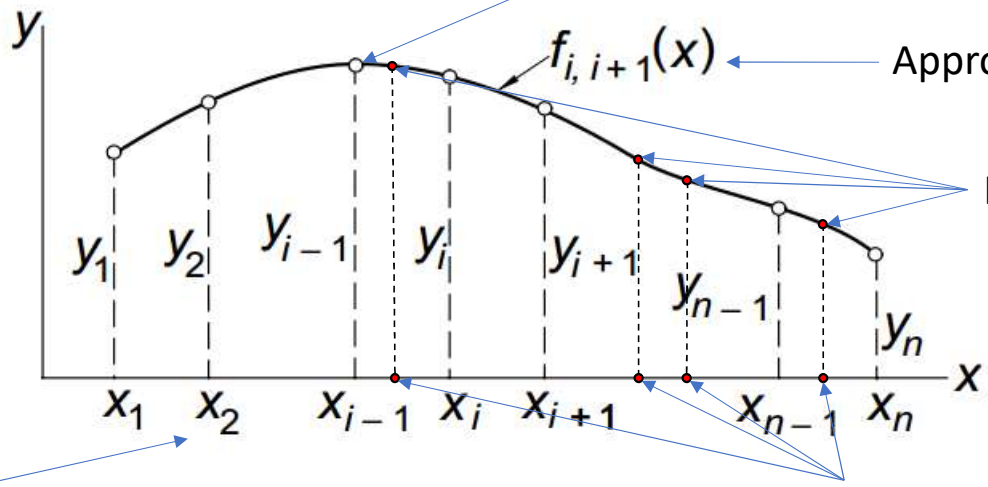
- oneAPI Image Processing Library (oneIPL) provisional specification v0.5: https://spec.oneapi.io/oneipl/latest/index.html

What Data Fitting is

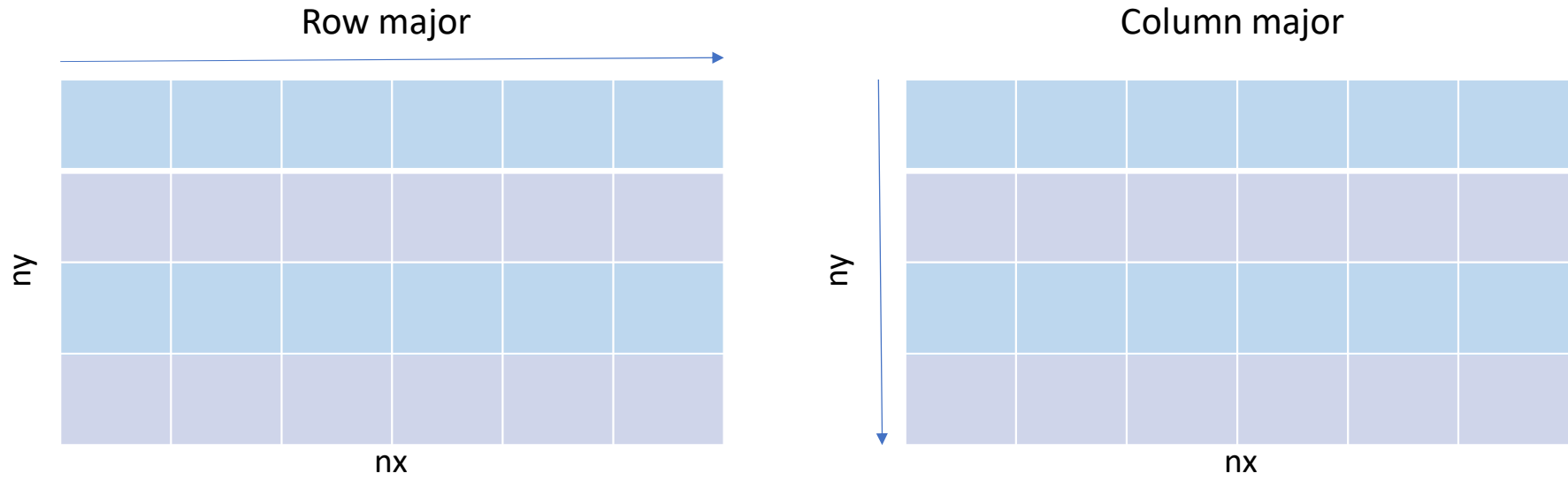
What Data Fitting is

- 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

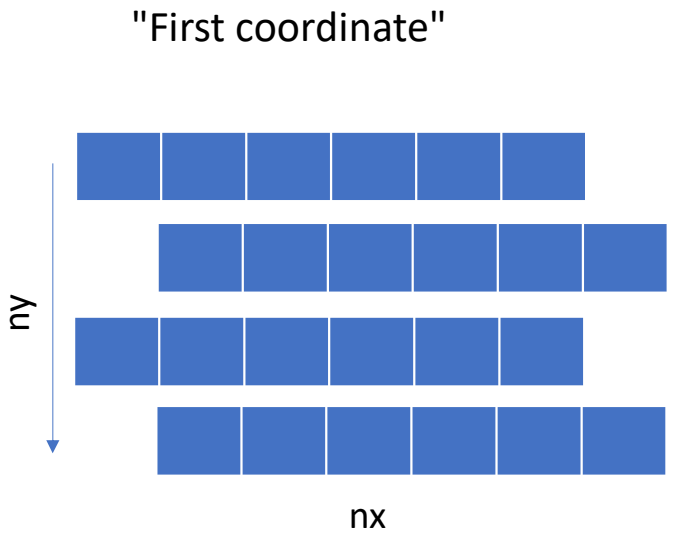


nx - quantity of partitions
ny - quantity of functions

Data Layouts for coefficients and functions

Suppose having multiple (ny) functions with same partitioning (nx points) to interpolate, but with non-contiguous memory layout (it may also happen if we want to perform coefficient computations only for part of functions in contiguous memory layout).

```
/* pseudocode */
for (int i = 0; i < quantity_of_functions; ++i) {
   spline.set_function_values(f_ptr[i], first_coordinate_indicator);
}
spline.construct();
interpolate(spline, … /*other parameters*/);
```



Independent memory for each set of coefficients/function

Does it make sense to add this layout for oneMKL? Do you have some use cases?

DPC++ Data Fitting APIs Updates

From Previous Discussion

- Why have a separate construct function, and not have the constructor do the work?
  - We want to let user control when the computations are performed. We can’t return event from constructor.
- Is it permitted to mix precision types? For example, double precision partition and single precision function?
  - No, only same floating-point type should be used for spline coefficients/functions/partitions and interpolation sites/results.
- Is it necessary to store the queue in the cubic spline?
  - Yes, we need context of execution inside spline.
- Why doesn't the spline own its own coefficients?
  - Lifetime of, e.g., coefficients may be longer than spline’s one. And if the spline will free user’s memory, the data will be lost. Also, we want to avoid deep copying of data and we prohibit copy constructor and assignment operator for this purpose.

Supported APIs

Support only USM APIs.
APIs are experimental and may be modified in future.

- Interpolate function
- Splines: linear / cubic Hermite

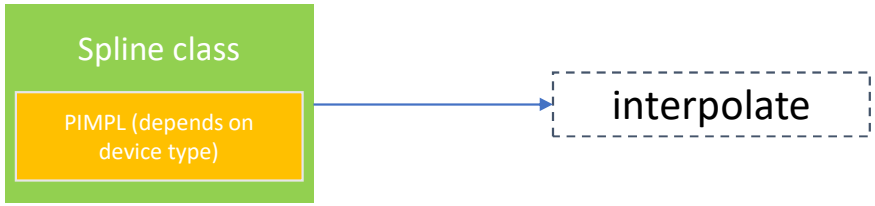
- Partitions: uniform / non-uniform / quasi-uniform
- Function values layouts: row major / column major
- Boundary conditions: free end / not-a-knot / periodic / 1st derivative / 2nd derivative
- ...

Which other types of spline you find useful for the Spec?

APIs* for spline-based Data Fitting

Support only USM APIs

- Spline class:
  - Templated by floating-point type, spline type and Dimension of the spline
- Interpolate function:
  - performs interpolation (including function values and derivatives computation)



Spline Interface

```
template<typename FpType, typename SplineType, int Dimensions>
class spline {
public:
    using value_type= FpType;
    using spline_type= cubic_spline::hermite;

    spline(const sycl::queue& queue, std::int64_t ny = 1, bool were_coeffs_computed = false);
    spline(const sycl::device& device, const sycl::context& context, std::int64_t ny = 1, bool were_coeffs_computed = false);
    // deleted copy/move constructors and assignment operators

    spline& set_partitions(FpType* input_data, std::int64_t nx, partition_hint PartitionHint = partition_hint::non_uniform);
    spline& set_function_values(FpType* input_ data, function_hint FunctionHint = function_hint::row_major);
    spline& set_coefficients(FpType* data, coefficient_hint CoeffHint = coefficient_hint::row_major);

    bool is_initialized() const;
    std::int64_t get_required_coeffs_size() const;
    sycl::event construct(const std::vector<sycl::event>& dependencies = {});
}
```

There are some splines that requires internal conditions and boundary conditions to be set. For such spline types, the following member functions must be called.

```
    spline& set_internal_conditions(FpType* input_ data);
    spline& set_boundary_conditions(bc_type BCType = bc_type::free_end, FpType value = {});
```

Interpolate function

```
template<typename Interpolant>
sycl::event interpolate(Interpolant & interpolant,
    typename Interpolant ::value_type* sites, std::int64_t n_sites, typename Interpolant :: value_type* results,
    std::bitset<32> derivatives_indicator, const std::vector<sycl::event>& dependencies,
    interpolate_hint ResultHint, site_hint SiteHint);
```

```
template<typename Interpolant>
sycl::event interpolate(sycl::queue& queue, const Interpolant & interpolant,
    typename Interpolant :: value_type* sites, std::int64_t n_sites, typename Interpolant :: value_type* results,
    std::bitset<32> derivatives_indicator, const std::vector<sycl::event>& dependencies, interpolate_hint
    ResultHint, site_hint SiteHint);
```

There are 2 more interfaces. These don't have parameters for derivatives

Example

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

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

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::spline<float, mkl_df::cubic_spline::hermite> hermite_spline(q, ny); // create spline object
        hermite_spline.set_partitions(partitions, nx /*, partition_hint */ ).set_coefficients(coeffs /*, coefficient_hint */ ).set_function_values(functions /*, function_hint */).
                set_internal_conditions(ic). set_boundary_conditions(bc_type /*, value_if_needed*/ ) // set parameters
        auto event = hermite_spline .construct(); // compute spline coefficients
        // prepare data for interpolation results
…
        event = mkl_df::interpolate(hermite_spline , sites, nsites, results, {event, other_dependencies}); // interpolate data
….
}
```

Summary

We're adding DPC++ Data Fitting interfaces to oneMKL Spec.

Waiting for feedback!

https://github.com/oneapi-src/oneAPI-spec/pull/413

Wrap-up

Next Steps

- Focuses for next meeting(s):
  - Overview/update on oneMKL Interfaces open source project
  - Any topics from oneMKL TAB members?

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

Resources

- oneAPI Main Page: https://www.oneapi.com/
- Latest release of oneMKL Spec (currently v. 1.1): 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