rocFFT Documentation

Release 1.0.11

Advanced Micro Devices

May 28, 2021

CONTENTS:

CHAPTER

ONE

ROCFFT

1.1 Introduction

The rocFFT library is an implementation of the discrete Fast Fourier Transform (FFT) written in HIP for GPU devices. The code is open and hosted here: https://github.com/ROCmSoftwarePlatform/rocFFT

The rocFFT library:

- Provides a fast and accurate platform for calculating discrete FFTs.
- Supports single and double precision floating point formats.
- Supports 1D, 2D, and 3D transforms.
- Supports computation of transforms in batches.
- Supports real and complex FFTs.
- Supports arbitrary lengths, with optimizations for combinations of powers of 2, 3, and 5.

1.2 FFT Computation

The FFT is an implementation of the Discrete Fourier Transform (DFT) that makes use of symmetries in the DFT definition to reduce the mathematical complexity from $O(N^2)$ to $O(N \log N)$.

What is computed by the library? Here are the formulas:

For a 1D complex DFT:

$\tilde{x}_j = \sum_{k=0}^{n-1} x_k \exp\left(\pm i \frac{2\pi jk}{n}\right)$ for $j = 0, 1, \ldots, n-1$

where, x_k are the complex data to be transformed, $\tilde{x}_j$ are the transformed data, and the sign $\pm$ determines the direction of the transform: $-$ for forward and $+$ for backward.

For a 2D complex DFT:

$\tilde{x}_{jk} = \sum_{q=0}^{m-1} \sum_{r=0}^{n-1} x_{rq} \exp\left(\pm i \frac{2\pi jr}{n}\right) \exp\left(\pm i \frac{2\pi kq}{m}\right)$

for $j = 0, 1, \ldots, n-1$ and $k = 0, 1, \ldots, m-1$, where, x_{rq} are the complex data to be transformed, $\tilde{x}_{jk}$ are the transformed data, and the sign $\pm$ determines the direction of the transform.

For a 3D complex DFT:

$\tilde{x}_{jkl} = \sum_{s=0}^{p-1} \sum_{q=0}^{m-1} \sum_{r=0}^{n-1} x_{rqs} \exp\left(\pm i \frac{2\pi jr}{n}\right) \exp\left(\pm i \frac{2\pi kq}{m}\right) \exp\left(\pm i \frac{2\pi ls}{p}\right)$

for $j = 0, 1, \ldots, n-1$ and $k = 0, 1, \ldots, m-1$ and $l = 0, 1, \ldots, p-1$, where x_{rqs} are the complex data to be transformed, $\tilde{x}_{jkl}$ are the transformed data, and the sign $\pm$ determines the direction of the transform.

1.3 Library Setup and Cleanup

At the beginning of the program, before any of the library APIs are called, the function *rocfft_setup()* has to be called. Similarly, the function *rocfft_cleanup()* has to be called at the end of the program. These APIs ensure resources are properly allocated and freed.

1.4 Workflow

In order to compute an FFT with rocFFT, a plan has to be created first. A plan is a handle to an internal data structure that holds the details about the transform that the user wishes to compute. After the plan is created, it can be executed (a separate API call) with the specified data buffers. The execution step can be repeated any number of times with the same plan on different input/output buffers as needed. And when the plan is no longer needed, it gets destroyed.

To do a transform,

1. Initialize the library by calling *rocfft_setup()*.
2. Create a plan, for each distinct type of FFT needed:
    - To create a plan, do either of the following
        - If the plan specification is simple, call *rocfft_plan_create()* and specify the value of the fundamental parameters.
        - If the plan has more details, first a plan description is created with *rocfft_plan_description_create()*, and additional APIs such as *rocfft_plan_description_set_data_layout()* are called to specify plan details. And then, *rocfft_plan_create()* is called with the description handle passed to it along with other details.
    - Optionally, allocate a work buffer for the plan:
        - Call *rocfft_plan_get_work_buffer_size()* to check the size of work buffer required by the plan.
        - If a nonzero size is required:
            * Create an execution info object with *rocfft_execution_info_create()*.
            * Allocate a buffer using `hipMalloc()` and pass the allocated buffer to *rocfft_execution_info_set_work_buffer()*.
3. Execute the plan:
    - The execution API *rocfft_execute()* is used to do the actual computation on the data buffers specified.
    - Extra execution information such as work buffers and compute streams are passed to *rocfft_execute()* in the *rocfft_execution_info* object.
    - *rocfft_execute()* can be called repeatedly as needed for different data, with the same plan.
    - If the plan requires a work buffer but none was provided, *rocfft_execute()* will automatically allocate a work buffer and free it when execution is finished.
4. If a work buffer was allocated:
    - Call `hipFree()` to free the work buffer.
    - Call *rocfft_execution_info_destroy()* to destroy the execution info object.
5. Destroy the plan by calling *rocfft_plan_destroy()*.
6. Terminate the library by calling *rocfft_cleanup()*.

1.5 Example

```
#include <iostream>
#include <vector>
#include "hip/hip_runtime_api.h"
#include "hip/hip_vector_types.h"
#include "rocfft.h"

int main()
{
        // rocFFT gpu compute
        // ========================================

        rocfft_setup();

        size_t N = 16;
        size_t Nbytes = N * sizeof(float2);

        // Create HIP device buffer
        float2 *x;
        hipMalloc(&x, Nbytes);

        // Initialize data
        std::vector<float2> cx(N);
        for (size_t i = 0; i < N; i++)
        {
                cx[i].x = 1;
                cx[i].y = -1;
        }

        //  Copy data to device
        hipMemcpy(x, cx.data(), Nbytes, hipMemcpyHostToDevice);

        // Create rocFFT plan
        rocfft_plan plan = nullptr;
        size_t length = N;
        rocfft_plan_create(&plan, rocfft_placement_inplace,
             rocfft_transform_type_complex_forward, rocfft_precision_single,
             1, &length, 1, nullptr);

        // Check if the plan requires a work buffer
        size_t work_buf_size = 0;
        rocfft_plan_get_work_buffer_size(plan, &work_buf_size);
        void* work_buf = nullptr;
        rocfft_execution_info info = nullptr;
        if(work_buf_size)
        {
                rocfft_execution_info_create(&info);
                hipMalloc(&work_buf, work_buf_size);
                rocfft_execution_info_set_work_buffer(info, work_buf, work_buf_size);
        }
```

(continues on next page)

(continued from previous page)

```
        // Execute plan
        rocfft_execute(plan, (void**) &x, nullptr, info);

        // Wait for execution to finish
        hipDeviceSynchronize();

        // Clean up work buffer
        if(work_buf_size)
        {
                hipFree(work_buf);
                rocfft_execution_info_destroy(info);
        }

        // Destroy plan
        rocfft_plan_destroy(plan);

        // Copy result back to host
        std::vector<float2> y(N);
        hipMemcpy(y.data(), x, Nbytes, hipMemcpyDeviceToHost);

        // Print results
        for (size_t i = 0; i < N; i++)
        {
                std::cout << y[i].x << ", " << y[i].y << std::endl;
        }

        // Free device buffer
        hipFree(x);

        rocfft_cleanup();

        return 0;
}
```

1.6 Plans

A plan is the collection of (almost) all the parameters needed to specify an FFT computation. A rocFFT plan includes the following information:

- Type of transform (complex or real)
- Dimension of the transform (1D, 2D, or 3D)
- Length or extent of data in each dimension
- Number of datasets that are transformed (batch size)
- Floating-point precision of the data
- In-place or not in-place transform
- Format (array type) of the input/output buffer
- Layout of data in the input/output buffer

The rocFFT plan does not include the following parameters:

- The handles to the input and output data buffers.
- The handle to a temporary work buffer (if needed).
- Other information to control execution on the device.

These parameters are specified when the plan is executed.

1.7 Data

The input/output buffers that hold the data for the transform must be allocated, initialized and specified to the library by the user. For larger transforms, temporary work buffers may be needed. Because the library tries to minimize its own allocation of memory regions on the device, it expects the user to manage work buffers. The size of the buffer needed can be queried using *rocfft_plan_get_work_buffer_size()* and after their allocation can be passed to the library by *rocfft_execution_info_set_work_buffer()*. The samples in the source repository show how to use these.

1.8 Transform and Array types

There are two main types of FFTs in the library:

- Complex FFT - Transformation of complex data (forward or backward); the library supports the following two array types to store complex numbers:
  1. Planar format - where the real and imaginary components are kept in 2 separate arrays:
     - Buffer1: `RRRRR...`
     - Buffer2: `IIIII...`
  2. Interleaved format - where the real and imaginary components are stored as contiguous pairs in the same array:
     - Buffer: `RIRIRIRIRI...`
- Real FFT - Transformation of real data. For transforms involving real data, there are two possibilities:
  - Real data being subject to forward FFT that results in complex data (Hermitian).
  - Complex data (Hermitian) being subject to backward FFT that results in real data.

The library provides the *rocfft_transform_type* and *rocfft_array_type* enums to specify transform and array types, respectively.

1.9 Batches

The efficiency of the library is improved by utilizing transforms in batches. Sending as much data as possible in a single transform call leverages the parallel compute capabilities of devices (GPU devices in particular), and minimizes the penalty of control transfer. It is best to think of a device as a high-throughput, high-latency device. Using a networking analogy as an example, this approach is similar to having a massively high-bandwidth pipe with very high ping response times. If the client is ready to send data to the device for compute, it should be sent in as few API calls as possible, and this can be done by batching. rocFFT plans have a parameter *number_of_transforms* (this value is also referred to as batch size in various places in the document) in *rocfft_plan_create()* to describe the number of transforms being requested. All 1D, 2D, and 3D transforms can be batched.

1.10 Strides and Distances

Strides and distances enable users to specify custom layout of data using *rocfft_plan_description_set_data_layout()*.

For 1D data, if `strides[0] == strideX == 1`, successive elements in the first dimension (dimension index 0) are stored contiguously in memory. If `strideX` is a value greater than 1, gaps in memory exist between each element of the vector. For multi-dimensional cases; if `strides[1] == strideY == LenX` for 2D data and `strides[2] == strideZ == LenX * LenY` for 3D data, no gaps exist in memory between each element, and all vectors are stored tightly packed in memory. Here, `LenX`, `LenY`, and `LenZ` denote the transform lengths `lengths[0]`, `lengths[1]`, and `lengths[2]`, respectively, which are used to set up the plan.

Distance is the stride that exists between corresponding elements of successive FFT data instances (primitives) in a batch. Distance is measured in units of the memory type; complex data measures in complex units, and real data measures in real units. For tightly packed data, the distance between FFT primitives is the size of the FFT primitive, such that `dist == LenX` for 1D data, `dist == LenX * LenY` for 2D data, and `dist == LenX * LenY * LenZ` for 3D data. It is possible to set the distance of a plan to be less than the size of the FFT vector; typically 1 when doing column (strided) access on packed data. When computing a batch of 1D FFT vectors, if `distance == 1`, and `strideX == length(vector)`, it means data for each logical FFT is read along columns (in this case along the batch). You must verify that the distance and strides are valid, such that each logical FFT instance is not overlapping with any other; if not valid, undefined results may occur. A simple example would be to perform a 1D length 4096 on each row of an array of 1024 rows x 4096 columns of values stored in a column-major array, such as a FORTRAN program might provide. (This would be equivalent to a C or C++ program that has an array of 4096 rows x 1024 columns stored in a row-major manner, on which you want to perform a 1D length 4096 transform on each column.) In this case, specify the strides as [1024] and distance as 1.

1.11 Result placement

The API supports both in-place and not in-place transforms via the *rocfft_result_placement* enum. With in-place transforms, only input buffers are provided to the execution API, and the resulting data is written to the same buffer, overwriting the input data. With not in-place transforms, distinct output buffers are provided, and the results are written into the output buffer.

Note that rocFFT may still modify the input buffer even if a transform is requested to be not in-place. Real-complex transforms in particular are more efficient if they can modify the original input.

1.11.1 Real data

When real data is subject to DFT, the resulting complex output data follows a special property. About half of the output is redundant because they are complex conjugates of the other half. This is called the Hermitian redundancy. So, for space and performance considerations, it is only necessary to store the non-redundant part of the data. Most FFT libraries use this property to offer specific storage layouts for FFTs involving real data. rocFFT provides three enumeration values for *rocfft_array_type* to deal with real data FFTs:

- REAL (`rocfft_array_type_real`)
- HERMITIAN_INTERLEAVED (`rocfft_array_type_hermitian_interleaved`)
- HERMITIAN_PLANAR (`rocfft_array_type_hermitian_planar`)

The REAL (`rocfft_array_type_real`) enum specifies that the data is purely real. This can be used to feed real input or get back real output. The HERMITIAN_INTERLEAVED (`rocfft_array_type_hermitian_interleaved`) and HERMITIAN_PLANAR (`rocfft_array_type_hermitian_planar`) enums are similar to the corresponding full complex enums in the way they store real and imaginary components, but store only about half of the complex output.

Client applications can do just a forward transform and analyze the output or they can process the output and do a backward transform to get back real data. This is illustrated in the following figure.

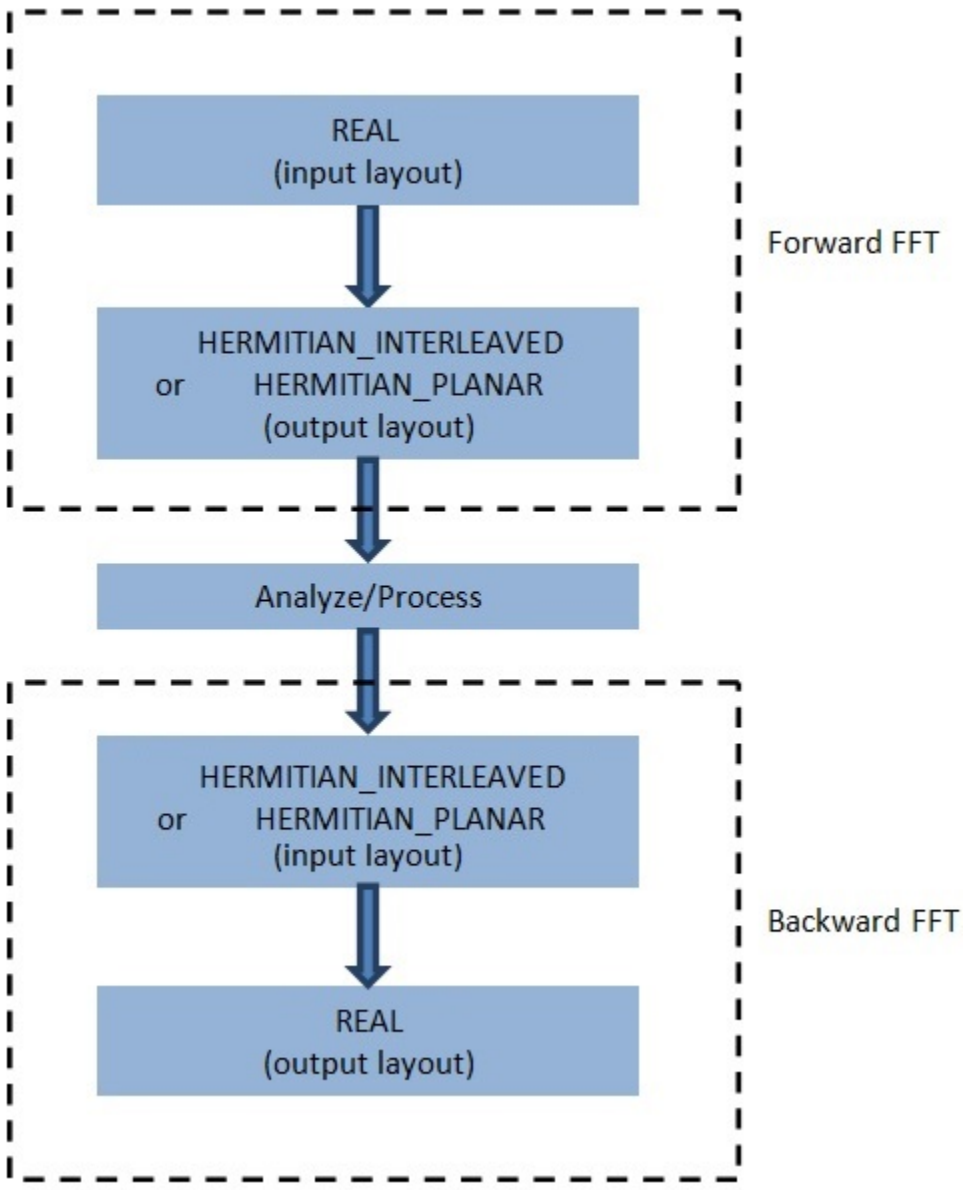

Fig. 1: **Forward and Backward Real FFTs**

Let us consider a 1D real FFT of length N. The full output looks as shown in following figure.

Here, C* denotes the complex conjugate. Since the values at indices greater than $N/2$ can be deduced from the first half of the array, rocFFT stores data only up to the index $N/2$. This means that the output contains only $1 + N/2$ complex elements, where the division $N/2$ is rounded down. Examples for even and odd lengths are given below.

Example for $N = 8$ is shown in following figure.

Example for $N = 7$ is shown in following figure.

For length 8, only $(1 + 8/2) = 5$ of the output complex numbers are stored, with the index ranging from 0 through 4. Similarly for length 7, only $(1 + 7/2) = 4$ of the output complex numbers are stored, with the index ranging from

C(0)	C(1)	C(2)	…	C(N/2)	…	C(N-2) = C*(2)	C(N-1) = C*(1)

Fig. 2: **1D Real FFT of Length N**

C(0)	C(1)	C(2)	C(3)	C(4)	C(5) = C*(3)	C(6) = C*(2)	C(7) = C*(1)

Fig. 3: **Example for N = 8**

0 through 3. For 2D and 3D FFTs, the FFT length along the innermost dimension is used to compute the $(1 + N/2)$ value. This is because the FFT along the innermost dimension is computed first and is logically a real-to-hermitian transform. The FFTs along other dimensions are computed next, and they are simply 'complex-to-complex' transforms. For example, assuming $Lengths[2]$ is used to set up a 2D real FFT, let $N1 = Lengths[1]$, and $N0 = Lengths[0]$. The output FFT has $N1 * (1 + N0/2)$ complex elements. Similarly, for a 3D FFT with $Lengths[3]$ and $N2 = Lengths[2]$, $N1 = Lengths[1]$, and $N0 = Lengths[0]$, the output has $N2 * N1 * (1 + N0/2)$ complex elements.

1.11.2 Supported array type combinations

Not In-place transforms:

- Forward: REAL to HERMITIAN_INTERLEAVED
- Forward: REAL to HERMITIAN_PLANAR
- Backward: HERMITIAN_INTERLEAVED to REAL
- Backward: HERMITIAN_PLANAR to REAL

In-place transforms:

- Forward: REAL to HERMITIAN_INTERLEAVED
- Backward: HERMITIAN_INTERLEAVED to REAL

C(0)	C(1)	C(2)	C(3)	C(4) = C*(3)	C(5) = C*(2)	C(6) = C*(1)

Fig. 4: **Example for N = 7**

1.11.3 Setting strides

The library currently requires the user to explicitly set input and output strides for real transforms for non simple cases. See the following examples to understand what values to use for input and output strides under different scenarios. These examples show typical usages, but the user can allocate the buffers and choose data layout according to their need.

1.11.4 Examples

The following figures and examples explain in detail the real FFT features of this library.

Here is a schematic that illustrates the forward 1D FFT (real to hermitian).

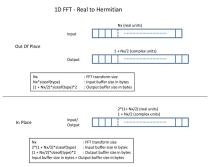

Fig. 5: **1D FFT - Real to Hermitian**

Below is a schematic that shows an example of not in-place transform with even N and how strides and distances are set.

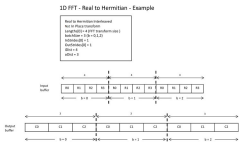

Fig. 6: **1D FFT - Real to Hermitian, Example 1**

Below is a schematic that shows an example of in-place transform with even N and how strides and distances are set. Notice that even though we are dealing with only 1 buffer (in-place), the output strides/distance can take different values compared to input strides/distance.

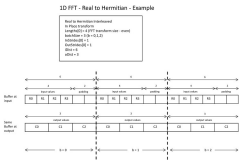

Fig. 7: **1D FFT - Real to Hermitian, Example 2**

Below is a schematic that shows an example of in-place transform with odd N and how strides and distances are set. Notice that even though we are dealing with only 1 buffer (in-place), the output strides/distance can take different values compared to input strides/distance.

And here is a schematic that illustrates the backward 1D FFT (hermitian to real).

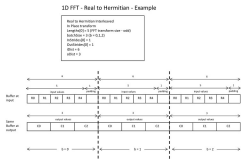

Fig. 8: **1D FFT - Real to Hermitian, Example 3**

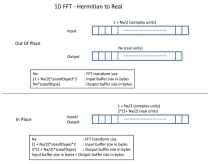

Fig. 9: **1D FFT - Hermitian to Real**

Below is a schematic that shows an example of in-place transform with even N and how strides and distances are set. Notice that even though we are dealing with only 1 buffer (in-place), the output strides/distance can take different values compared to input strides/distance.

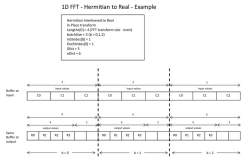

Fig. 10: **1D FFT - Hermitian to Real, Example**

And here is a schematic that illustrates the in-place forward 2D FFT (real to hermitian) .

Below is a schematic that shows an example of in-place 2D transform and how strides and distances are set. Notice that even though we are dealing with only 1 buffer (in-place), the output strides/distance can take different values compared to input strides/distance.

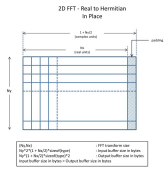

Fig. 11: **2D FFT - Real to Hermitian In Place**

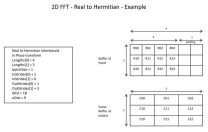

Fig. 12: **2D FFT - Real to Hermitian, Example**

CHAPTER

TWO

API USAGE

This section describes usage of the rocFFT library API.

2.1 Types

There are a few data structures that are internal to the library. The pointer types to these structures are given below. The user would need to use these types to create handles and pass them between different library functions.

typedef struct rocfft_plan_t ***rocfft_plan**
Pointer type to plan structure.

This type is used to declare a plan handle that can be initialized with *rocfft_plan_create*.

typedef struct rocfft_plan_description_t ***rocfft_plan_description**
Pointer type to plan description structure.

This type is used to declare a plan description handle that can be initialized with *rocfft_plan_description_create*.

typedef struct rocfft_execution_info_t ***rocfft_execution_info**
Pointer type to execution info structure.

This type is used to declare an execution info handle that can be initialized with *rocfft_execution_info_create*.

2.2 Library Setup and Cleanup

The following functions deal with initialization and cleanup of the library.

rocfft_status **rocfft_setup**()
Library setup function, called once in program before start of library use.

rocfft_status **rocfft_cleanup**()
Library cleanup function, called once in program after end of library use.

2.3 Plan

The following functions are used to create and destroy plan objects.

rocfft_status **rocfft_plan_create**(*rocfft_plan* *plan, *rocfft_result_placement* placement, *rocfft_transform_type* transform_type, *rocfft_precision* precision, size_t dimensions, const size_t *lengths, size_t number_of_transforms, const *rocfft_plan_description* description)

Create an FFT plan.

This API creates a plan, which the user can execute subsequently. This function takes many of the fundamental parameters needed to specify a transform.

The dimensions parameter can take a value of 1, 2, or 3. The 'lengths' array specifies the size of data in each dimension. Note that lengths[0] is the size of the innermost dimension, lengths[1] is the next higher dimension and so on (column-major ordering).

The 'number_of_transforms' parameter specifies how many transforms (of the same kind) needs to be computed. By specifying a value greater than 1, a batch of transforms can be computed with a single API call.

Additionally, a handle to a plan description can be passed for more detailed transforms. For simple transforms, this parameter can be set to NULL.

The plan must be destroyed with a call to *rocfft_plan_destroy*.

Parameters

- **plan** – **[out]** plan handle
- **placement** – **[in]** placement of result
- **transform_type** – **[in]** type of transform
- **precision** – **[in]** precision
- **dimensions** – **[in]** dimensions
- **lengths** – **[in]** dimensions-sized array of transform lengths
- **number_of_transforms** – **[in]** number of transforms
- **description** – **[in]** description handle created by rocfft_plan_description_create; can be NULL for simple transforms

rocfft_status **rocfft_plan_destroy**(*rocfft_plan* plan)

Destroy an FFT plan.

This API frees the plan after it is no longer needed.

Parameters **plan** – **[in]** plan handle

The following functions are used to query for information after a plan is created.

rocfft_status **rocfft_plan_get_work_buffer_size**(const *rocfft_plan* plan, size_t *size_in_bytes)

Get work buffer size.

Get the work buffer size required for a plan.

Parameters

- **plan** – **[in]** plan handle
- **size_in_bytes** – **[out]** size of needed work buffer in bytes

rocfft_status **rocfft_plan_get_print**(const *rocfft_plan* plan)
Print all plan information.

Prints plan details to stdout, to aid debugging

Parameters **plan** – **[in]** plan handle

2.4 Plan description

Most of the time, *rocfft_plan_create()* is able to fully specify a transform. Advanced plan details such as strides and offsets require creation of a plan description object, which is configured and passed to the *rocfft_plan_create()* function.

The plan description object can be safely destroyed after it is given to the *rocfft_plan_create()* function.

rocfft_status **rocfft_plan_description_create**(*rocfft_plan_description* *description)
Create plan description.

This API creates a plan description with which the user can set extra plan properties. The plan description must be freed with a call to *rocfft_plan_description_destroy*.

Parameters **description** – **[out]** plan description handle

rocfft_status **rocfft_plan_description_destroy**(*rocfft_plan_description* description)
Destroy a plan description.

This API frees the plan description. A plan description can be freed any time after it is passed to *rocfft_plan_create*.

Parameters **description** – **[in]** plan description handle

rocfft_status **rocfft_plan_description_set_data_layout**(*rocfft_plan_description* description, const *rocfft_array_type* in_array_type, const *rocfft_array_type* out_array_type, const size_t *in_offsets, const size_t *out_offsets, const size_t in_strides_size, const size_t *in_strides, const size_t in_distance, const size_t out_strides_size, const size_t *out_strides, const size_t out_distance)
Set advanced data layout parameters on a plan description.

This API specifies advanced layout of input/output buffers for a plan description.

The following parameters are supported for inputs and outputs:

- Array type (real, hermitian, or complex data, in either interleaved or planar format).
  - Real forward transforms require real input and hermitian output.
  - Real inverse transforms require hermitian input and real output.
  - Complex transforms require complex input and output.
  - Hermitian and complex data defaults to interleaved if a specific format is not specified.
- Offset of first data element in the data buffer. Defaults to 0 if unspecified.
- Stride between consecutive elements in each dimension. Defaults to contiguous data in all dimensions if unspecified.
- Distance between consecutive batches. Defaults to contiguous batches if unspecified.

Not all combinations of array types are supported and error codes will be returned for unsupported cases.

Parameters

- **description** – **[inout]** description handle
- **in_array_type** – **[in]** array type of input buffer
- **out_array_type** – **[in]** array type of output buffer
- **in_offsets** – **[in]** offsets, in element units, to start of data in input buffer
- **out_offsets** – **[in]** offsets, in element units, to start of data in output buffer
- **in_strides_size** – **[in]** size of in_strides array (must be equal to transform dimensions)
- **in_strides** – **[in]** array of strides, in each dimension, of input buffer; if set to null ptr library chooses defaults
- **in_distance** – **[in]** distance between start of each data instance in input buffer
- **out_strides_size** – **[in]** size of out_strides array (must be equal to transform dimensions)
- **out_strides** – **[in]** array of strides, in each dimension, of output buffer; if set to null ptr library chooses defaults
- **out_distance** – **[in]** distance between start of each data instance in output buffer

2.5 Execution

After a plan has been created, it can be executed using the *rocfft_execute()* function, to compute a transform on specified data. Aspects of the execution can be controlled and any useful information returned to the user.

rocfft_status **rocfft_execute**(const *rocfft_plan* plan, void *in_buffer[], void *out_buffer[], *rocfft_execution_info* info)

Execute an FFT plan.

This API executes an FFT plan on buffers given by the user.

If the transform is in-place, only the input buffer is needed and the output buffer parameter can be set to NULL. For not in-place transforms, output buffers have to be specified.

Input and output buffer are arrays of pointers. Interleaved array formats are the default, and require just one pointer per input or output buffer. Planar array formats require two pointers per input or output buffer - real and imaginary pointers, in that order.

Note that input buffers may still be overwritten during execution of a transform, even if the transform is not in-place.

The final parameter in this function is a rocfft_execution_info handle. This optional parameter serves as a way for the user to control execution streams and work buffers.

Parameters

- **plan** – **[in]** plan handle
- **in_buffer** – **[inout]** array (of size 1 for interleaved data, of size 2 for planar data) of input buffers
- **out_buffer** – **[inout]** array (of size 1 for interleaved data, of size 2 for planar data) of output buffers, ignored for in-place transforms
- **info** – **[in]** execution info handle created by rocfft_execution_info_create

2.6 Execution info

rocfft_execute() takes an optional *rocfft_execution_info* parameter. This parameter encapsulates information such as the work buffer and compute stream for the transform.

rocfft_status **rocfft_execution_info_create**(*rocfft_execution_info* *info)
Create execution info.

This API creates an execution info with which the user can control plan execution and work buffers. The execution info must be freed with a call to *rocfft_execution_info_destroy*.

Parameters **info** – **[out]** execution info handle

rocfft_status **rocfft_execution_info_destroy**(*rocfft_execution_info* info)
Destroy an execution info.

This API frees the execution info. An execution info object can be freed any time after it is passed to *rocfft_execute*.

Parameters **info** – **[in]** execution info handle

rocfft_status **rocfft_execution_info_set_work_buffer**(*rocfft_execution_info* info, void *work_buffer, const size_t size_in_bytes)
Set work buffer in execution info.

This is one of the execution info functions to specify optional additional information to control execution. This API provides a work buffer for the transform. It must be called before *rocfft_execute*.

When a non-zero value is obtained from *rocfft_plan_get_work_buffer_size*, that means the library needs a work buffer to compute the transform. In this case, the user should allocate the work buffer and pass it to the library via this API.

If a work buffer is required for the transform but is not specified using this function, *rocfft_execute* will automatically allocate the required buffer and free it when execution is finished.

Users should allocate their own work buffers if they need precise control over the lifetimes of those buffers, or if multiple plans need to share the same buffer.

Parameters

- **info** – **[in]** execution info handle
- **work_buffer** – **[in]** work buffer
- **size_in_bytes** – **[in]** size of work buffer in bytes

rocfft_status **rocfft_execution_info_set_stream**(*rocfft_execution_info* info, void *stream)
Set stream in execution info.

Associates an existing compute stream to a plan. This must be called before the call to *rocfft_execute*.

Once the association is made, execution of the FFT will run the computation through the specified stream.

The stream must be of type hipStream_t. It is an error to pass the address of a hipStream_t object.

Parameters

- **info** – **[in]** execution info handle
- **stream** – **[in]** underlying compute stream

2.7 Enumerations

This section provides all the enumerations used.

enum **rocfft_status**
rocFFT status/error codes

Values:

enumerator **rocfft_status_success**

enumerator **rocfft_status_failure**

enumerator **rocfft_status_invalid_arg_value**

enumerator **rocfft_status_invalid_dimensions**

enumerator **rocfft_status_invalid_array_type**

enumerator **rocfft_status_invalid_strides**

enumerator **rocfft_status_invalid_distance**

enumerator **rocfft_status_invalid_offset**

enumerator **rocfft_status_invalid_work_buffer**

enum **rocfft_transform_type**
Type of transform.

Values:

enumerator **rocfft_transform_type_complex_forward**

enumerator **rocfft_transform_type_complex_inverse**

enumerator **rocfft_transform_type_real_forward**

enumerator **rocfft_transform_type_real_inverse**

enum **rocfft_precision**
Precision.

Values:

enumerator **rocfft_precision_single**

enumerator **rocfft_precision_double**

enum **rocfft_result_placement**
Result placement.

Declares where the output of the transform should be placed. Note that input buffers may still be overwritten during execution of a transform, even if the transform is not in-place.

Values:

enumerator **rocfft_placement_inplace**

enumerator **rocfft_placement_notinplace**

enum **rocfft_array_type**

Array type.

Values:

enumerator **rocfft_array_type_complex_interleaved**

enumerator **rocfft_array_type_complex_planar**

enumerator **rocfft_array_type_real**

enumerator **rocfft_array_type_hermitian_interleaved**

enumerator **rocfft_array_type_hermitian_planar**

enumerator **rocfft_array_type_unset**

CHAPTER

THREE

API REFERENCE

file **rocfft.h**

#include <cstddef> *rocfft.h* defines all the public interfaces and types

Defines

ROCFFT_EXPORT

Typedefs

typedef struct rocfft_plan_t ***rocfft_plan**

Pointer type to plan structure.

This type is used to declare a plan handle that can be initialized with *rocfft_plan_create*.

typedef struct rocfft_plan_description_t ***rocfft_plan_description**

Pointer type to plan description structure.

This type is used to declare a plan description handle that can be initialized with *rocfft_plan_description_create*.

typedef struct rocfft_execution_info_t ***rocfft_execution_info**

Pointer type to execution info structure.

This type is used to declare an execution info handle that can be initialized with *rocfft_execution_info_create*.

Enums

enum **rocfft_status**

rocFFT status/error codes

Values:

enumerator **rocfft_status_success**

enumerator **rocfft_status_failure**

enumerator **rocfft_status_invalid_arg_value**

enumerator **rocfft_status_invalid_dimensions**

enumerator **rocfft_status_invalid_array_type**

enumerator **rocfft_status_invalid_strides**

enumerator **rocfft_status_invalid_distance**

enumerator **rocfft_status_invalid_offset**

enumerator **rocfft_status_invalid_work_buffer**

enum **rocfft_transform_type**
Type of transform.

Values:

enumerator **rocfft_transform_type_complex_forward**

enumerator **rocfft_transform_type_complex_inverse**

enumerator **rocfft_transform_type_real_forward**

enumerator **rocfft_transform_type_real_inverse**

enum **rocfft_precision**
Precision.

Values:

enumerator **rocfft_precision_single**

enumerator **rocfft_precision_double**

enum **rocfft_result_placement**
Result placement.

Declares where the output of the transform should be placed. Note that input buffers may still be overwritten during execution of a transform, even if the transform is not in-place.

Values:

enumerator **rocfft_placement_inplace**

enumerator **rocfft_placement_notinplace**

enum **rocfft_array_type**
Array type.

Values:

enumerator **rocfft_array_type_complex_interleaved**

enumerator **rocfft_array_type_complex_planar**

enumerator **rocfft_array_type_real**

enumerator **rocfft_array_type_hermitian_interleaved**

enumerator **rocfft_array_type_hermitian_planar**

enumerator **rocfft_array_type_unset**

Functions

rocfft_status **rocfft_setup**()
Library setup function, called once in program before start of library use.

rocfft_status **rocfft_cleanup**()
Library cleanup function, called once in program after end of library use.

rocfft_status **rocfft_plan_create**(*rocfft_plan* *plan, *rocfft_result_placement* placement, *rocfft_transform_type* transform_type, *rocfft_precision* precision, size_t dimensions, const size_t *lengths, size_t number_of_transforms, const *rocfft_plan_description* description)

Create an FFT plan.

This API creates a plan, which the user can execute subsequently. This function takes many of the fundamental parameters needed to specify a transform.

The dimensions parameter can take a value of 1, 2, or 3. The 'lengths' array specifies the size of data in each dimension. Note that lengths[0] is the size of the innermost dimension, lengths[1] is the next higher dimension and so on (column-major ordering).

The 'number_of_transforms' parameter specifies how many transforms (of the same kind) needs to be computed. By specifying a value greater than 1, a batch of transforms can be computed with a single API call.

Additionally, a handle to a plan description can be passed for more detailed transforms. For simple transforms, this parameter can be set to NULL.

The plan must be destroyed with a call to *rocfft_plan_destroy*.

Parameters

- **plan** – **[out]** plan handle
- **placement** – **[in]** placement of result
- **transform_type** – **[in]** type of transform
- **precision** – **[in]** precision
- **dimensions** – **[in]** dimensions
- **lengths** – **[in]** dimensions-sized array of transform lengths
- **number_of_transforms** – **[in]** number of transforms
- **description** – **[in]** description handle created by rocfft_plan_description_create; can be NULL for simple transforms

rocfft_status **rocfft_execute**(const *rocfft_plan* plan, void *in_buffer[], void *out_buffer[], *rocfft_execution_info* info)

Execute an FFT plan.

This API executes an FFT plan on buffers given by the user.

If the transform is in-place, only the input buffer is needed and the output buffer parameter can be set to NULL. For not in-place transforms, output buffers have to be specified.

Input and output buffer are arrays of pointers. Interleaved array formats are the default, and require just one pointer per input or output buffer. Planar array formats require two pointers per input or output buffer - real and imaginary pointers, in that order.

Note that input buffers may still be overwritten during execution of a transform, even if the transform is not in-place.

The final parameter in this function is a rocfft_execution_info handle. This optional parameter serves as a way for the user to control execution streams and work buffers.

Parameters

- **plan** – **[in]** plan handle
- **in_buffer** – **[inout]** array (of size 1 for interleaved data, of size 2 for planar data) of input buffers
- **out_buffer** – **[inout]** array (of size 1 for interleaved data, of size 2 for planar data) of output buffers, ignored for in-place transforms
- **info** – **[in]** execution info handle created by rocfft_execution_info_create

rocfft_status **rocfft_plan_destroy**(*rocfft_plan* plan)

Destroy an FFT plan.

This API frees the plan after it is no longer needed.

Parameters **plan** – **[in]** plan handle

rocfft_status **rocfft_plan_description_set_data_layout**(*rocfft_plan_description* description, const *rocfft_array_type* in_array_type, const *rocfft_array_type* out_array_type, const size_t *in_offsets, const size_t *out_offsets, const size_t in_strides_size, const size_t *in_strides, const size_t in_distance, const size_t out_strides_size, const size_t *out_strides, const size_t out_distance)

Set advanced data layout parameters on a plan description.

This API specifies advanced layout of input/output buffers for a plan description.

The following parameters are supported for inputs and outputs:

- Array type (real, hermitian, or complex data, in either interleaved or planar format).
  - Real forward transforms require real input and hermitian output.
  - Real inverse transforms require hermitian input and real output.
  - Complex transforms require complex input and output.
  - Hermitian and complex data defaults to interleaved if a specific format is not specified.
- Offset of first data element in the data buffer. Defaults to 0 if unspecified.
- Stride between consecutive elements in each dimension. Defaults to contiguous data in all dimensions if unspecified.
- Distance between consecutive batches. Defaults to contiguous batches if unspecified.

Not all combinations of array types are supported and error codes will be returned for unsupported cases.

Parameters

- **description** – **[inout]** description handle
- **in_array_type** – **[in]** array type of input buffer
- **out_array_type** – **[in]** array type of output buffer
- **in_offsets** – **[in]** offsets, in element units, to start of data in input buffer

- **out_offsets** – **[in]** offsets, in element units, to start of data in output buffer
- **in_strides_size** – **[in]** size of in_strides array (must be equal to transform dimensions)
- **in_strides** – **[in]** array of strides, in each dimension, of input buffer; if set to null ptr library chooses defaults
- **in_distance** – **[in]** distance between start of each data instance in input buffer
- **out_strides_size** – **[in]** size of out_strides array (must be equal to transform dimensions)
- **out_strides** – **[in]** array of strides, in each dimension, of output buffer; if set to null ptr library chooses defaults
- **out_distance** – **[in]** distance between start of each data instance in output buffer

rocfft_status **rocfft_get_version_string**(char *buf, size_t len)
Get library version string.

Parameters

- **buf** – **[inout]** buffer that receives the version string
- **len** – **[in]** length of buf, minimum 30 characters

rocfft_status **rocfft_plan_get_work_buffer_size**(const *rocfft_plan* plan, size_t *size_in_bytes)
Get work buffer size.

Get the work buffer size required for a plan.

Parameters

- **plan** – **[in]** plan handle
- **size_in_bytes** – **[out]** size of needed work buffer in bytes

rocfft_status **rocfft_plan_get_print**(const *rocfft_plan* plan)
Print all plan information.

Prints plan details to stdout, to aid debugging

Parameters **plan** – **[in]** plan handle

rocfft_status **rocfft_plan_description_create**(*rocfft_plan_description* *description)
Create plan description.

This API creates a plan description with which the user can set extra plan properties. The plan description must be freed with a call to *rocfft_plan_description_destroy*.

Parameters **description** – **[out]** plan description handle

rocfft_status **rocfft_plan_description_destroy**(*rocfft_plan_description* description)
Destroy a plan description.

This API frees the plan description. A plan description can be freed any time after it is passed to *rocfft_plan_create*.

Parameters **description** – **[in]** plan description handle

rocfft_status **rocfft_execution_info_create**(*rocfft_execution_info* *info)
Create execution info.

This API creates an execution info with which the user can control plan execution and work buffers. The execution info must be freed with a call to *rocfft_execution_info_destroy*.

Parameters **info** – **[out]** execution info handle

rocfft_status **rocfft_execution_info_destroy**(*rocfft_execution_info* info)
Destroy an execution info.

This API frees the execution info. An execution info object can be freed any time after it is passed to *rocfft_execute*.

Parameters **info** – **[in]** execution info handle

rocfft_status **rocfft_execution_info_set_work_buffer**(*rocfft_execution_info* info, void *work_buffer, const size_t size_in_bytes)
Set work buffer in execution info.

This is one of the execution info functions to specify optional additional information to control execution. This API provides a work buffer for the transform. It must be called before *rocfft_execute*.

When a non-zero value is obtained from *rocfft_plan_get_work_buffer_size*, that means the library needs a work buffer to compute the transform. In this case, the user should allocate the work buffer and pass it to the library via this API.

If a work buffer is required for the transform but is not specified using this function, *rocfft_execute* will automatically allocate the required buffer and free it when execution is finished.

Users should allocate their own work buffers if they need precise control over the lifetimes of those buffers, or if multiple plans need to share the same buffer.

Parameters

- **info** – **[in]** execution info handle
- **work_buffer** – **[in]** work buffer
- **size_in_bytes** – **[in]** size of work buffer in bytes

rocfft_status **rocfft_execution_info_set_stream**(*rocfft_execution_info* info, void *stream)
Set stream in execution info.

Associates an existing compute stream to a plan. This must be called before the call to *rocfft_execute*.

Once the association is made, execution of the FFT will run the computation through the specified stream.

The stream must be of type hipStream_t. It is an error to pass the address of a hipStream_t object.

Parameters

- **info** – **[in]** execution info handle
- **stream** – **[in]** underlying compute stream

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- search

INDEX

R