Discrete Fourier Transform Interface for oneMKL

Finlay Marno

Staff Software Engineer

Contents

- Project Goals
- Why do we care?
- Features of other FFT interfaces
- Example
- Parameters rundown
- Difficulties
- Future timeline

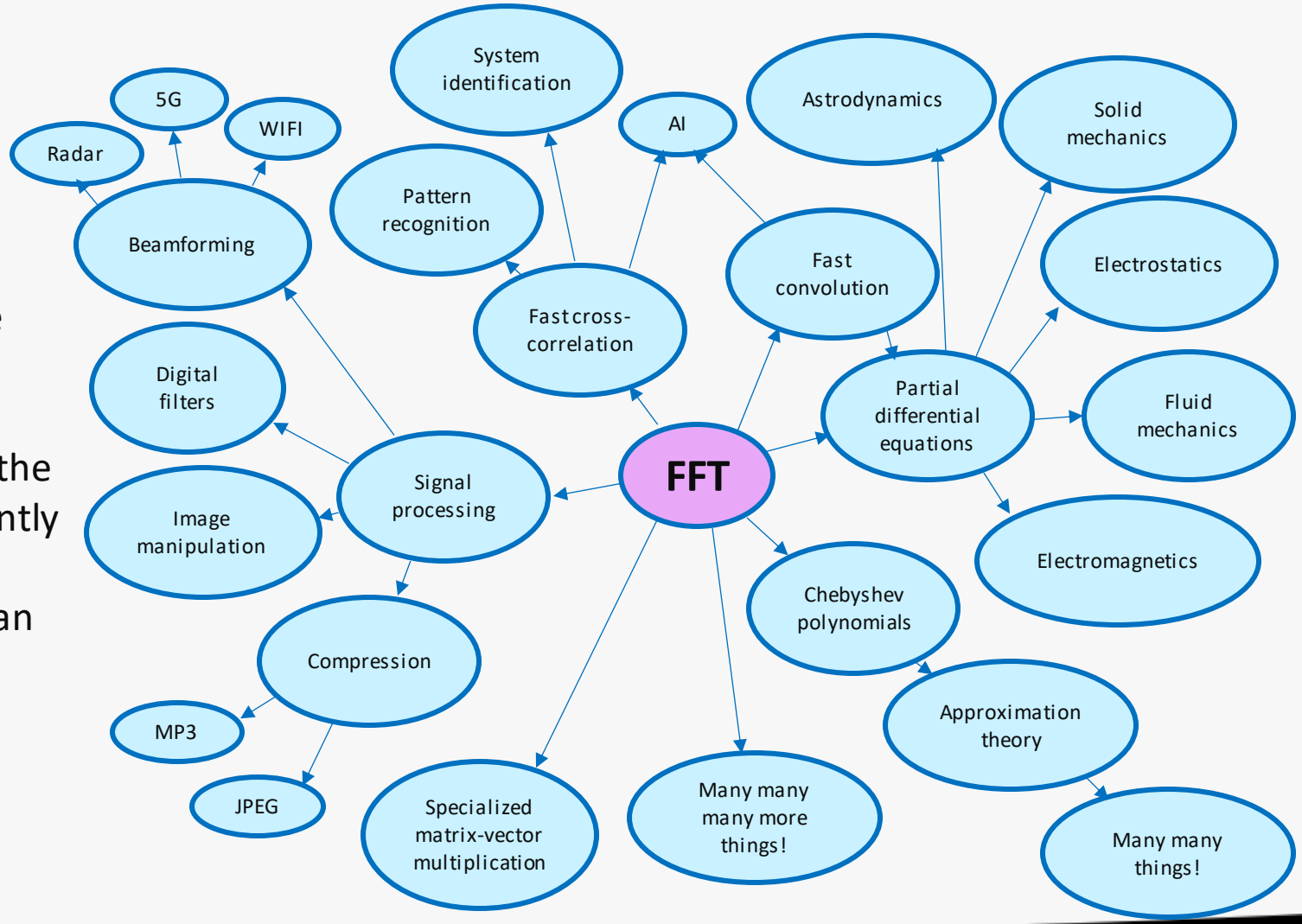
The Goal

- Cross-Platform SYCL DFT interface backed by existing implementations
- Low overhead for backends
- Conforms to the open source oneMKL DFT interface as defined at https://spec.oneapi.io/versions/latest/elements/oneMKL/source/domains/dft/dft.html

Why care about FFTs?

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{i2\pi}{N} kn}$$

- The naïve implementation of a Discrete Fourier Transform (DFT) has run-time complexity $O(N^2)$
- Fast Fourier Transforms (FFT) compute the DFT, are $O(N \log(N))$, and can be efficiently computed by GPUs
- If a problem can be posed as a DFT, it can be solved much more quickly!



General Features of an FFT interface

- A plan or description of the problem
- A commit stage (can be at the same time as above)
- Execution calls
- Libraries that follow this: FFTW, closed source OneMKL CPU/GPU, cuFFT, rocFFT, vkFFT
- This allows for:
  - Dynamic code generation for the problem
  - Pre-calculation of any data needed
  - Reuse of configuration

Example

```
namespace dft = oneapi::mkl::dft;
std::vector<std::complex<float>> input_data(N);
std::vector<std::complex<float>> output_data(N);

dft::descriptor<dft::precision::SINGLE, dft::domain::COMPLEX> desc(static_cast<std::int64_t>(N));

desc.set_value(dft::config_param::PLACEMENT, dft::config_value::NOT_INPLACE);
desc.set_value(dft::config_param::NUMBER_OF_TRANSFORMS, std::int64_t(1));

desc.commit(oneapi::mkl::backend_selector<oneapi::mkl::backend::mklgpu>{ gpu_queue });

{
    sycl::buffer<std::complex<float>> input_buffer(input_data.data(), sycl::range<1>(N));
    sycl::buffer<std::complex<float>> output_buffer(output_data.data(), sycl::range<1>(N));
    dft::compute_forward<decltype(desc), std::complex<float>, std::complex<float>>(desc,
        input_buffer, output_buffer);
}
```

Taken from https://github.com/oneapi-src/oneMKL/blob/develop/examples/dft/compile_time_dispatching/complex_fwd_buffer_mklgpu.cpp

Parameters

Read-Only Parameters

- Part of the descriptor
- Define only at construction time
- Commit status

Parameter	Description
FORWARD_DOMAIN	Is the forward (time) domain real or complex valued
DIMENSION	The number of dimensions
LENGTHS	The length of each dimension in elements
PRECISION	Single or double floating-point precision
COMMIT_STATUS	Has the descriptor been committed

More obvious parameters

Parameter	Description
FORWARD_SCALE	Multiply forward output by scalar value
BACKWARD_SCALE	Multiply backward output by scalar value (1/N to get back to input
NUMBER_OF_TRANSFORMS	Number of transforms per execution
PLACEMENT	INPLACE or NOT-INPLACE
WORKSPACE*	Allow additional scratch space to be allocated
ORDERING*	Allow backward domain data to be in an implementation defined order
TRANSPOSE*	Output is transposed
PACKED_FORMAT	Compressed format of the backward domain values for real to complex transforms (only CCE_FORMAT supported)

* Not implemented by any backend

Less Obvious Parameters
COMPLEX_STORAGE

- Only applies when doing complex to complex transforms
- COMPLEX_COMPLEX
  - Applies forward and backward
  - Array of std::complex<> values

Real+Imag	Real+Imag	Real+Imag	Real+Imag

- REAL_REAL
  - Applies forward and backward
  - Two arrays of float/double

Real	Real	Real	Real
Imag	Imag	Imag	Imag

Less Obvious Parameters
REAL_STORAGE

- Only applies when doing real to complex transforms
- REAL_REAL
  - Only forward (time) domain
  - Array of float/double

Real	Real	Real	Real

Less Obvious Parameters CONJUGATE_EVEN_STORAGE

- Only applies when doing real to complex transforms
- COMPLEX_COMPLEX
  - Only backward (frequency) domain
  - Array of float/double

Real	Imag	Real	Imag

What is Conjugate Even Form?

- The output of real to complex Fourier transforms has conjugate symmetry
- output[x] == complex conjugate(output[N-x])
- output[0] not included
- Only floor(N/2) + 1 elements are needed (for a 1D transform)
- It is always the contiguous dimension that is shortened
- e.g. 32x16x8 => 32x16x5

Less Obvious Parameters FWD/BWD_DISTANCE

- Distance between two transforms for batched transforms in the forward/backward domain
- Counted in elements
- For real to complex transforms you need to account for conjugate even form e.g. 16x8 transform must have a BWD_DISTANCE >= 16*5.

Less Obvious Parameters INPUT/OUTPUT_STRIDES

- Index multipliers to read a given element – not the dimension of the input array
  - e.g. for a 3D transform with strides {s0,s1,s2,s3}, element {k1,k2,k3} will be at position arr[s0+s1*k1+s2*k2+s3*k3]
  - A tightly packed 8x8x8 transform will have strides {0,64,8,1}
- Account for conjugate even for in the backward domain when doing real to complex transform
  - e.g. 8x8x8 => 8x8x5 and strides of {0,40,5,1}

Compute Forward 1

```
namespace oneapi::mkl::dft {
template <typename descriptor_type, typename data_type>
void compute_forward(descriptor_type &desc,
                     sycl::buffer<data_type, 1> &inout);
}
```

Example configuration for a 16x8 real domain problem

PLACEMENT	OUTPUT_STRIDES	BWD_DISTANCE	CONJUGATE_EVEN_ST...
INPLACE	{0,5,1}	80	COMPLEX_COMPLEX

Other configuration parameters may be needed (INPUT_STRIDES, FWD_DISTANCE, FORWARD_SCALE, NUMBER_OF_TRANSFORMS)

Compute Forward 2

```
namespace oneapi::mkl::dft {
template <typename descriptor_type, typename data_type>
void compute_forward(descriptor_type &desc,
                     sycl::buffer<data_type, 1> &inout_re,
                     sycl::buffer<data_type, 1> &inout_im);
}
```

Example configuration for a 16x8 complex domain problem. `data_type` would be float or double.

PLACEMENT	OUTPUT_STRIDES	BWD_DISTANCE	COMPLEX_STORAGE
INPLACE	{0,8,1}	128	REAL_REAL

Other configuration parameters may be needed (INPUT_STRIDES, FWD_DISTANCE, FORWARD_SCALE, NUMBER_OF_TRANSFORMS)

Compute Forward 3

```
namespace oneapi::mkl::dft {
template <typename descriptor_type, typename input_type, typename output_type>
void compute_forward(descriptor_type &desc,
                     sycl::buffer<input_type, 1>  &in,
                     sycl::buffer<output_type, 1> &out);
}
```

Example configuration for a 16x8 complex domain problem. `input_type` and `output_type` would be `std::complex<>`.

PLACEMENT	OUTPUT_STRIDES	BWD_DISTANCE	COMPLEX_STORAGE
NOT_INPLACE	{0,8,1}	128	COMPLEX_COMPLEX

Other configuration parameters may be needed (INPUT_STRIDES, FWD_DISTANCE, FORWARD_SCALE, NUMBER_OF_TRANSFORMS)

Compute Forward 4

```
namespace oneapi::mkl::dft {
template <typename descriptor_type, typename input_type, typename output_type>
void compute_forward(descriptor_type &desc,
                     sycl::buffer<input_type, 1>  &in_re,
                     sycl::buffer<input_type, 1>  &in_im,
                     sycl::buffer<output_type, 1> &out_re,
                     sycl::buffer<output_type, 1> &out_im);
}
```

Example configuration for a 32x32 complex domain problem. input_type and output_type would be std::complex<>.

PLACEMENT	OUTPUT_STRIDES	BWD_DISTANCE	COMPLEX_STORAGE
NOT_INPLACE	{0,32,1}	1024	REAL_REAL

Other configuration parameters may be needed (INPUT_STRIDES, FWD_DISTANCE, FORWARD_SCALE, NUMBER_OF_TRANSFORMS)

Backward/USM Support

There are also equivalent compute_backward and USM functions for all the previous examples

```
namespace oneapi::mkl::dft {
template <typename descriptor_type, typename data_type>
sycl::event compute_backward(descriptor_type &desc,
                             data_type *inout,
                             const std::vector<cl::sycl::event> &dependencies = {});
}
```

Supported Backends

- MKLCPU – Implemented by Intel
- MKLGPU – Implemented by Codeplay
- cuFFT – Implemented by Codeplay

Backend Difficulties

- cuFFT and MKLCPU use input/output distance
  - Got around this by creating two backend specific plans/descriptors, one for each direction
  - Caused issues for the MKLCPU backend when the strides where invalid for one of the directions
  - Overhead might not be so bad if there are caching mechanisms in place
- cuFFT doesn't do scaling - https://github.com/oneapi-src/oneMKL/issues/313

Specification Difficulties

- Input/output strides vs forward/backward distance
- This is mostly just inconsistent and easy to work around
- Forward/Backward can save you from creating more descriptors
- Most backends use input/output - cuFFT, MKLCPU, vkFFT, rocFFT

Specification Difficulties

- Compile time dispatch doesn't work with multiple backends linked

Function Dispatch

- Runtime
  - Map from device type (`x86cpu`, `intelgpu`, `nvidiagpu`...) to function pointer table for each backend
  - This means only one backend per device type at a time.
    - Can't have both rocFFT and cuFFT backend for `nvidiagpu` in one build
    - This is the same as the other interfaces
- Compile Time
  - Function takes an extra parameter, the `backend_selector<>`
  - Unique function signature
  - Easy for the compiler to inline

Specification Difficulties

- Compile time dispatch doesn't work with multiple backends linked
- https://github.com/oneapi-src/oneMKL/issues/309
- Each backend defines compute_forward and compute_backward functions with the same name and arguments
- RNG backend has a similar situation but handles this by always using dynamic dispatch via virtual functions.

Specification Difficulties

- Descriptor lifetime needs some clarity
- Currently it is underspecified what happens when a transform is queued or in progress and the descriptor is destroyed.
- e.g. https://github.com/oneapi-src/oneMKL/blob/develop/src/dft/backends/mklcpu/commit.cpp#L73
  - Calls DftiFreeDescriptor with no synchronization
  - Similar story with cuFFT and MKLGPU backend.
- There could be some reference counting in the background but there are no guarantees

Future Timeline

- rocFFT – Aim for end of June
- More tests
  - FWD/BWD_DISTANCE
  - INPUT/OUTPUT_STRIDES
  - FORWARD/BACKWARD_SCALE

Acknowledgements

- Thank you to our colleagues at Intel in the FastFour team for their code reviews and help with the design and implementation of the backends

Please email us with sizes and configurations of interest

sycl@codeplay.com