AMD Math Libraries

AMD Accelerated Parallel Processing TECHNOLOGY

OpenCL Fast Fourier Transforms (FFTs)

clAmdFft

April 2012

Revision 1.8

© 2012 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, AMD Accelerated Parallel Processing, the AMD Accelerated Parallel Processing logo, ATI, the ATI logo, Radeon, FireStream, FirePro, Catalyst, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Microsoft, Visual Studio, Windows, and Windows Vista are registered trademarks of Microsoft Corporation in the U.S. and/or other jurisdictions. Other names are for informational purposes only and may be trademarks of their respective owners. OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission by Khronos.

The contents of this document are provided in connection with Advanced Micro Devices, Inc. ("AMD") products. AMD makes no representations or warranties with respect to the accuracy or completeness of the contents of this publication and reserves the right to make changes to specifications and product descriptions at any time without notice. The information contained herein may be of a preliminary or advance nature and is subject to change without notice. No license, whether express, implied, arising by estoppel or otherwise, to any intellectual property rights is granted by this publication. Except as set forth in AMD's Standard Terms and Conditions of Sale, AMD assumes no liability whatsoever, and disclaims any express or implied warranty, relating to its products including, but not limited to, the implied warranty of merchantability, fitness for a particular purpose, or infringement of any intellectual property right.

AMD's products are not designed, intended, authorized or warranted for use as components in systems intended for surgical implant into the body, or in other applications intended to support or sustain life, or in any other application in which the failure of AMD's product could create a situation where personal injury, death, or severe property or environmental damage may occur. AMD reserves the right to discontinue or make changes to its products at any time without notice.

Advanced Micro Devices, Inc.
One AMD Place
P.O. Box 3453
Sunnyvale, CA 94088-3453
www.amd.com

For AMD Accelerated Parallel Processing:

URL:	**developer.amd.com/appsdk**
Developing:	**developer.amd.com/**
Support:	**developer.amd.com/appsdksupport**
Forum:	**developer.amd.com/openclforum**

For AMD Math Library pages:

developer.amd.com/libraries/appmathlibs/Pages/default.aspx

Contents

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

Chapter 1 OpenCL Fast Fourier Transforms (FFTs)

The clAmdFft library is an OpenCL library implementation of discrete Fast Fourier Transforms. It:

- Provides a fast and accurate platform for calculating discrete FFTs.
- Works on CPU or GPU backends.
- Supports in-place or out-of-place transforms.
- Supports 1D, 2D, and 3D transforms with a batch size that can be greater than 1.
- Supports planar (real and complex components in separate arrays) and interleaved (real and complex components as a pair contiguous in memory) formats.
- Supports dimension lengths that can be any mix of powers of 2, 3, and 5.
- Supports single and double precision floating point formats.

1.1 Installation of clAmdFft library

1.1.1 Downloadable Binaries

AMD provides clAmdFft pre-compiled library packages for recent versions of Microsoft Windows operating systems and several flavors of Linux.

The downloadable binary packages are freely available from AMD at `http://developer.amd.com/gpu/appmathlibs/Pages/default.aspx`.

Once the appropriate package for the respective OS has finished downloading, uncompress the package using the native tools available on the platform in a directory of the user's choice. Everything needed to build a program using clAmdFft is included in the directory tree, including documentation, header files, binary library components, and sample programs for programming illustration.

1.1.1.1 CMake

After the clAmdFft package is uncompressed on the user's hard drive, a samples directory exists with source code, but no Visual Studio project files, Unix makefiles, or other native build system exist. Instead, it contains a `CMakeLists.txt` file. clAmdFft uses CMake as its build system, and other build files, such as Visual Studio projects, nmake makefiles, or Unix makefiles, are

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

generated by the CMake build system, during configuration. CMake is freely available for download from: `http://www.cmake.org/`

NOTE: CMake generates the native OS build files, so any changes made to the native build files are overwritten the next time CMake is run.

CMake is written to pull compiler information from environment variables, and to look in default install directories for tools. Once installed, a popular interface to control the process of creating native build files is CMake-gui. When the GUI is launched, two text boxes appear at the top of the dialog: a path to source and a separate path to generate binaries. For the `browse source...` box, find the path to where you unzipped clAmdFft, and select the root `samples` directory that contains the `CMakeLists.txt`; for clAmdFft, this should be `clAmdFft/samples`. For `browse build...`, select an appropriate directory where the build environment generates build files; a convenient location is a sibling directory to the source. This makes it easy to wipe all the binaries and start a fresh build. For instance, for a debug configuration of NMake, an example directory could be `clAmdFft/bin/NMakeDebug`. This is where the generated makefile, native build files, and intermediate object files are built. These generated files are kept separate from the source; this is referred to as 'out-of-source' builds, and is very similar in concept to what 'autotools' does for Linux. To build using NMake, simply type `NMake` in the build directory containing the makefile. To build using Visual Studio, generate the solution and project files into a directory such as `clAmdFft/bin/vs10`, find the generated `.sln` file, and open the solution.

The first time the `configure` button near the bottom of the screen is clicked, it causes CMake to prompt for what type of native build files to make. Various properties appear in red in the `properties` box. Red indicates that the value has changed since last time `configure` was clicked. (The first time configure is clicked, everything is red.) CMake tries to configure itself automatically to the client's system by looking at a systems environment variables and by searching through default install locations for project dependencies. Take a moment to verify the settings and paths that are displayed on the configuration screen; if any changes must be made, you can provide correct paths or adjust settings by typing directly into the CMake configuration screen. Click the `configure` button a second time to 'bake' those settings and serialize them to disk.

Options relevant to the clAmdFft project include:

- `AMDAPPSDKROOT` — Location of the Stream SDK installation. This value is already populated if CMake could determine the location by looking at the environment variables. If not, the user must provide a path to the root installation of the Stream SDK here.
- `BOOST_ROOT` — Location of the Boost SDK installation. This value is already populated if CMake could determine the location by looking at the environment variables or default install locations. If not, the user must provide a path to the root installation of the Stream SDK here. This dependency is only relevant to the sample client; the FFT library does not depend on Boost.

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

- `CMAKE_BUILD_TYPE` — Defines the build type (default is debug). For Visual Studio projects, this does not appear (modifiable in IDE); for makefile-based builds, this is set in CMake.
- `CMAKE_INSTALL_PREFIX` — The path to install all binaries and headers generated from the build. This is used when the user types `make install` or builds the INSTALL project in Visual Studio. All generated binaries and headers are copied into the path prefixed with `CMAKE_INSTALL_PREFIX`.

  The Visual Studio projects are self explanatory, but a few other projects are autogenerated; these might be unfamiliar.
- `ALL_BUILD` — A project that is empty of files, but since it depends on all user projects, it provides a convenient way to rebuild everything.
- `ZERO_CHECK` — A CMake-specific project that checks to see if the generated solution and project files are in sync with the `CMakeLists.txt` file. If these files are modified, the solutions and projects are now out-of-sync, and this project prompts the user to regenerate their environment.

Note: If the user chooses to build on Windows with a NMake based build, it is important to launch CMake from within a Visual Studio Command Prompt (20xx). This is because CMake must be able to parse environment variables to properly initialize NMake. This is not necessary if a Visual Studio solution is generated, because solution files contain their own environmental setup.

1.1.1.2 Boost

clAmdFft includes one sample project that has source dependencies on Boost: the sample client project. Boost is freely available from: `http://www.boost.org/`.

The command-line clFFT sample client links with the `program_options` library, which provides functionality for parsing command-line parameters and `.ini` files in a cross-platform manner. Once Boost is downloaded and extracted on the hard drive, the `program_options` library must be compiled. The Boost build system uses the BJam builder (a project for a CMake-based Boost build is available for separate download). This is available for download from the Boost website, or the user can build BJam; Boost includes the source to BJam in its distribution, and the user can execute `bootstrap.bat` (located in the root `boost` directory) to build it.

After BJam is either built or installed, an example BJam command-line is given below for building a 64-bit `program_options` binary, for both static and dynamic linking:

```
bjam --with-program_options address-model=64 link=static,shared stage
```

The last step to make boost readily available and usable by CMake and the native compiler is to add an environment variable to the system called BOOST_ROOT. In Windows, right click on the computer icon and go to

```
Properties|Advanced system settings|Advanced|Environment Variables...
```

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

Remember to relaunch any new processes that are open, in order to inherit the new environment variable. On Linux, consider modifying the `.bash_rc` file (or shell equivalent) to export a new environment variable every time you log in.

If you are on a Linux system and have used a package manager to install Boost, you may have to confirm where the Boost `include` and `library` files have been placed. For example, after installing Boost with the Ubuntu Synaptic Package Manager, the Boost `include` files are in `/usr/include/boost`, and the `library` files either `/usr/lib` or `/usr/lib64`. The `CMakeLists.txt` file in this project defaults the `BOOST_ROOT` value to `/usr` on Linux; so, if the system is set up similarly, no further action is necessary. If the system is set up differently, you may have to set the `BOOST_ROOT` environmental variable accordingly.

Note that CMake does not recognize version numbers at the end of the library filename; so, if the package manager only created a `libboost_module_name.so.x.xx.x` file (where `x.xx.x` is the version of Boost), the user may need to manually create a soft link called `libboost_module_name.so` to the versioned `libboost_module_name.so.x.xx.x`. See the clAmdFft binary artifacts in the install directory for an example.

1.2 Introduction to clAmdFft

The FFT is an implementation of the Discrete Fourier Transform (DFT) that makes use of symmetries in the FFT definition to reduce the mathematical intensity required from O(N^2) to O($N \log N$) when the sequence length, `N`, is the product of small prime factors. Currently, there is no standard API for FFT routines. Hardware vendors usually provide a set of high-performance FFTs optimized for their systems: no two vendors employ the same interfaces for their FFT routines. clAmdFft provides a set of FFT routines that are optimized for AMD graphics processors, but also are functional across CPU and other compute devices.

1.2.1 Supported Transform Sizes

clAmdFft supports powers of 2, 3 and 5 sizes. This means that the vector lengths that can be configured through a plan can be any length that is a power of two, three, and five; examples include 2^0, 2^1*3^1, 3^3*5^5, $2^2*3^3*5^5$; up to the limit that the device can support.

1.2.2 Transform Size Limits

Currently, there is an upper bound on the transform size the library supports. This limit is 2^{24} for single precision and 2^{22} for double precision. This means that the product of transform lengths must not exceed these values. As an example, a 1D single-precision FFT of size 1024 is valid since $1024 \leq 2^{24}$. Similarly, a 2D double-precision FFT of size 1024x1024 is also valid, since $1024*1024 \leq 2^{22}$. But, a 2D single-precision FFT of size 4096x8192 is not valid because $4096*8192 > 2^{24}$.

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

1.2.3 Dimensionality

clAmdFft currently supports FFTs of up to three dimensions, given by the enum `clAmdFft-Dim`. This enum is a required parameter into `clAmdFftCreateDefaultPlan()` to create an initial plan; there is no default for this parameter. Depending on the dimensionality that the client requests, clAmdFft uses the formulations shown below to compute the DFT.

The definition of a 1D complex DFT used by clAmdFft is given by:

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

where x_k are the complex data to be transformed, $\tilde{x}_j$ are the transformed data, and the sign of ± determines the direction of the transform: – for forward, and + for backward. Note that the user must provided the scaling factor. Typically, the scale is set to 1 for forward transforms, and $\frac{1}{N}$ for backward transforms.

The definition of a complex 2D DFT used by clAmdFft is given by:

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

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 of ± determines the direction of the transform. Typically, the scale is set to 1 for forward transforms, and $\frac{1}{M \cdot N}$ for backward transforms.

The definition of a complex 3D DFT used by clAmdFft is given by:

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

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 of ± determines the direction of the transform. Typically, the scale is set to 1 for forward transforms, and $\frac{1}{M \cdot N \cdot P}$ for backward transforms.

1.2.4 Setup and Teardown of clAmdFft

clAmdFft is initialized by a call to `clAmdFftSetup()`, which must be called before any other API exported from clAmdFft. This allows the library to create resources used to manage the plans that are created and destroyed by the user. This API also takes a structure `clAmdFftInitSetupData` that is initialized by the client to control the behavior of the library. The corresponding `clAmdFftTeardown()` method must be called by the client when it is done using the library. This instructs clAmdFft to release all resources, including any acquired references to any OpenCL objects that may have been allocated or passed to it through the API.

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

1.2.5 Thread Safety

The clAmdFft API is designed to be thread-safe. It is safe to create plans from multiple threads, and to destroy those plans in separate threads. Multiple threads can call `clAmdFftEnqueueTransform()` to place work into a command queue at the same time. clAmdFft does not provide a single-threaded version of the library. It is expected that the overhead of the synchronization mechanisms inside of clAmdFft thread safe is minor.

Currently, multi-device operation must be managed by the user. OpenCL contexts can be created that are associated with multiple devices, but clAmdFft only uses a single device from that context to transform the data. Multi-device operation can be managed by the user by creating multiple contexts, where each context contains a different device, and the user is responsible for scheduling and partitioning the work across multiple devices and contexts.

1.2.6 Row-Major Formats

clAmdFft expects all multi-dimensional input passed to it to be in row-major format. This is compatible with C-based languages. However, clAmdFft is very flexible in the input and output data organization it accepts by allowing the user to specify a stride for each dimension. This feature can be used to process data in column major arrays, and other non-contiguous data formats. See `clAmdFftSetPlanInStride` and `clAmdFftSetPlanOutStride`.

1.2.7 OpenCL Object Creation

OpenCL objects, such as contexts, `cl_mem` buffers, and command queues, are the responsibility of the user application to allocate and manage. All of the clAmdFft interfaces that must interact with OpenCL objects take those objects as references through the API. Specifically, the plan creation function `clAmdFftCreateDefaultPlan()` takes an OpenCL context as a parameter reference, increments the reference count on that object, and keeps the object alive until the corresponding plan has been destroyed through a call to `clAmdFftDestroyPlan()`.

1.2.8 Flushing Command Queues

The clAmdFft API operates asynchronously, and with the exception of thread safety, all APIs return immediately. Specifically, the `clAmdFftEnqueueTransform()` API does not explicitly flush the command queues passed by reference to it; it pushes the transform work into the command queue and returns the modified queue to the client. The client is free to issue its own blocking logic, using OpenCL synchronization mechanisms, or push further work onto the queue to continue processing.

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

1.3 clAmdFft Plans

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

- What OpenCL context executes the transform?
- Is this a 1D, 2D or 3D transform?
- What are the lengths or extents of the data in each dimension?
- How many datasets are being transformed?
- What is the data precision?
- Should a scaling factor be applied to the transformed data?
- Does the output transformed data replace the original input data in the same buffer (or buffers), or is the output data written to a different buffer (or buffers).
- How is the input data stored in its data buffers?
- How is the output data stored in its data buffers?

The plan does not include:

- The OpenCL handles to the input and output data buffers.
- The OpenCL handle to a temporary scratch buffer (if needed).
- Whether to execute a forward or reverse transform.

These are specified when the plan is executed.

1.3.1 Default Plan Values

When a new plan is created by calling `clAmdFftCreateDefaultPlan`, its parameters are initialized as follows:

- Dimensions: as provided by the caller.
- Lengths: as provided by the caller.
- Batch size:1.
- Precision: `CLFFT_SINGLE`.
- Scaling factors:
  a. For the forward transform, the default is 1.0, or no scale factor is applied.
  b. For the reverse transform, the default is 1.0 / P, where P is the product of the FFT lengths.
- Location: `CLFFT_INPLACE`.
- Input layout: `CLFFT_COMPLEX_INTERLEAVED`.
- Input strides: the strides of a multidimensional array of the lengths specified, where the data is compactly stored using the row-major convention.
- Output layout: `CLFFT_COMPLEX_INTERLEAVED`.

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

- Output strides: same as input strides.

Writing client programs that depend on these initial values is **not** recommended.

1.3.2 Supported Memory Layouts

There are two main families of Discrete Fourier Transform (DFT):

- Routines for the transformation of complex data. clAmdFft supports two layouts to store complex numbers: a 'planar' format, where the real and imaginary components are kept in separate arrays:

  Buffer1: `RRRRR`

  Buffer2: `IIIII`

  and an interleaved format, where the real and imaginary components are stored as contiguous pairs:

  Buffer1: `RIRIRIRIRIRI`
- Routines for the transformation of real to complex data and vice versa; clAmdFft provides enums to define these formats. For transforms involving real data, there are two possibilities:
  - Real data being subject to forward FFT transform that results in complex data.
  - Complex data being subject to backward FFT transform that results in real data. See the Section 1.5, “FFTs of Real Data,” page 1-11.

1.3.2.1 Strides and Distances

For one-dimensional data, if clStrides[0]=strideX=1, successive elements in the first dimension are stored contiguously in memory. If strideX is an integral value greater than 1, gaps in memory exist between each element of the vectors.

For multi-dimensional data, if clStrides[1]=strideY = LenX for 2 dimensional data and clStrides[2]=strideZ = LenX*LenY for 3 dimensional 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 clLengths[0], clLengths[1], and clLengths[2], respectively, which are used to set up the plan.

By specifying non-default strides, it is possible to process either row-major or column-major arrays. Data can be extracted from arrays of structures. Almost any regular data storage pattern can be accommodated.

Distance is the amount of memory that exists between corresponding elements in an FFT primitive in a batch. Distance is measured in the units of the FFT primitive; complex data measures in complex units, and real data measures in real data. Stride between tightly packed elements is 1 in either case. Typically, one can measure the distance between any two elements in a batch primitive, be it 1D, 2D, or 3D data. 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

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

possible to set the distance of a plan to be less than the size of the FFT vector; most often 1 for this case. When computing a batch of 1D FFT vectors, if distance == 1, and strideX == length(vector), a transposed output is produced for a batch of 1D vectors. It is left to the user to verify that the distance and strides are valid (not intersecting); if not valid, undefined results can occur.

A simple example is 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 had an array of 4096 rows x 1024 columns stored in a row-major manner, and you wanted to perform a 1-D length 4096 transform on each column.) In this case, specify the strides [1024, 1].

For a more complex example, an input buffer contained a raster grid of 1024 x 1024 monochrome pixel values, and you want to compute a 2D FFT for each 64 x 64 subtile of the grid. Specifying strides allows you to treat each horizontal band of 1024 x 64 pixels as an array of 16 64 x 64 matrixes, and process an entire band with a single call to `clAmdFftEnqueueTransform`. (Specifying strides is not quite flexible enough to transform the entire grid of this example with a single kernel execution.) It is possible to create a Plan to compute arrays of 64 x 64 2D FFTs, then specify three strides: [1, 1024, 64]. The first stride, 1, indicates that the rows of each matrix are stored consecutively; the second stride, 1024, gives the distance between rows, and the third stride, 64, defines the distance from matrix to matrix. Then call `clAmdFftEnqueueTransform` 16 times: once for each horizontal band of pixels.

1.3.3 Supported Precisions in clAmdFft

Both `CLFFT_SINGLE` and `CLFFT_DOUBLE` precisions are supported by the library for all supported radices. With both of these enums the host computer's math functions are used to produce tables of sines and cosines for use by the OpenCL kernel.

Both `CLFFT_SINGLE_FAST` and `CLFFT_DOUBLE_FAST` are meant to generate faster kernels with reduced accuracy, but are disabled in the current build..

See `clAmdFftPrecision`, `clAmdFftSetPlanPrecision`, and `clAmdFftGetPlanPrecision`.

1.3.4 clAmdFftDirection

The direction of the transform is not baked into the plan; the same plan can be used to specify both forward and backward transforms. Instead, `clAmdFftDirection` is passed as a parameter into `clAmdFftEnqueueTransform()`.

1.3.5 In-Place and Out-of-Place

The clAmdFft API supports both in-place and out-of-place transforms. With in-place transforms, only input buffers are provided to the `clAmdFftEnqueueTransform()` API, and the resulting data is written in the same

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

buffers, overwriting the input data. With out-of-place transforms, distinct output buffers are provided to the `clAmdFftEnqueue-Transform()` API, and the input data is preserved. In-place transforms require that the `cl_mem` objects the client creates have both `read` and `write` permissions. This is given in the nature of the in-place algorithm. Out-of-place transforms require that the destination buffers have `read` and `write` permissions, but input buffers can still be created with read-only permissions. This is a clAmdFft requirement because internally the FFT algorithms may go back and forth between the destination buffers and internally allocated temp buffers. For out-of-place transforms, clAmdFft never writes back to the input buffers.

1.3.6 Batches

The efficiency of clAmdFft is improved by utilizing transforms in batches. Sending as much data as possible in a single transform call leverages the parallel compute capabilities of OpenCL devices (and GPU devices in particular), and minimizes the penalty of transfer overhead. It's best to think of an OpenCL device as a high-throughput, high-latency device. Using a networking analogy as an example, it's 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. This can be done by batching. clAmdFft plans have a parameter to describe the number of transforms being batched: `clAmdFftSetPlanBatchSize()`, and to describe how those batches are laid out and spaced in memory: `clAmdFft-SetPlanDistance()`. 1D, 2D, or 3D transforms can be batched.

1.4 Using clAmdFft on a Client Application

To perform FFT calculations using clAmdFft, the client program must:

- Initialize the library by calling `clAmdFftSetup`.
- For each distinct type of FFT needed:
  a. Create an FFT Plan object. This usually is done by calling the factory function `clAmdFftCreateDefaultPlan`. Some of the most fundamental parameters are specified at this time, and others assume default values. The OpenCL context must be provided when the plan is created; it cannot be changed. Another way is to call `clAmdFftCopyPlan`. In either case, the function returns an opaque handle to the Plan object.
  b. Complete the specification of all of the Plan parameters by calling the various parameter-setting functions, `clAmdFFtSet`_____.
  c. Optionally, "bake" or finalize the plan, calling `clAmdFftBakePlan`. This signals to the library the end of the specification phase, and causes it to generate and compile the exact OpenCL kernels needed to perform the specified FFT on the OpenCL device provided.

     At this point, all performance-enhancing optimizations are applied, possibly including executing benchmark kernels on the OpenCL device context in order to maximize runtime performance.

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

Although this step is optional, most users probably want to include it so that they can control when this work is done. Usually, this time-consuming step is done when the application is initialized. If the user does not call `clAmdFftBakePlan`, this work is done during the first call to `clAmdFftEnqueueTransform`.

The OpenCL FFT kernels now are ready to execute as many times as needed.

a. Call `clAmdFftEnqueueTransform`. At this point, specify forward or reverse transform. The OpenCL `cl_mem` handles for the input buffer(s), output buffer(s) are provided--unless you want the transformed data to overwrite the input buffers, and (optionally) scratch buffer.

   `clAmdFftEnqueueTransform` performs one or more calls to the OpenCL function `clEnqueueNDRangeKernel`. Like `clEnqueueNDRangeKernel`, `clAmdFftEnqueueTransform` is a non-blocking call. The commands to execute the FFT compute kernel(s) are added to the OpenCL context queue to be executed asynchronously. An OpenCL event handle is returned to the caller. If multiple NDRangeKernel operations are queued, the final event handle is returned.

b. The application now can add additional OpenCL tasks to the OpenCL context's queue. For example, if the next step in the application's process is to apply a filter to the transformed data, the application would generate that `clEnqueueNDRangeKernel`, specifying the transform's output buffer(s) as the input to the filter kernel, and providing the transform's event handle to ensure proper synchronization.

c. If the application must access the transformed data directly, it must call one of the OpenCL functions for synchronizing the host computer's execution with the OpenCL device (for example: `clFinish()`).

- Terminate the library by calling `clAmdFftTeardown`.

1.5 FFTs of Real Data

When real data is subject to DFT transformation, the resulting complex output 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. clAmdFft provides 3 enumerated types to deal with real data FFTs:

- CLFFT_REAL
- CLFFT_HERMITIAN_INTERLEAVED
- CLFFT_HERMITIAN_PLANAR

The first enum specifies that the data is purely real. This can be used to feed real input or get back real output. The second and third enums specify layouts for storing FFT output. They are similar to the corresponding full complex enums

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

in the way they store real and imaginary components. The difference is that they store only about half of the complex output. Client applications can do just a forward transform and analyze the output. Or they can do some processing of the output and do a backward transform to get back real data. This is illustrated in Figure 1.1.

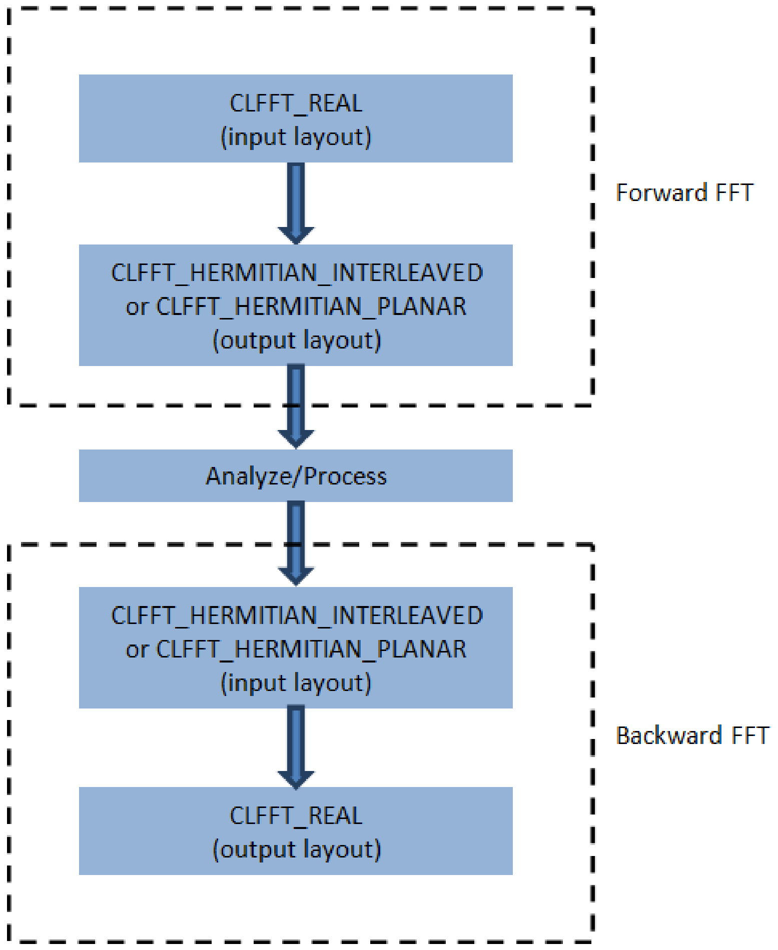


Figure 1.1 Forward and Backward Transform Processes

Let us consider a 1D real FFT of length N. The full output looks as shown in Figure 1.2.

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

Figure 1.2 1D Real FFT of Length N

Here, C* denotes *the complex conjugate of*. Since the values at indices greater than N/2 can be deduced from the first half of the array, clAmdFft stores data only up to the index N/2. This means that the output contains only 1 + N/2

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

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 Figure 1.3.

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

Figure 1.3 Example for N = 8.

Example for N = 7 is shown in Figure 1.4.

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

Figure 1.4 Example for N = 7.

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 0 through 3.

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

1.5.1 Supported Modes

Out-of-place transforms:

- CLFFT_REAL to CLFFT_HERMITIAN_INTERLEAVED
- CLFFT_REAL to CLFFT_HERMITIAN_PLANAR
- CLFFT_HERMITIAN_INTERLEAVED to CLFFT_REAL
- CLFFT_ CLFFT_HERMITIAN_PLANAR to CLFFT_REAL

In-place transforms:

- CLFFT_REAL to CLFFT_HERMITIAN_INTERLEAVED
- CLFFT_HERMITIAN_INTERLEAVED to CLFFT_REAL

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

1.5.2 Examples

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

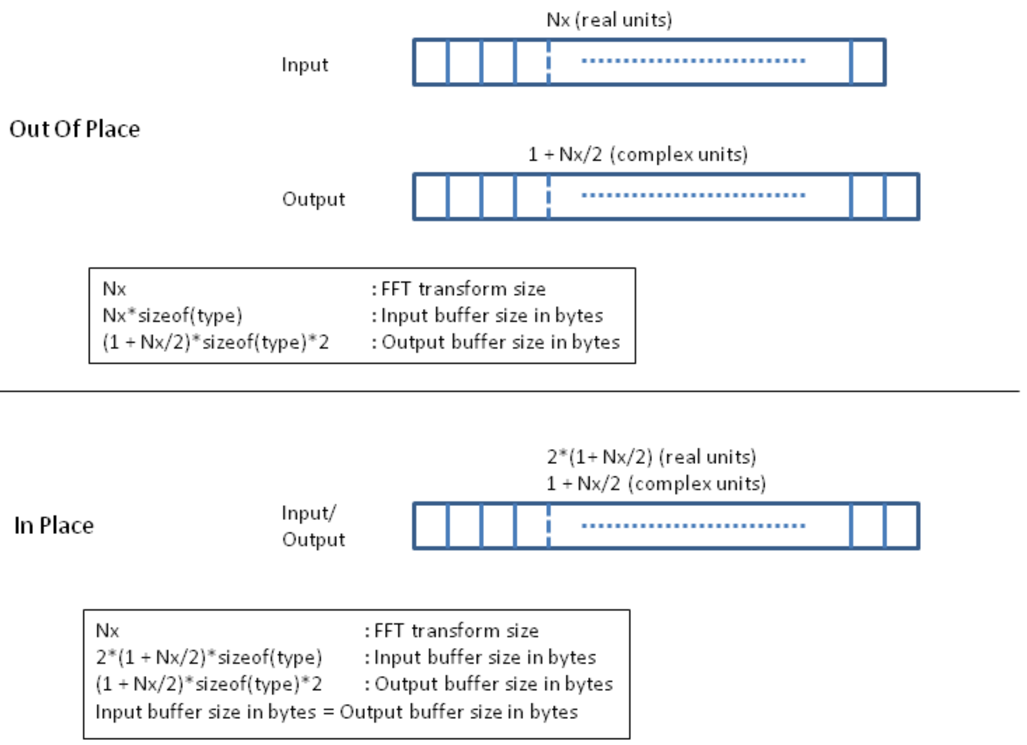

Nx (real units)
Input
Out Of Place
1 + Nx/2 (complex units)
Output
Nx : FFT transform size
Nx*sizeof(type) : Input buffer size in bytes
(1 + Nx/2)*sizeof(type)*2 : Output buffer size in bytes
2*(1+ Nx/2) (real units)
1 + Nx/2 (complex units)
In Place
Input/
Output
Nx : FFT transform size
2*(1 + Nx/2)*sizeof(type) : Input buffer size in bytes
(1 + Nx/2)*sizeof(type)*2 : Output buffer size in bytes
Input buffer size in bytes = Output buffer size in bytes

Figure 1.5 1D FFT - Real to Hermitian

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

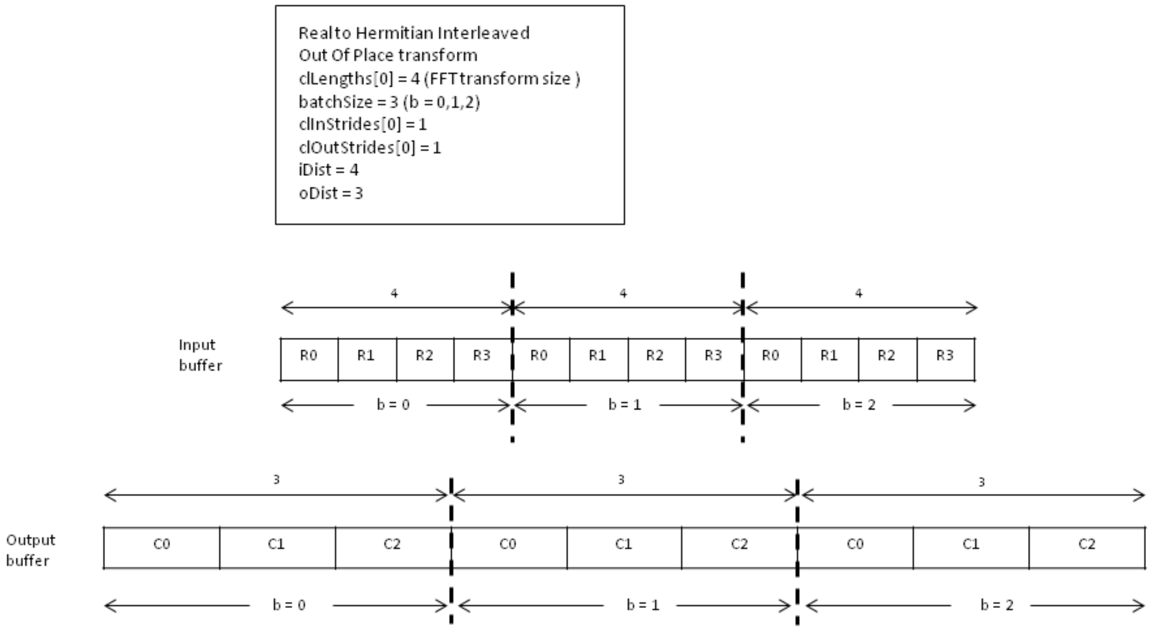


Figure 1.6 1D FFT - Real to Hermitian, Example 1

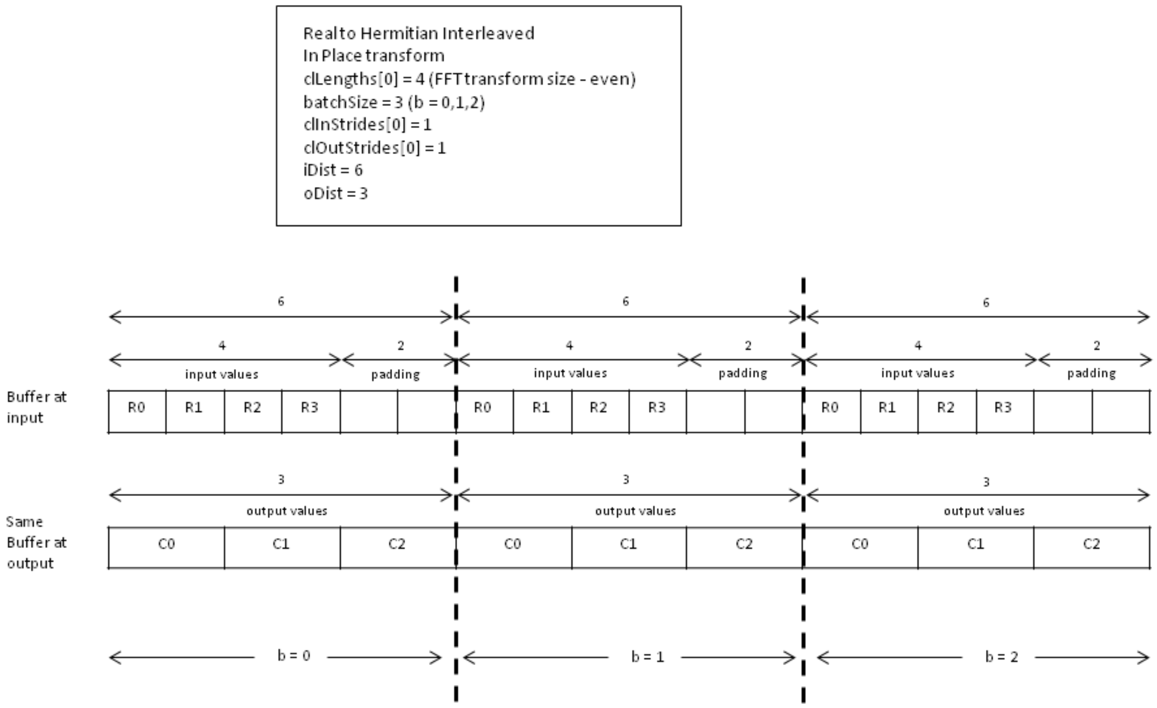


Figure 1.7 1D FFT - Real to Hermitian, Example 2

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

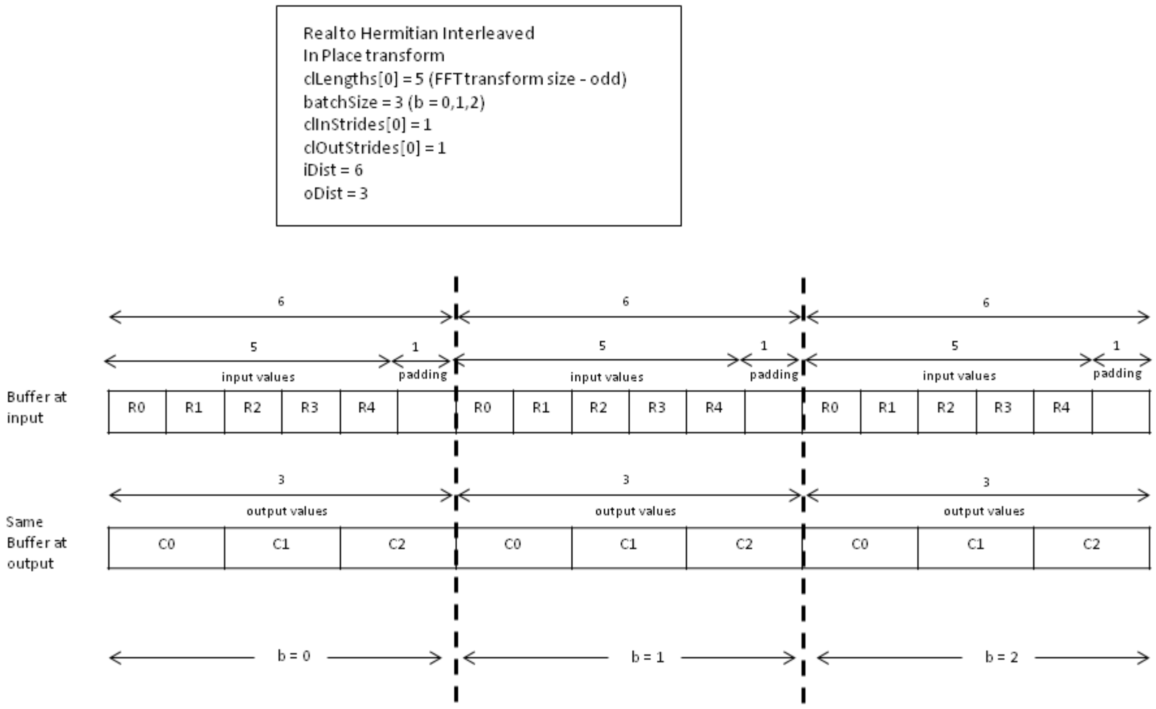


Figure 1.8 1D FFT - Real to Hermitian, Example 3

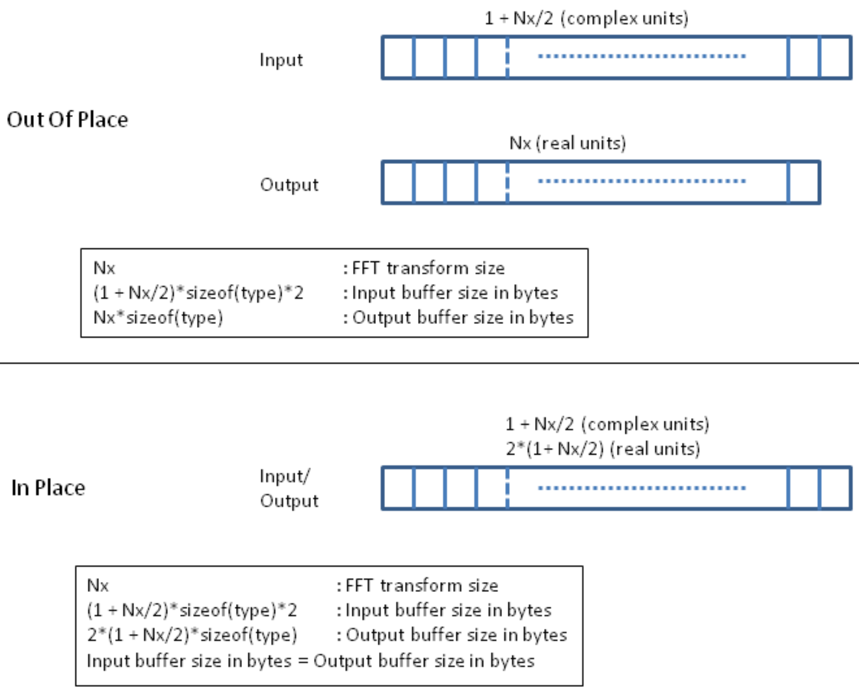

1 + Nx/2 (complex units)
Input
Out Of Place
Nx (real units)
Output
Nx : FFT transform size
(1 + Nx/2)*sizeof(type)*2 : Input buffer size in bytes
Nx*sizeof(type) : Output buffer size in bytes
1 + Nx/2 (complex units)
2*(1+ Nx/2) (real units)
In Place
Input/ Output
Nx : FFT transform size
(1 + Nx/2)*sizeof(type)*2 : Input buffer size in bytes
2*(1 + Nx/2)*sizeof(type) : Output buffer size in bytes
Input buffer size in bytes = Output buffer size in bytes

Figure 1.9 1D FFT - Hermitian to Real, Example

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

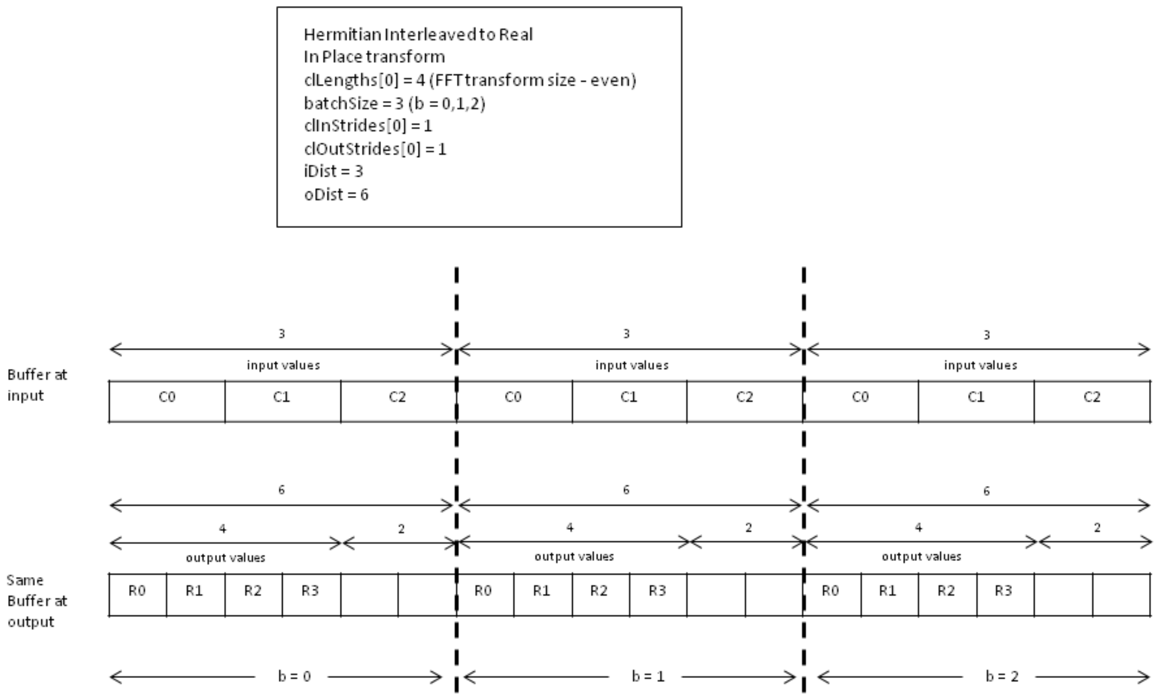
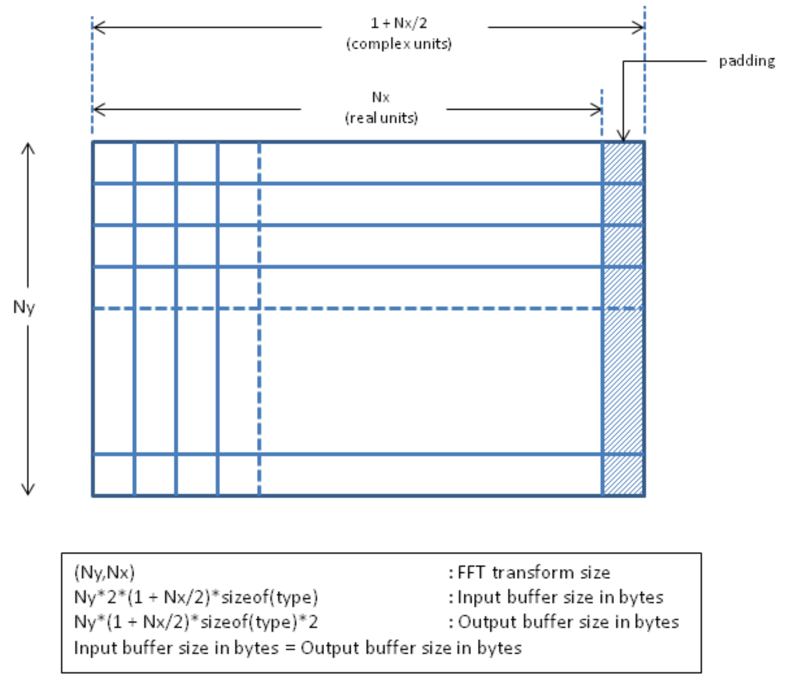

Hermitian Interleaved to Real
In Place transform
clLengths[0] = 4 (FFT transform size - even)
batchSize = 3 (b = 0,1,2)
clInStrides[0] = 1
clOutStrides[0] = 1
iDist = 3
oDist = 6

Figure 1.10 1D FFT - Hermitian to Real Example 4



(Ny,Nx)	: FFT transform size
Ny*2*(1 + Nx/2)*sizeof(type)	: Input buffer size in bytes
Ny*(1 + Nx/2)*sizeof(type)*2	: Output buffer size in bytes

Input buffer size in bytes = Output buffer size in bytes

Figure 1.11 2D FFT - Real to Hermitian In Place

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

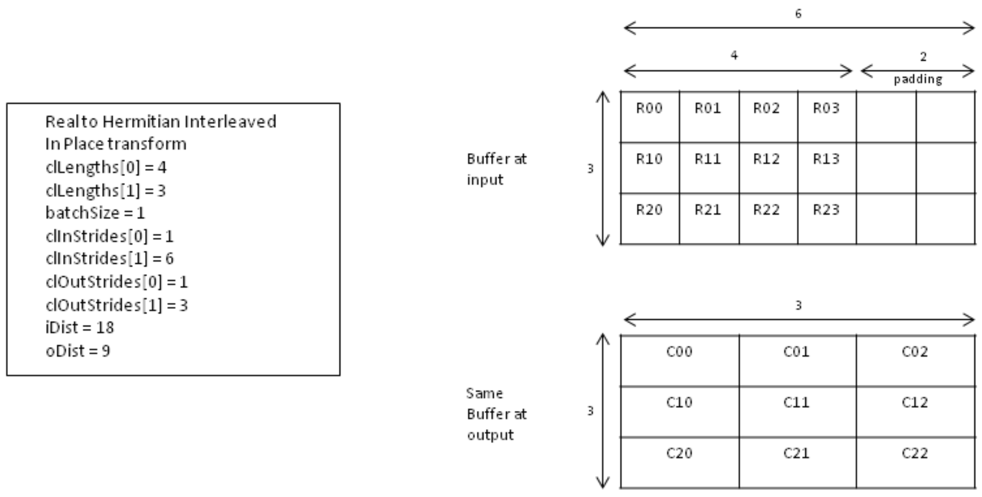


Figure 1.12 2D FFT Real to Hermitian, Example

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

Chapter 2 Class Documentation

2.1 `clAmdFftSetupData` Struct Reference

Data structure that can be passed to `clAmdFftSetup()` to control the behavior of the FFT runtime.

Public attributes are:

- cl_uint major
- cl_uint minor
- cl_uint patch
- cl_ulong debugFlags

2.1.1 Description

Data structure that can be passed to `clAmdFftSetup()` to control the behavior of the FFT runtime. This structure contains values that can be initialized before instantiation of the FFT runtime with `clAmdFftSetup()`. To initialize this structure, a pointer is passed to `clAmdFftInitSetupData()`, which clears the structure and sets the version member variables to current values.

2.1.2 Member Data Documentation

2.1.2.1 `cl ulong clAmdFftSetupData::debug` Flags

Bitwise flags that control the behavior of library debug logic.

2.1.2.2 `cl uint clAmdFftSetupData::major`

Major version number of the project; signifies major API changes.

2.1.2.3 `cl uint clAmdFftSetupData::minor`

Minor version number of the project; minor API changes that could break backwards compatibility.

2.1.2.4 `cl uint clAmdFftSetupData::patch`

Patch version number of the project, always incrementing number; signifies change over time.

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

2.2 Global Values

This section introduces all global values available from the clAmdFft header file. These values typically are passed into the clAmdFft API to control some behavior of the FFT plan.

2.2.1 Detailed Description

`clAmdFft.h` defines all of the public interfaces and types that are meant to be used by clFFT clients This is the one public header file that must be consumed by clFFT clients.

It is written to adhere to native C interfaces to make the clAmdFft library as portable as possible; it is callable from C, C++, .NET and Fortran, either with the proper linking or using wrapper classes.

2.2.2 Define Documentation

2.2.2.1 #define CLAMDFFTAPI

This preprocessor definition is the standard way of making exporting APIs from a DLL simpler. All files within this DLL are compiled with the CLAMDFFT_EXPORTS symbol defined on the command line. This symbol must not be defined on any project that uses this DLL. This way any other project whose source files include this file see clAmdFft functions as being imported from a DLL, whereas this DLL sees symbols defined with this macro as being exported.

2.2.2.2 #define CLFFT DUMP PROGRAMS 0x1

BitMasks to be used with `clAmdFftSetupData.debugFlags`.

2.2.3 Enumeration Type Documentation

2.2.3.1 enum clAmdFftDim

The dimensions of the input and output buffers that are fed into all FFT transforms.

Enumerator

`CLFFT_1D` — 1 Dimensional FFT transform (default).

`CLFFT_2D` — 2 Dimensional FFT transform.

`CLFFT_3D` — 3 Dimensional FFT transform.

`ENDDIMENSION` — This value is always last; it marks the length of `clAmdFftDim`.

2.2.3.2 enum clAmdFftDirection

This is the expected direction of each FFT, time or the frequency domains.

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

Enumerator

`CLFFT_FORWARD` — FFT transform from the time to the frequency domain.

`CLFFT_BACKWARD` — FFT transform from the frequency to the time domain.

`CLFFT_MINUS` — Alias for the forward transform.

`CLFFT_PLUS` — Alias for the backward transform.

`ENDDIRECTION` — This value is always last; it marks the length of `clAmdFftDirection`.

2.2.3.3 enum clAmdFftLayout

These are the expected layouts of the complex numbers.

Currently, only the `CLFFT_COMPLEX_INTERLEAVED` and `CLFFT_COMPLEX_PLANAR` formats are supported.

Enumerator

`CLFFT_COMPLEX_INTERLEAVED` — An array of complex numbers, with real and imaginary components together (default).

`CLFFT_COMPLEX_PLANAR` — Arrays of real components and arrays of imaginary components that have been separated out.

`CLFFT_HERMITIAN_INTERLEAVED` — Compressed form of complex numbers; complex conjugates not stored, real and imaginary components in same array.

`CLFFT_HERMITIAN_PLANAR` — Compressed form of complex numbers; complex-conjugates not stored, real and imaginary components in separate arrays.

`CLFFT_REAL` — An array of real numbers, with no corresponding imaginary components.

`ENDLAYOUT` — This value is always last; it marks the length of `clAmdFftLayout`.

2.2.3.4 enum clAmdFftPrecision

This is the expected precision of each FFT. Strides and Pitches.

Enumerator

`CLFFT_SINGLE` — An array of complex numbers, with real and imaginary components as floats (default).

`CLFFT_DOUBLE` — An array of complex numbers, with real and imaginary components as doubles.

`CLFFT_SINGLE_FAST` — Faster implementation preferred.

`CLFFT_DOUBLE_FAST` — Faster implementation preferred.

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

`ENDPRECISION` — This value is always last; it marks the length of `clAmdFftPrecision`.

2.2.3.5 enum clAmdFftResultLocation

Are the input buffers overwritten with the results.

Enumerator

`CLFFT_INPLACE` — The input and output buffers are the same (default).

`CLFFT_OUTOFPLACE` — Separate input and output buffers.

`ENDPLACE` — This value is always last; it marks the length of `clAmdFftPlaceness`.

2.2.3.6 enum clAmdFftStatus

The clAmdFft error codes definition, incorporating OpenCL error definitions.

This enumeration is a superset of the OpenCL error codes. For example, `CL_OUT_OF_HOST_MEMORY`, which is defined in cl.h is aliased as `CLFFT_OUT_OF_HOST_MEMORY`. The set of basic OpenCL error codes is extended to add extra values specific to the clAmdFft package.

Enumerator

`CLFFT_NOTIMPLEMENTED` — Functionality is not implemented yet.

`CLFFT_FILE_NOT_FOUND` — Tried to open an existing file on the host system, but failed.

`CLFFT_FILE_CREATE_FAILURE` — Tried to create a file on the host system, but failed.

`CLFFT_VERSION_MISMATCH` — Version conflict between client and library.

`CLFFT_INVALID_PLAN` — Requested plan could not be found.

`CLFFT_DEVICE_NO_DOUBLE` — Double precision not supported on this device.

2.2.3.7 enum clAmdFftResultTransposed

This determines whether the result is returned in original order. It is valid only for dimensions greater than 1.

Enumerator

`CLFFT_NOTRANSPOSE` -	The results are returned in the original preserved order (default).
`CLFFT_TRANSPOSED` -	The result is transposed where transpose kernel is supported (possibly faster).

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

`ENDTRANSPOSED` -This value is always last and marks the length of `clAmdFftResultTransposed`.

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

Chapter 3 Function Documentation

3.1 Libary Management Functions

Initialize an clAmdFftSetupData struct for the client

Function `clAmdFftStatus clAmdFftInitSetupData (clAmdFftSetupData * setupData)`

Description Initialize an `clAmdFftSetupData` struct for the client.
`clAmdFftSetupData` is passed to `clAmdFftSetup` to control the behavior of the FFT runtime.

Parameters

`out`	*setupData*	Data structure is cleared, then initialized with version information and default values.

Returns Enum describing error condition; superset of OpenCL error codes.

Release all internal resources

Function `CLAMDFFTAPI clAmdFftStatus clAmdFftTeardown ()`

Description Release all internal resources.
Call when client is done with this FFT library, allowing the library to destroy all resources it has cached.

Parameters

None.

Returns Enum describing error condition; superset of OpenCL error codes.

Query the FFT library for version information

Function `CLAMDFFTAPI clAmdFftStatus clAmdFftGetVersion (cl_uint * major, cl_uint * minor,cl_uint * patch)`

Description Query the FFT library for version information.
Return the major, minor, and patch version numbers associated with this FFT library.

Parameters

`out`	*major*	Major functionality change.
`out`	*minor*	Minor functionality change.
`out`	*patch*	Bug fixes, documentation changes, no new features introduced.

Returns Enum describing error condition; superset of OpenCL error codes.

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

Initialize internal FFT resources

Function `CLAMDFFTAPI clAmdFftStatus clAmdFftSetup (const clAmdFftSetupData * setupData)`

Description Initialize internal FFT resources.
The AMD FFT implementation caches kernels, programs, and buffers for its internal use.

Parameters

`in`	*setupData*	Data structure that can be passed into the setup routine to control FFT generation behavior and debug functionality.

Returns Enum describing error condition; superset of OpenCL error codes.

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

3.2 Plan Management Functions

Create a plan object initialized entirely with default values

Function `CLAMDFFTAPI clAmdFftStatus clAmdFftCreateDefaultPlan (clAmdFftPlanHandle * plHandle, cl_context context, const clAmdFftDim dim, const size_t * clLengths)`

Description Create a plan object initialized entirely with default values.
A plan is a repository of state for calculating FFTs. Allows the runtime to pre-calculate kernels, programs, and buffers, then associate them with buffers of specified dimensions.

Parameters

`out`	*plHandle*	Handle to the newly created plan.
`in`	*context*	Client is responsible for providing an OpenCL context for the plan.
`in`	*dim*	The dimensionality of the FFT transform; describes how many elements are in the array.
`in`	*clLengths*	An array of lengths, of size `dim`. Each value describes the length of additional dimensions.

Returns Enum describing error condition; superset of OpenCL error codes.

Create a copy of an existing plan

Function `CLAMDFFTAPI clAmdFftStatus clAmdFftCopyPlan (clAmdFftPlanHandle * out_plHandle, cl_context new_context, clAmdFftPlanHandle in_plHandle)`

Description Create a copy of an existing plan.
This API allows a client to create a new plan based upon an existing plan. This is a convenience function provided for quickly creating plans that are similar, but can differ slightly.

Parameters

`out`	*out_plHandle*	Handle to the newly created plan that is based on in_plHandle.
`in`	*new_context*	Client is responsible for providing a new context for the new plan.
`in`	*in_plHandle*	Handle to a plan to be copied, previously created.

Returns Enum describing error condition; superset of OpenCL error codes.

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

Prepare the plan for execution

Function

```
CLAMDFFTAPI clAmdFftStatus clAmdFftBakePlan (clAmdFftPlanHandle plHandle,cl_uint
numQueues, cl_command_queue * commQueueFFT, void(CL_CALLBACK * pfn_notify)
(clAmdFftPlanHandle plHandle, void * user_data), void * user_data)
```

Description

Prepare the plan for execution.
After all plan parameters are set, the client has the option of 'baking' the plan, which tells the runtime no more changes to the plan's parameters are expected, and the OpenCL kernels are to be compiled. This optional function allows the client application to perform this function when the application is being initialized instead of on the first execution. At this point, the clAmdFft runtime applies all implemented optimizations, possibly including running kernel experiments on the devices in the plan context.
Users should assume that this function takes a long time to execute. If a plan is not baked before being executed, users should assume that the first call to `clAmdFftEnqueueTransform` takes a long time to execute.
If any significant parameter of a plan is changed after the plan is baked (by a subsequent call to one of the `clAmdFftSetPlan____` functions), it is not considered an error. Instead, the plan reverts to the unbaked state, discarding the benefits of the baking operation.

Parameters

`in`	*plHandle*	Handle to a previously created plan.
`in`	*numQueues*	Number of command queues in commQueueFFT. 0 is a valid value, in which case client does not want the runtime to run load experiments and only pre-calculate state information.
`in`	*commQueueFFT*	An array of `cl_command_queues` created by the client; the command queues must be a proper subset of the devices included in the plan context.
`in`	*pfn_notify*	A function pointer to a notification routine. The notification routine is a callback function that an application can register and that is called when the program executable has been built (successfully or unsuccessfully). Currently, this parameter MUST be NULL or nullptr.
`in`	*user_data*	Passed as an argument when `pfn_notify` is called. Currently, this parameter MUST be NULL or nullptr.

Returns

Enum describing error condition; superset of OpenCL error codes

Release the resources of a plan

Function

```
CLAMDFFTAPI clAmdFftStatus clAmdFftDestroyPlan (clAmdFftPlanHandle * plHandle)
```

Description

Release the resources of a plan.
A plan can include kernels, programs, and buffers associated with it that consume memory. When a plan is no longer needed, the client must release the plan.

Parameters

`in, out`	*plHandle*	Handle to a previously created plan.

Returns

Enum describing error condition; superset of OpenCL error codes.

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

Retrieve the OpenCL context of a previously created plan

Function

```
CLAMDFFTAPI clAmdFftStatus clAmdFftGetPlanContext (const clAmdFftPlanHandle
plHandle, cl_context * context)
```

Description

Retrieve the OpenCL context of a previously created plan.
User should pass a reference to an cl_context variable, which will be changed to point to a context set in the specified plan.

Parameters

`in`	*plHandle*	Handle to a previously created plan.
`out`	*context*	Reference to user allocated `cl_context`, which points to the context set in plan.

Returns

Enum describing error condition; superset of OpenCL error codes.

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

Enqueue an FFT transform operation, and return immediately (non-blocking)

Function

```
CLAMDFFTAPI clAmdFftStatus clAmdFftEnqueueTransform (clAmdFftPlanHandle
plHandle, clAmdFftDirection dir, cl_uint numQueuesAndEvents, cl_command_queue *
commQueues, cl_uint numWaitEvents, const cl_event * waitEvents, cl_event *
outEvents, cl_mem * inputBuffers, cl_mem * outputBuffers, cl_mem tmpBuffer)
```

Description

`Enqueue` an FFT transform operation, and return immediately (non-blocking).
This transform API is specific to the interleaved complex format, taking an input buffer with real and imaginary components paired together, and outputting the results into an output buffer in the same format.

Parameters

`in`	*plHandle*	Handle to a previously created plan.
`in`	*dir*	Forwards or backwards transform.
`in`	*numQueuesAndEvents*	Number of command queues in commQueues; number of expected events to be returned in `outEvents`.
`in`	*commQueues*	An array of `cl_command_queues` created by the client; the command queues must be a proper subset of the devices included in the plan context.
`in`	*numWaitEvents*	Specify the number of elements in the `eventWaitList` array.
`in`	*waitEvents*	Events that this transform should wait to complete before executing on the device.
`out`	*outEvents*	The runtime fills this array with events corresponding 1-to-1 with the input command queues passed in commQueues. This parameter can be NULL or nullptr, in which case client does not need to receive notifications when transforms are finished; otherwise, if not NULL, the client must allocate this array, with at least as many elements as specified in `numQueuesAndEvents`.
`in`	*inputBuffers*	An array of `cl_mem` objects that contain data for processing by the FFT runtime. Only OpenCL buffer objects are supported; OpenCL image objects return an error code. If the transform is in place, the FFT results overwrite the input buffers.
`out`	*outputBuffers*	An array of `cl_mem` objects that store the results of out-of-place transforms. Only OpenCL buffer objects are supported; OpenCL image objects return an error code. If the transform is in place, this parameter can be NULL, in which case it is completely ignored.
`in`	*tmpBuffer*	A `cl_mem` object that is reserved as a temporary buffer for FFT processing. Only OpenCL buffer objects are supported; OpenCL image objects return an error code. If `clTmpBuffers` is NULL or nullptr, and the runtime needs temporary storage, an internal temporary buffer is created on the fly and managed by the runtime.

Returns

Enum describing error condition; superset of OpenCL error codes.

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

3.3 Plan Accessors Functions

Retrieve the floating point precision of the FFT data

Function `CLAMDFFTAPI clAmdFftStatus clAmdFftGetPlanPrecision (const clAmdFftPlanHandle plHandle, clAmdFftPrecision * precision)`

Description Retrieve the floating point precision of the FFT data.

Parameters

`in`	*plHandle*	Handle to a previously created plan.
`out`	*precision*	Reference to user `clAmdFftPrecision` enum.

Returns Enum describing error condition; superset of OpenCL error codes.

Set the floating point precision of the FFT data

Function `CLAMDFFTAPI clAmdFftStatus clAmdFftSetPlanPrecision (clAmdFftPlanHandle plHandle, clAmdFftPrecision precision)`

Description Set the floating point precision of the FFT data.
Set the plan property which is the precision of the FFT complex data in the plan.

Parameters

`in`	*plHandle*	Handle to a previously created plan.
`in`	*precision*	Reference to user `clAmdFftPrecision` enum.

Currently, only `CLFFT_SINGLE` and `CLFFT_SINGLE_FAST` are supported.

Returns Enum describing error condition; superset of OpenCL error codes.

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

Retrieve the scaling factor that should be applied to the FFT data

Function
```
CLAMDFFTAPI clAmdFftStatus clAmdFftGetPlanScale (const clAmdFftPlanHandle
plHandle, clAmdFftDirection dir, cl_float * scale)
```

Description Retrieve the scaling factor to be applied to the FFT data.

Parameters

`in`	*plHandle*	Handle to a previously created plan.
`in`	*dir*	The direction to which the scaling factor applies.
`out`	*scale*	Reference to user `cl_float` variable.

Returns Enum describing error condition; superset of OpenCL error codes.

Set the scaling factor that should be applied to the FFT data

Function
```
CLAMDFFTAPI clAmdFftStatus clAmdFftSetPlanScale (clAmdFftPlanHandle plHandle,
clAmdFftDirection dir, cl_float scale)
```

Description Retrieve the floating point scaling factor to be multiplied across the FFT data.

Parameters

`in`	*plHandle*	Handle to a previously created plan.
`in`	*dir*	The direction to which the scaling factor applies.
`in`	*scale*	Reference to user `cl_float` variable.

Returns Enum describing error condition; superset of OpenCL error codes.

Retrieve the number of discrete arrays that this plan can handle concurrently

Function
```
CLAMDFFTAPI clAmdFftStatus clAmdFftGetPlanBatchSize (const clAmdFftPlanHandle
plHandle, size_t * batchSize)
```

Description Retrieve the number of discrete arrays that this plan can handle concurrently.

Parameters

`in`	*plHandle*	Handle to a previously created plan.
`out`	*batchSize*	How many discrete FFTs are to be performed.

Returns Enum describing error condition; superset of OpenCL error codes.

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

Set the number of discrete arrays that this plan can handle concurrently

Function `CLAMDFFTAPI clAmdFftStatus clAmdFftSetPlanBatchSize (clAmdFftPlanHandle plHandle, size_t batchSize)`

Description Set the number of discrete arrays (1D or 2D) to be batched (that this plan can handle concurrently).

Parameters

`in`	*plHandle*	Handle to a previously created plan.
`in`	*batchSize*	How many discrete FFTs are to be performed.

Returns Enum describing error condition; superset of OpenCL error codes.

Retrieve the dimensionality of FFTs to be transformed in the plan

Function `CLAMDFFTAPI clAmdFftStatus clAmdFftGetPlanDim (const clAmdFftPlanHandle plHandle, clAmdFftDim * dim, cl_uint * size)`

Description Retrieve the dimensionality of FFTs to be transformed in the plan.
Queries a plan object and retrieves the dimensionality that the plan is set for. A size is returned to help the client allocate the proper storage to hold the dimensions in a further call to `clAmdFftGetPlanLength`.

Parameters

`in`	*plHandle*	Handle to a previously created plan.
`out`	*dim*	The dimensionality of the FFTs to be transformed
`out`	*size*	Value used to allocate an array to hold the FFT dimensions.

Returns Enum describing error condition; superset of OpenCL error codes.

Set the dimensionality of FFTs to be transformed by the plan

Function `CLAMDFFTAPI clAmdFftStatus clAmdFftSetPlanDim (clAmdFftPlanHandle plHandle, const clAmdFftDim dim)`

Description Set the dimensionality of FFTs to be transformed by the plan.

Parameters

in	*plHandle*	Handle to a previously created plan.
in	*dim*	The dimensionality of the FFTs to be transformed.

Returns Enum describing error condition; superset of OpenCL error codes.

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

Retrieve the length of each dimension of the FFT

Function `CLAMDFFTAPI clAmdFftStatus clAmdFftGetPlanLength (const clAmdFftPlanHandle plHandle, const clAmdFftDim dim, size_t * clLengths)`

Description Retrieve the length of each dimension of the FFT.

Parameters

`in`	*plHandle*	Handle to a previously created plan.
`in`	*dim*	The dimension of the length parameters; describes how many elements are in the array.
`out`	*clLengths*	An array describing the length of each discrete dimension of the FFT computation. The size of the array is given by the `dim` parameter.

Returns Enum describing error condition; superset of OpenCL error codes

Set the length of each dimension of the FFT

Function `CLAMDFFTAPI clAmdFftStatus clAmdFftSetPlanLength (clAmdFftPlanHandle plHandle, const clAmdFftDim dim, const size_t * clLengths)`

Description Set the length of each discrete dimension of the FFT.

Parameters

`in`	*plHandle*	Handle to a previously created plan.
`in`	*dim*	The dimension of the length parameters. Describes how many elements are in the array.
`in`	*clLengths*	An array describing the length of each discrete dimension of the FFT computation. The size of the array is given by the `dim` parameter.

Currently, all lengths must be powers of 2.

Returns Enum describing error condition; superset of OpenCL error codes

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

Retrieve the distance between consecutive elements for input buffers in a dimension

Function

```
CLAMDFFTAPI clAmdFftStatus clAmdFftGetPlanInStride (const clAmdFftPlanHandle
plHandle, const clAmdFftDim dim, size_t * clStrides)
```

Description Retrieve the distance between consecutive elements for input buffers in a dimension. Depending on how the dimension is set in the plan (for 2D or 3D FFTs), strideY or strideZ can be safely ignored.

Parameters

`in`	*plHandle*	Handle to a previously created plan.
`in`	*dim*	The dimension of the stride parameters. Describes how many elements are in the array.
`out`	*clStrides*	An array of strides, of size 'dim'. Usually strideX=1 so that successive elements in the first dimension are stored contiguously. Typically strideY=LenX, strideZ=LenX∗LenY such that successive elements in the second and third dimensions are stored contiguously.

Returns Enum describing error condition; superset of OpenCL error codes.

Set the distance between consecutive elements for input buffers in a dimension

Function

```
CLAMDFFTAPI clAmdFftStatus clAmdFftSetPlanInStride (clAmdFftPlanHandle plHandle,
const clAmdFftDim dim, size_t * clStrides)
```

Description Set the distance between consecutive elements for input buffers in a dimension. Set the distance between elements in a given dimension (units are in terms of clAmdFftPrecision).

Parameters

in	*plHandle*	Handle to a previously created plan.
in	*dim*	The dimension of the stride parameters. Describes how many elements are in the array.
in	*clStrides*	An array of strides, of size `dim`. See Strides and Pitches for details.

Returns Enum describing error condition; superset of OpenCL error codes.

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

Retrieve the distance between consecutive elements for output buffers in a dimension

Function `CLAMDFFTAPI clAmdFftStatus clAmdFftGetPlanOutStride (const clAmdFftPlanHandle plHandle, const clAmdFftDim dim, size_t * clStrides)`

Description Retrieve the distance between consecutive elements for output buffers in a dimension. Depending on how the dimension is set in the plan (for 2D or 3D FFTs), strideY or strideZ can be safely ignored.

Parameters

`in`	*plHandle*	Handle to a previously created plan.
`in`	*dim*	The dimension of the stride parameters; describes how many elements are in the array
`out`	*clStrides*	An array of strides, of size 'dim'. Usually strideX=1 so that successive elements in the first dimension are stored contiguously. Typically strideY=LenX, strideZ=LenX∗LenY such that successive elements in the second and third dimensions are stored contiguously.

Returns Enum describing error condition; superset of OpenCL error codes.

Set the distance between consecutive elements for output buffers in a dimension

Function `CLAMDFFTAPI clAmdFftStatus clAmdFftSetPlanOutStride (clAmdFftPlanHandle plHandle, const clAmdFftDim dim, size_t * clStrides)`

Description Set the distance between consecutive elements for output buffers in a dimension. Set the distance between elements in a given dimension (units are in terms of `clAmdFftPrecision`).

Parameters

`in`	*plHandle*	Handle to a previously created plan.
`in`	*dim*	The dimension of the stride parameters. Describes how many elements are in the array.
`in`	*clStrides*	An array of strides, of size `dim`. Usually strideX=1, so that successive elements in the first dimension are stored contiguously. Typically, strideY=LenX, and strideZ=LenX∗LenY, so that successive elements in the second and third dimensions are stored contiguously.

See also `clAmdFftSetPlanInStride`

Returns Enum describing error condition; superset of OpenCL error codes.

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

Retrieve the distance between Array objects

Function `CLAMDFFTAPI clAmdFftStatus clAmdFftGetPlanDistance (const clAmdFftPlanHandle plHandle, size_t * iDist, size_t * oDist)`

Description Retrieve the distance between array objects.
Pitch is the distance between each discrete array object in an FFT array. This is only used for `array` dimensions in `clAmdFftDim`; see `clAmdFftSetPlanDimension` (units are in terms of `clAmdFftPrecision`).

Parameters

`in`	*plHandle*	Handle to a previously created plan.
`out`	*iDist*	The distance between the beginning elements of the discrete array objects in memory on input. For contiguous arrays in memory, iDist=(strideX*strideY*strideZ).
`out`	*oDist*	The distance between the beginning elements of the discrete array objects in memory on output. For contiguous arrays in memory, oDist=(strideX*strideY*strideZ).

Returns Enum describing error condition; superset of OpenCL error codes.

Set the distance between Array objects

Function `CLAMDFFTAPI clAmdFftStatus clAmdFftSetPlanDistance (clAmdFftPlanHandle plHandle, size_t iDist, size_t oDist)`

Description Set the distance between Array objects.
Pitch is the distance between each discrete array object in an FFT array. This is only used for `array` dimensions in `clAmdFftDim`; see `clAmdFftSetPlanDimension` (units are in terms of `clAmdFftPrecision`).

Parameters

`in`	*plHandle*	Handle to a previously created plan.
`out`	*iDist*	The distance between the beginning elements of the discrete array objects in memory on input. For contiguous arrays in memory, iDist=(strideX*strideY*strideZ).
`out`	*oDist*	The distance between the beginning elements of the discrete array objects in memory on output. For contiguous arrays in memory, oDist=(strideX*strideY*strideZ).

Returns Enum describing error condition; superset of OpenCL error codes.

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

Retrieve the expected layout of the input and output buffers

Function `CLAMDFFTAPI clAmdFftStatus clAmdFftGetLayout (const clAmdFftPlanHandle plHandle, clAmdFftLayout * iLayout, clAmdFftLayout * oLayout)`

Description Retrieve the expected layout of the output buffers.
Output buffers can be filled with either hermitian or complex numbers. Complex numbers can be stored in various layouts; this informs the FFT engine what layout to produce on output

Parameters

`in`	*plHandle*	Handle to a previously created plan.
`out`	*iLayout*	Indicates how the input buffers are laid out in memory.
`out`	*oLayout*	Indicates how the output buffers are laid out in memory.

Returns Enum describing error condition; superset of OpenCL error codes.

Set the expected layout of the input and output buffers

Function `CLAMDFFTAPI clAmdFftStatus clAmdFftSetLayout (clAmdFftPlanHandle plHandle, clAmdFftLayout iLayout, clAmdFftLayout oLayout)`

Description Set the expected layout of the output buffers.
Output buffers can be filled with either hermitian or complex numbers. Complex numbers can be stored in various layouts; this informs the FFT engine what layout to produce on output.

Parameters

`in`	*plHandle*	Handle to a previously created plan.
`in`	*iLayout*	Indicates how the input buffers are laid out in memory.
`in`	*oLayout*	Indicates how the output buffers are laid out in memory.

Returns Enum describing error condition; superset of OpenCL error codes.

Determine if the input buffers are going to be overwritten with results

Function `CLAMDFFTAPI clAmdFftStatus clAmdFftGetResultLocation (const clAmdFftPlanHandle plHandle, clAmdFftResultLocation * placeness)`

Description Determine if the input buffers are going to be overwritten with results.
If the setting is to do an in-place transform the input buffers are overwritten with the results of the transform. If the setting is for out-of-place transforms, the engine knows to look for separate output buffers on the `enqueue` call.

Parameters

`in`	*plHandle*	Handle to a previously created plan.
`out`	*placeness*	Tells the FFT engine to overwrite the input buffers, or to expect output buffers for results.

Returns Enum describing error condition; superset of OpenCL error codes.

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

Set whether the input buffers are going to be overwritten with results

Function `CLAMDFFTAPI clAmdFftStatus clAmdFftSetResultLocation (clAmdFftPlanHandle plHandle, clAmdFftResultLocation placeness)`

Description Set whether the input buffers are going to be overwritten with results.
If the setting is to do an in-place transform, the input buffers are overwritten with the results of the transform. If the setting is for out-of-place transforms, the engine knows to look for separate output buffers on the `enqueue` call.

Parameters

`in`	*plHandle*	Handle to a previously created plan.
`in`	*placeness*	Tells the FFT engine to overwrite the input buffers or to expect output buffers for results.

Returns Enum describing error condition; superset of OpenCL error codes.

Get buffer size (in bytes), which may be needed internally for an intermediate buffer

Function `CLAMDFFTAPI clAmdFftStatus clAmdFftGetTmpBufSize (const clAmdFftPlanHandle plHandle, size_t * buffersize)`

Description Get buffer size (in bytes), which may be needed internally for an intermediate buffer.
Very large FFT transforms may need multiple passes, and the operation might need a temporary buffer to hold intermediate results. This function is only valid after the plan is baked; otherwise, an invalid operation error is returned. If `buffersize` returns 0, the runtime needs no temporary buffer.

Parameters

`in`	*plHandle*	Handle to a previously created plan.
`out`	*buffersize*	Size in bytes for intermediate buffer.

Returns Enum describing error condition; superset of OpenCL error codes.

Retrieve the final transpose setting of a multi-dimensional FFT

Function `CLAMDFFTAPI clAmdFftStatus clAmdFftGetPlanTransposeResult(const clAmdFftPlanHandle plHandle, clAmdFftResultTransposed *transposed)`

Description Retrieve the final transpose setting of a muti-dimensional FFT. A multi-dimensional FFT typically transposes the data several times during calculation. If the client does not care about the final transpose to put data back in proper dimension, the final transpose can be skipped for possible speed improvements.

Parameters

`in`	*plHandle*	Handle to a previously created plan.
`out`	*transposed*	Parameter specifies whether the final transpose can be skipped.

Returns Enum describing error condition; superset of OpenCL error codes.

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.

Set the final transpose setting of a multi-dimensional FFT

Function

```
CLAMDFFTAPI clAmdFftStatus clAmdFftSetPlanTransposeResult(clAmdFftPlanHandle
plHandle, clAmdFftResultTransposed transposed)
```

Description

Set the final transpose setting of a muti-dimensional FFT. A multi-dimensional FFT typically transposes the data several times during calculation. If the client does not care about the final transpose to put data back in proper dimension, the final transpose can be skipped for possible speed improvements.

Parameters

`in`	*plHandle*	Handle to a previously created plan.
`out`	*transposed*	Parameter specifies whether the final transpose can be skipped.

Returns

Enum describing error condition; superset of OpenCL error codes.

Copyright © 2012 Advanced Micro Devices, Inc. All rights reserved.