Math Special Interest Group

October 2, 2024

Discussion on Discrete Fourier Transform APIs: improving (type) safety for the `{s,g}et_value` member functions of the `descriptor` class template

Outline

- Targeted scope
  - Context reminder: how DFT descriptors are supposed to be used
  - Context reminder: how DFT descriptors have been used
  - Root cause and other related shortcomings
- Proposal
  - For configuration-setting member functions
  - For configuration-querying member functions
- Q&A

TARGETED SCOPE

Context reminder: how DFT descriptors are supposed to be used

oneMKL users interested in computing any DFT must first create a DFT `descriptor` object and commit it (to the desired configuration and the desired SYCL queue). For any DFT that involves a configuration different than default, users *must* communicate their intentions via the `set_value` member function of the `descriptor` class template. The counterpart `get_value` member function enables them to query configuration values and must sometimes be used to fully set a descriptor before computation. For instance,

```
// EXAMPLE: configuration of a DFT descriptor for a single-precision out-of-place 3D
// real DFT of lengths AxBxC, batched 5 times, using an externally-allocated USM workspace
// ------------------ EXPECTED USAGE ACCORDING TO THE SPECIFICATIONS ----------------------
namespace dft_ns = oneapi::mkl::dft;
dft_ns::descriptor<dft_ns::precision::SINGLE, dft_ns::domain::REAL> desc({A,B,C});
std::vector<std::int64_t> fwd_strides = {0, B*C,       C,       1};
std::vector<std::int64_t> bwd_strides = {0, B*(C/2+1), C/2 + 1, 1};
desc.set_value(dft_ns::config_param::FWD_STRIDES, fwd_strides);                     // non-POD data *cannot* be passed in variadic lists...
desc.set_value(dft_ns::config_param::BWD_STRIDES, bwd_strides);                     // non-POD data *cannot* be passed in variadic lists...
desc.set_value(dft_ns::config_param::PLACEMENT, dft_ns::config_value::NOT_INPLACE);
desc.set_value(dft_ns::config_param::NUMBER_OF_TRANSFORMS, std::int64(5));          // for real, std::int64_t? Strictly speaking, yes...

desc.set_value(dft_ns::config_param::FORWARD_SCALE, 1.0f/(A*B*C));                  // for real, 1.0f? Strictly speaking, yes...
desc.set_value(dft_ns::config_param::FWD_DISTANCE, A*B*C);
desc.set_value(dft_ns::config_param::BWD_DISTANCE, A*B*(C/2+1));
desc.set_value(dft_ns::config_param::WORKSPACE_PLACEMENT,
               dft_ns::config_value::WORKSPACE_EXTERNAL);
desc.commit(queue);
std::int64_t ws_size;
desc.get_value(dft_ns::config_param::WORKSPACE_BYTES, &ws_size);
float* dft_workspace = (float*) malloc_device(ws_byte_size, queue);
desc.set_workspace(dft_workspace);
// desc is now usable in (USM) compute routines
```

→ Would not even compile for any “implementation” (no implementation ***can*** align with the specifications)

→ Ignoring the stride-setting inconsistency, some possible UB does exist if usage is “intuitive” instead of strictly specification-abiding

Context reminder: how DFT descriptors have been used

oneMKL users interested in computing any DFT must first create a DFT `descriptor` object and commit it (to the desired configuration and the desired SYCL queue). For any DFT that involves a configuration different than default, users *must* communicate their intentions via the `set_value` member function of the `descriptor` class template. The counterpart `get_value` member function enables them to query configuration values and must sometimes be used to fully set a descriptor before computation. For instance,

```
// EXAMPLE: configuration of a DFT descriptor for a single-precision out-of-place 3D
// real DFT of lengths AxBxC, batched 5 times, using an externally-allocated USM workspace
// ----------------------------------- ACTUAL USAGE ---------------------------------------
namespace dft_ns = oneapi::mkl::dft;
dft_ns::descriptor<dft_ns::precision::SINGLE, dft_ns::domain::REAL> desc({A,B,C});
std::int64_t fwd_strides[4] = {0, B*C,       C,       1};
std::int64_t bwd_strides[4] = {0, B*(C/2+1), C/2 + 1, 1};
desc.set_value(dft_ns::config_param::FWD_STRIDES, fwd_strides);                     // assumed arg type: array of (rank + 1) std::int64_t
desc.set_value(dft_ns::config_param::BWD_STRIDES, bwd_strides);                     // assumed arg type: array of (rank + 1) std::int64_t
desc.set_value(dft_ns::config_param::PLACEMENT, dft_ns::config_value::NOT_INPLACE); // assumed arg type: dft_ns::config_value
desc.set_value(dft_ns::config_param::NUMBER_OF_TRANSFORMS, 5);                      // despite assumed arg type: std::int64_t.
                                                                                    // /!\ NOTE: possible UB in this case
desc.set_value(dft_ns::config_param::FORWARD_SCALE, 1.0f/(A*B*C));                  // despite assumed arg type: double (fine here though)
desc.set_value(dft_ns::config_param::FWD_DISTANCE, A*B*C);                          // assumed arg type: std::int64_t
desc.set_value(dft_ns::config_param::BWD_DISTANCE, A*B*(C/2+1));                    // assumed arg type: std::int64_t
desc.set_value(dft_ns::config_param::WORKSPACE_PLACEMENT,
               dft_ns::config_value::WORKSPACE_EXTERNAL);                           // assumed arg type: dft_ns::config_value
desc.commit(queue);
std::int64_t ws_size;
desc.get_value(dft_ns::config_param::WORKSPACE_BYTES, &ws_size);                    // assumed arg type: std::int64_t*
float* dft_workspace = (float*) malloc_device(ws_byte_size, queue);
desc.set_workspace(dft_workspace);
// desc is now usable in (USM) compute routines
```

→ implementations *cannot* align with the specs for vector-valued configuration parameters (so *something else* was done);

→ strict input validation cannot be done in most cases and implementations (must) blindly process input data assuming they're valid;

→ possible undefined behaviors exist (possible disaster if using negative integer literal value for an integer-valued parameter).

Root cause and other related shortcomings

The root cause of these issues lies in the declaration of the `{s,g}et_value` member functions of the `descriptor` class template:

```
namespace oneapi::mkl::dft {
    template <precision prec, domain dom>
    class descriptor {
    // private:
    //     using real_scalar_t = std::conditional_t<prec == precision::DOUBLE, double, float>
    public:
        void set_value(config_param param, ...); // 2nd (and only) arg. must be of the corresp. type of value for param
        void get_value(config_param param, ...); // 2nd (and only) arg. must be *a pointer* to the corresp. type of value for param
    };
}}}
```

(implicitly) enforcing a strictly specification-abiding usage according to the following table that specifies the types of the configuration value corresponding to every configuration parameter.

Value of param	Value of configuration value	
`config_param::FORWARD_DOMAIN`	`domain`	
`config_param::PRECISION`	`precision`	
`config_param::NUMBER_OF_TRANSFORMS, config_param::{F,B}WD_DISTANCE, config_param::DIMENSION, config_param::WORKSPACE_EXTERNAL_BYTES, config_param::LENGTHS (if 1D)`	`std::int64_t`	Note: default promotion of variadic lists' integer values stops at `int`...
`config_param::FORWARD_SCALE, config_param::BACKWARD_SCALE`	`real_scalar_t`	Note: default promotion of all variadic lists' f-p values to `double`...
`config_param::{FWD,BWD,INPUT,OUTPUT}_STRIDES`	`std::vector<std::int64_t>` → **Cannot** be done for `set_value`	
`config_param::COMMIT_STATUS, config_param::COMPLEX_STORAGE, config_param::PLACEMENT, config_param::WORKSPACE_PLACEMENT`	`config_value`	

PROPOSAL

For configuration-setting member functions

Suggested resolution:

- overload the `set_value` member function with typed alternatives enabling:
  - common current usage enabled by implementations (aligned with the current version of the specifications or not);
  - support for all the intended use cases, including the possible “convenient” ones currently ill-defined;
- resolve possible compile-time ambiguities related to type promotion with the latter using SFINAE to enable/ignore entry points;
- mark all functions that prevent strict input validation by design as “deprecated”;
- enforce strict input validation for member functions that enable it by design (*e.g.*, exception to be thrown if vector size is not as expected);
- specify explicitly what are the types assumed by the deprecated variadic member.

```
namespace oneapi::mkl::dft {
    template <precision prec, domain dom>
    class descriptor {
    private:
         using real_scalar_t = std::conditional_t<prec == precision::DOUBLE, double, float>; // Suggested to add, also relevant for set_workspace
    public:
        void set_value(config_param param, config_value value);
        void set_value(config_param param, std::int64_t value);
        void set_value(config_param param, real_scalar_t value);
        [[deprecated(“Use set_value(config_param, const std::vector<std::int64_t>&), instead.”)]]
        void set_value(config_param param, const std::int64_t* value);  // currently enabled for strides, blindly dereferencing value[0 – rank]
        void set_value(config_param param, const std::vector<std::int64_t>& value);
        template <typename T,  std::enable_if_t<std::is_integral_v<T>, bool> = true>
        void set_value(config_param param, T value) {
            set_value(param, static_cast<std::int64_t>(value));         // set_value(config_param::NUMBER_OF_TRANSFORMS, 4) is ambiguous otherwise
        }
        template <typename T, std::enable_if_t<std::is_floating_point_v<T>, bool> = true>
        void set_value(config_param param, T value) {
            set_value(param, static_cast<real_scalar_t>(value));
        }
        [[deprecated("This set_value method is deprecated.")]]
        void set_value(config_param param, ...);                       // currently enabled for all param, root of all issues
    };
}}}
```

For configuration-querying member functions

Suggested resolution:

- overload the get_value member function with typed alternatives enabling:
    - common current usage enabled by implementations (aligned with the current version of the specifications or not);
    - support for all the intended use cases;
- mark all functions (or usages thereof) that prevent strict input validation by design as "deprecated";
- add a const qualifier for all of them as they leave the calling object unchanged;
- enforce strict input validation for member functions that enable it by design (*e.g.*, exception to be thrown if vector size is not as expected);
- specify explicitly what are the types assumed by the deprecated variadic member.

```
namespace oneapi::mkl::dft {
    template <precision prec, domain dom>
    class descriptor {
    private:
         using real_scalar_t = std::conditional_t<prec == precision::DOUBLE, double, float>; // also relevant for set_workspace
    public:
        void get_value(config_param param, config_value* value_ptr) const;
        void get_value(config_param param, domain* value_ptr) const;
        void get_value(config_param param, precision* value_ptr) const;
        void get_value(config_param param, std::int64_t* value_ptr) const;   // possible deprecated *usage* thereof may warrant a *runtime* warning
        void get_value(config_param param, real_scalar_t* value_ptr) const;
        void get_value(config_param param, std::vector<std::int64_t>* value_ptr) const;
        [[deprecated("This get_value method is deprecated.")]]
        void get_value(config_param param, ...) const; // NOTE: adding const here breaks BWD compatibility, though…
    };
}}}
```

Suggested changes: PR#593

Q&A