Installing Spiral & FFTX: Building FFTX, using its APIs, RTC and Linking with its Libraries

Patrick Broderick (SpiralGen)

Obtaining FFTX and Spiral

- **In order to install and build FFTX, first you must install SPIRAL. FFTX uses SPIRAL to generate the code used for run-time compilation (RTC), as well as for the libraries and examples. All the code is available from github:**
- **Links for the various repositories:**

https://github.com/spiral-software/spiral-software

https://github.com/spiral-software/spiral-package-fftx

https://github.com/spiral-software/spiral-package-simt

https://github.com/spiral-software/spiral-package-mpi

https://github.com/spiral-software/spiral-package-jit

https://github.com/spiral-software/fftx

Pre-requisites:

- C/C++ compiler and make tools (linux)
- Visual Studio (Windows)
- macOS 10.14 or later, Xcode + Xcode command line tools (MAC)
- CMake, 3.14 or later
- Python 3.6 or later
- MAC users: scripts check for python version 3, won't interfere with a version 2

Each repository contains a README with detailed installation instructions, including pre-requisites, for SPIRAL and FFTX. When building, please follow the directions carefully.

NOTE: A special branch is added to the FFTX repository for the tutorial; it is intended to memorialize the material and examples used here.

Build SPIRAL

First create a directory where you will install the software, e.g., ~/work

```
mkdir work
cd work
git clone https://github.com/spiral-software/spiral-software
pushd spiral-software
export SPIRAL_HOME=`pwd`
pushd namespaces/packages
git clone https://github.com/spiral-software/spiral-package-fftx fftx
git clone https://github.com/spiral-software/spiral-package-simt simt
git clone https://github.com/spiral-software/spiral-package-mpi mpi
git clone https://github.com/spiral-software/spiral-package-jit jit
popd
mkdir build
cd build
cmake ..
make install -j
popd

##          export PATH=$SPIRAL_HOME/bin:$PATH     ##  Not required, but may be nice to have
```

Build FFTX

The FFTX installation does not have to be co-located with Spiral, but for convenience we'll keep them at the same root folder (i.e., ~/work)

FFTX can be used with either fixed, prebuilt libraries or real-time compilation to generate and compile the necessary code as needed. Edit the config-fftx-libs.sh script to define which libraries, if any, will be pre-built. For the tutorial, we will only prebuild the MDDFT & IMDDFT libraries with 3 small sizes (for demonstration only) and will build for HIP (change appropriately for your platform). The script is self-explanatory, simply set the options True or False as desired. Once the script is edited as desired we can build FFTX. The script contains several entries of the following form:

```
##  Build the 3D DFT (complex to complex) library
MDDFT_LIB=true
##  Build the 3D DFT (real to complex, complex to real) library
MDPRDFT_LIB=false
##  Build the Real Convolution library
RCONV_LIB=false
##Build FFTX for CPU
BUILD_FOR_CPU=false
##  Build FFTX for CUDA
BUILD_FOR_CUDA=false
##  Build FFTX for HIP
BUILD_FOR_HIP=true
```

Build FFTX

```
git clone --branch ecp-tutorial-2023 https://github.com/spiral-software/fftx
pushd fftx
export FFTX_HOME=`pwd`          # could also be FFTX_HOME=`pwd`/HIP

./config-fftx-libs.sh           # this will be quick (just 2 very small libraries); perhaps much longer if you build many libraries and sizes

mkdir build
cd build
cmake -DCMAKE_INSTALL_PREFIX=$FFTX_HOME -D_codegen=HIP -DCMAKE_CXX_COMPILER=hipcc ..
make install -j
cd ..
#               directories lib, bin, include, and CMakeIncludes will be populated in the location specified as CMAKE_INSTALL_PREFIX
```

NOTE: FFTX_HOME is not required to be set in order to **build** FFTX. It is used in order to know where the libraries and include files may be found. Thus, it should be set to the install location (not necessarily the root of the fftx tree).

NOTE: $FFTX_HOME/include will contain all public header files from FFTX
$FFTX_HOME/lib will contain all libraries built by FFTX

Libraries Built

The following libraries are created during the build process:

Contents of $FFTX_HOME/lib:

libfftx_mpi.so	Support library for distributed FFTs (always built)
libfftx_mddft_gpu.so	Our small MDDFT [fixed sizes] library
libfftx_imddft_gpu.so	Our small inverse MDDFT [fixed sizes] library

Other libraries are built *depending on the configuration settings defined in config-fftx-libs.sh*
Libraries may be built for:

Multi-dimension (3) real-to-complex and complex-to-real packed DFT, (MDPRDFT and IMDPRDFT)
Multi-dimension (3) real convolution
Batch 1D DFT complex-to-complex, both forward and inverse
Batch 1D DFT real-to-complex and complex-to-real packed DFT, (PRDFT and IPRDFT)

FFTX has several test programs that exercise the libraries and compare their result with alternate methods of computation to demonstrate correctness (e.g., cufft or NumPy).

Transform Sizes in the Libraries

- Code generated by Spiral is optimized based on the problem size
- Each unique size needs to be generated and compiled (whether pre-compiled to a library or compiled on demand)
- Currently a set of fixed sizes is defined (see the cube-sizes files referenced in config-fftx-libs.sh). Additional sizes may be added as required to these files (if building pre-compiled libraries).
- Any supported size may be built at run time using the RTC mechanism.

Cube-sizes.txt:

```
##  Hash (#) as first non-white space indicates a comment
##  Lines containing white space only are ignored
##  All other lines must be valid size specs in the form:
##  szcube := [ x, y, z ]
##
szcube := [  48,  48,  48 ];
szcube := [  64,  64,  64 ];
szcube := [  80 , 80,  80 ];
```

Using CMake to Link Your Application to FFTX

- Spiral and FFTX are built using CMake. FFTX provides several CMake functions to simplify building both internal example programs and also to enable external applications to link with FFTX libraries and access the FFTX API.

- fftx-demo-extern-app is a small repo [external from FFTX] whose purpose is to demonstrate how to link an application with the FFTX libraries. It is available at:

https://github.com/spiral-software/fftx-demo-extern-app

Using CMake to find and link with FFTX

```
##  FFTX_HOME must be defined in the environment
if ( DEFINED ENV{FFTX_HOME} )
   message ( STATUS "FFTX_HOME = $ENV{FFTX_HOME}" )
   set ( FFTX_SOURCE_DIR $ENV{FFTX_HOME} )
else ()
   if ( "x${FFTX_HOME}" STREQUAL "x" )
      message ( FATAL_ERROR "FFTX_HOME environment variable undefined and not specified on command line" )
   endif ()
   set ( FFTX_SOURCE_DIR ${FFTX_HOME} )
endif ()
```

Using CMake to Link Your Application to FFTX

```
##  Include FFTX CMake functions
include ( "${FFTX_SOURCE_DIR}/CMakeIncludes/FFTXCmakeFunctions.cmake" )

##  FFTX_find_libraries () finds the FFTX libraries, paths, etc. and exposes the following variables:
##  FFTX_LIB_INCLUDE_PATHS -- include paths for FFTX include & library headers
##  FFTX_LIB_NAMES         -- list of FFTX libraries
##  FFTX_LIB_LIBRARY_PATH  -- path to libraries (for linker)
##
##  You don't need to call FFTX_find_libraries() directly unless you specifically want access to the variables listed above -- its
##  called as part of FFTX_add_includes_libs_to_target ( _target )

FFTX_find_libraries ()
message ( STATUS "Include paths:   ${FFTX_LIB_INCLUDE_PATHS}" )
message ( STATUS "Libraries found: ${FFTX_LIB_NAMES}" )
message ( STATUS "Library path is: ${FFTX_LIB_LIBRARY_PATH}" )

add_executable            ( ${PROJECT} ${PROJECT}.cpp )
set ( _targets ${PROJECT} )

foreach ( _targ ${_targets} )
   FFTX_add_includes_libs_to_target ( ${_targ} )
endforeach ()
```

Do I Have To Use CMake?

No…

But it might make life easier!

As mentioned earlier when FFTX is built it installs the artifacts in folders under $FFTX_HOME, as follows

$FFTX_HOME:

include/	All public header files
lib/	All libraries built
bin/	FFTX example program built
CMakeIncludes	CMake functions useful to simplify compile and linking with FFTX

If, for example, you want to use Make instead of CMake then you'll need to:

- Add -I switch for the include folder
- Add –L switch for the library path
- Add –l<library> for each library required for linking

NOTE: Library names will change based on the architecture for which you build (e.g., CPU or GPU)

Running Examples Created for The Tutorial

We just built some examples, they are all installed at $FFTX_HOME/bin

```
cd $FFTX_HOME/bin
./testmddft -h
##  Usage: ./testmddft: [ -i iterations ] [ -s MMxNNxKK ] [ -h (print help message) ]

./testmddft -i 5 -s 48x64x80

./testmdprdft -i 5 -s 48x64x80

./testrconv -i 5 -s 48x64x80
```

Simple Example – Basic Elements For Forward MDDFT

```
//  device runtimes & fftx headers – See example in ~/<fftx_home>/examples/mddft/simple_example.cpp
…
int main(int argc, char *argv[])
{
    int mm = 24, nn = 32, kk = 40;
    long arrsz = mm * nn * kk;
    double *hostX = new double[arrsz * 2];          // Allocate 2 * arrsz for complex; input
    double *hostY = new double[arrsz * 2];          // output
    hipDeviceptr_t  dY, dX, dsym;
    std::vector<int> sizes{ mm, nn, kk };
    buildInputBuffer (hostX, sizes);                // build a buffer of random values

    hipMalloc((void **)&dY, arrsz * sizeof(std::complex<double>));                  // Output [device]
    hipMalloc((void **)&dX, arrsz * sizeof(std::complex<double>));                  // Input [device]
    hipMemcpy(dX, hostX,  arrsz * sizeof(std::complex<double>), hipMemcpyHostToDevice);
    hipMalloc((void **)&dsym, 64* sizeof(double));
    MDDFTProblem mdp (std::vector<void*>{dY,dX,dsym}, sizes);
    mdp.transform();
    std::cout << mdp.getTime() << std::endl;

    hipMemcpy( hostY, dY,  arrsz * sizeof(std::complex<double>), hipMemcpyDeviceToHost);
    std::cout << "copied Y to host" << std::endl;

    delete[] hostX;
    delete[] hostY;
    return 0;
}
```

Simple Example – Compile & Run

Source code for the simple example may be found at ~/<fftx_home>/examples/mddft/simple_example.cpp

You can build the example by running the shell script compile_simple_example.sh:

```
#!/bin/bash
module load rocm/5.4.0
hipcc -I $FFTX_HOME/include/ -DFFTX_HIP simple_example.cpp
```

Then run the example:

```
$ ./a.out
Standalone test for size: 24x32x40
allocated Y
allocating memory
30720
allocated X
copied hostX
haven't seen size, generating

code for new size generated
0.064
haven't seen size, generating

code for new size generated
0.064
copied Y to host
Standalone test for size: 24x32x40 completed successfully
```

Other Library Information and Low-Level Library APIs

- We've primarily focused on HIP code generation for this tutorial. As noted, earlier Spiral also generates code for the CPU and CUDA (SYCL is not currently scheduled but can also be implemented)
- Each library (when building libraries) contains code for a single architecture (either CPU or GPU; GPU libraries contain code for a specific type – i.e., CUDA or HIP).
- Each generated library has a public header file:
- The public header files provide the minimum information needed to access and run the transforms from that library.

Library Name	Header File
libfftx_mddft_gpu.so	fftx_mddft_gpu_public.h
libfftx_mddft_cpu.so	fftx_mddft_cpu_public.h
libfftx_imddft_gpu.so	fftx_imddft_gpu_public.h
libfftx_imddft_cpu.so	fftx_imddft_cpu_public.h
libfftx_mdprdft_gpu.so	fftx_mdprdft_gpu_public.h
libfftx_mdprdft_cpu.so	fftx_mdprdft_cpu_public.h
libfftx_imdprdft_gpu.so	fftx_imdprdft_gpu_public.h
libfftx_imdprdft_cpu.so	fftx_imdprdft_cpu_public.h
libfftx_rconv_gpu.so	fftx_rconv_gpu_public.h
libfftx_rconv_cpu.so	fftx_rconv_cpu_public.h
libfftx_dftbat.so	fftx_dftbat_public.h
libfftx_idftbat.so	fftx_idftbat_public.h
libfftx_prdftbat.so	fftx_prdftbat_public.h
libfftx_iprdftbat.so	fftx_iprdftbat_public.h

External Demo App

- As mentioned earlier, fftx-demo-extern-app is a small repo [external from FFTX] whose purpose is to demonstrate how to link an application with the FFTX libraries. It is available at:
- https://github.com/spiral-software/fftx-demo-extern-app

This application contains:

- dftbatlib_test – a simple test program exercising all entries in the the DFT 1D libraries
- poissonTest – a small Poisson test with verification (for CPU only)
- transformlib_test – a small test exercising all entries in the **fftx_mddft** library and comparing the results obtained from FFTX/Spiral with cufft or rocfft (requires CUDA or HIP)
- perf_test_driver – a test excerising potentially all entries in the MDDFT and MDPRDFT libraries, comparing the results obtained from FFTX/Spiral with cufft or rocfft and providing performance times for each (requires CUDA or HIP)
- A complete **CMakeLists.txt** that checks for CUDA/HIP (won't attempt to build the GPU example on systems without GPU support), links against the FFTX libraries, etc.

Spack

- Initial Spack package definitions for FFTX and Spiral packages exist and have been merged into the 'develop' branch of Spack. These need to be updated to handle the additions for RTC; that work will be done later this Spring/Summer.