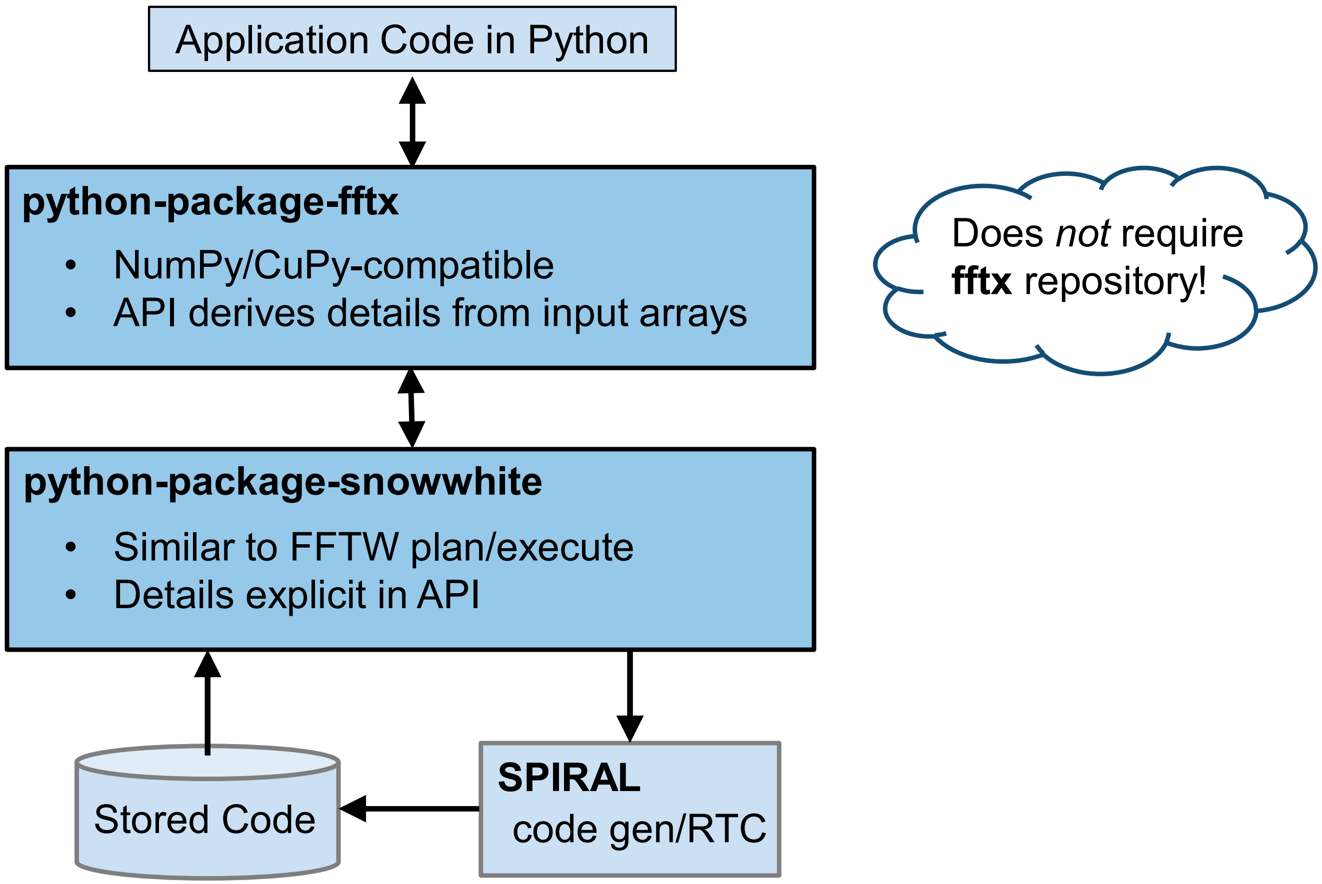
FFTX Python interface

Peter McCorquodale (LBNL)
ECP tutorial, February 7, 2023

EXASCALE COMPUTING PROJECT

Python packages and their relationships



Repositories at **`https://github.com/spiral-software`**

How to install the FFTX Python interface

Prerequisites:

- Have SPIRAL cloned and built in directory `$SPIRAL_HOME`
- Environment variable `PATH` needs to include `$SPIRAL_HOME/bin`
- Python library NumPy (on CPU) or CuPy (on GPU)

In a new directory, say `spiral-python-home`, do:

- `git clone https://github.com/spiral-software/python-package-snowwhite snowwhite`
- `git clone https://github.com/spiral-software/python-package-fftx fftx`

Set environment variable `PYTHONPATH` to include the directory `spiral-python-home` where you put `snowwhite` and `fftx`.

Then in Python, just do: `import fftx`

Functions in FFTX module `fftx.fft`

- Each function in FFTX module `fftx.fft` takes a NumPy or CuPy array as input, either double or single precision, and returns an array of same class and precision.
- Same basic interfaces as NumPy module `numpy.fft` and CuPy module `cupy.fft`.
- 1-dimensional transforms on NumPy/CuPy 1D array `src`:

Function	input array type (`src`)	output array type
`fftx.fft.fft(src)`	complex	complex
`fftx.fft.ifft(src)`	complex	complex

- N-dimensional transforms on NumPy/CuPy multi-D array `src`:

Function	input array type (`src`)	output array type
`fftx.fft.fftn(src)`	complex	complex
`fftx.fft.ifftn(src)`	complex	complex
`fftx.fft.rfftn(src)`	real	complex
`fftx.fft.irfftn(src)`	complex	real

Calling functions in the `fftx.fft` module

- First time you call a particular function on a particular array size and precision,
  - SPIRAL is run and (after some time) generates C or CUDA or HIP code;
  - code is compiled into a library file saved in `snowwhite/.libs` subdirectory.
- Subsequent calls to the same function on arrays of the same size and precision invoke the saved library.

Calling functions in the `fftx.fft` module

- First time you call a particular function on a particular array size and precision,
  - SPIRAL is run and (after some time) generates C or CUDA or HIP code;
  - code is compiled into a library file saved in `snowwhite/.libs` subdirectory.
- Subsequent calls to the same function on arrays of the same size and precision invoke the saved library.

Currently, SPIRAL fails on some input sizes, but these ones up to 320 work:

- 1D `fft`/`ifft` on lengths with all prime factors ≤13, except 13, 26, 243, 245, 297;
- 3D `fftn`/`ifftn` cubes with all prime factors ≤17, except 4, 169, 187, 221, 289;
- 3D `rfftn`/`irfftn` cubes with all prime factors ≤17, except 4.

If you use an array size that doesn't work, let us know and we'll prioritize fixing it.

Demonstration on crusher

CuPy is not already on crusher, but you can install it. Here's one way:

```
module load cray-python
module load rocm
export CUPY_INSTALL_USE_HIP=1
export ROCM_HOME=$ROCM_PATH
export HCC_AMDGPU_TARGET=gfx90a
pip install --no-cache-dir cupy
```

The `pip install` can take 10 minutes.

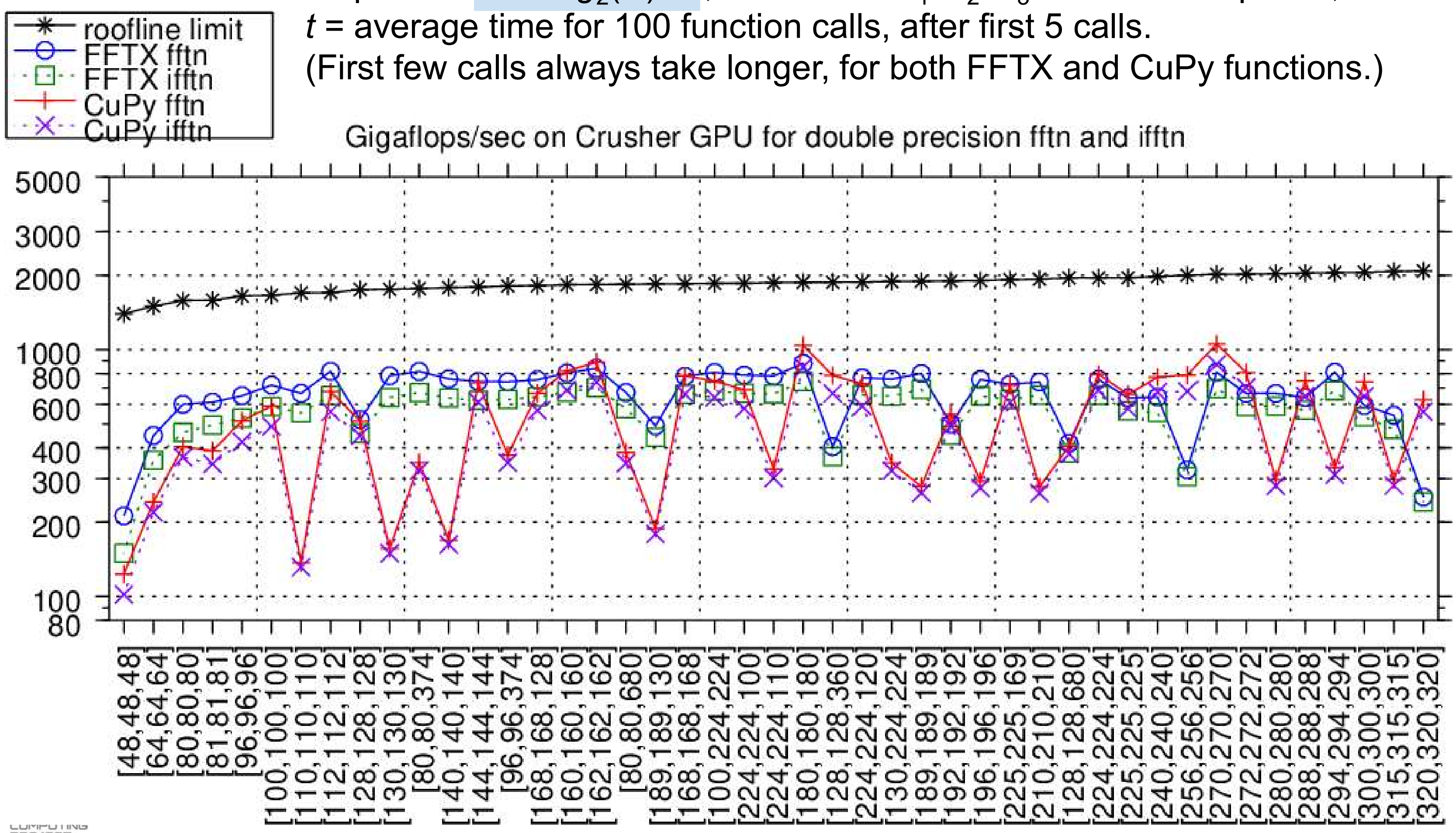
Instead of CuPy on GPU, you can use NumPy on CPU.

Acknowledgement

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

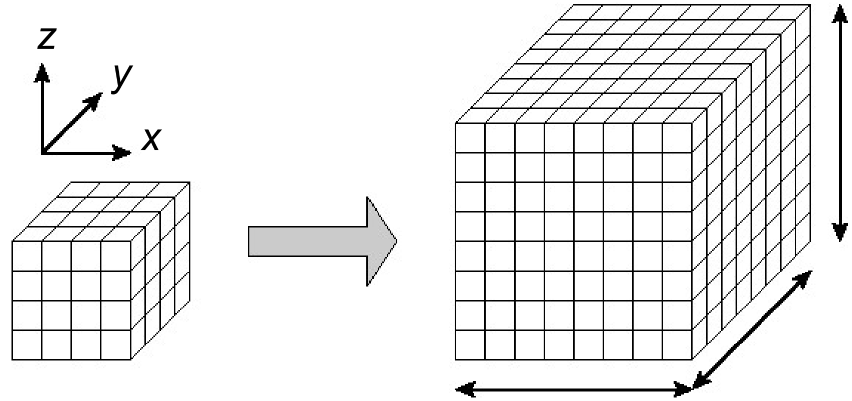
Flop rates for `fft.fftn`/`ifftn`, FFTX vs. CuPy

Flop rate = $5 \cdot N \cdot log_2(N) / t$, where $N = N_1 \cdot N_2 \cdot N_3$ is number of points, and t = average time for 100 function calls, after first 5 calls.
(First few calls always take longer, for both FFTX and CuPy functions.)



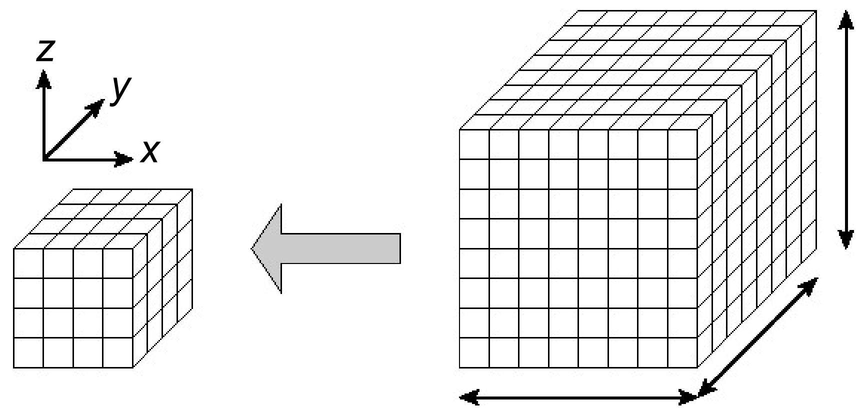
Free-space convolution calling “black box” 3D FFTs

1. Pad M x M x M input with zeros to 2M x 2M x 2M array, then do forward 3D FFT of size 2M x 2M x 2M.



2. Multiply 2M x 2M x 2M array by symbol array

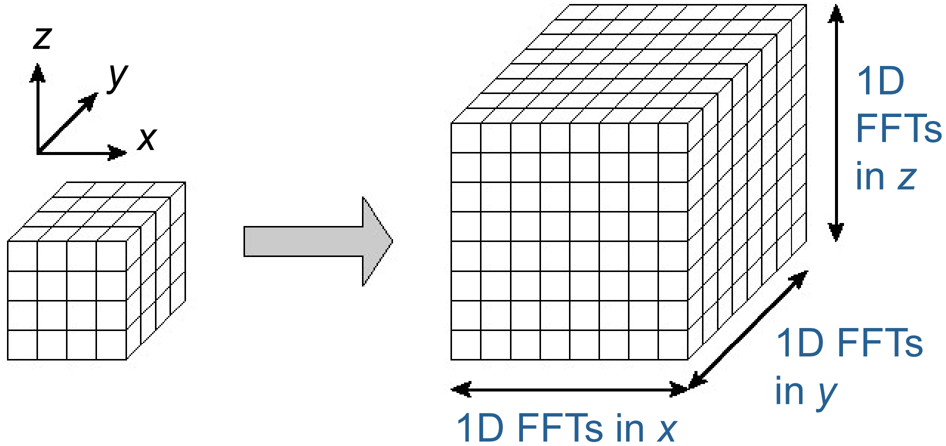
3. Do inverse 3D FFT of size 2M x 2M x 2M, then prune to M x M x M output.



Free-space convolution calling "black box" 3D FFTs

1. Pad M x M x M input with zeros to 2M x 2M x 2M array, then do <u>forward 3D FFT of size 2M x 2M x 2M</u>.

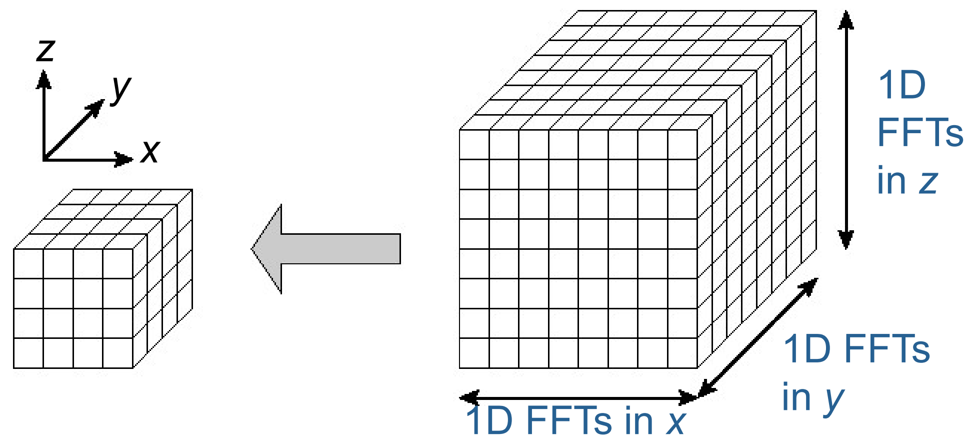
$3 \cdot (2M)^2$ forward 1D FFTs of length 2M



2. Multiply 2M x 2M x 2M array by symbol array

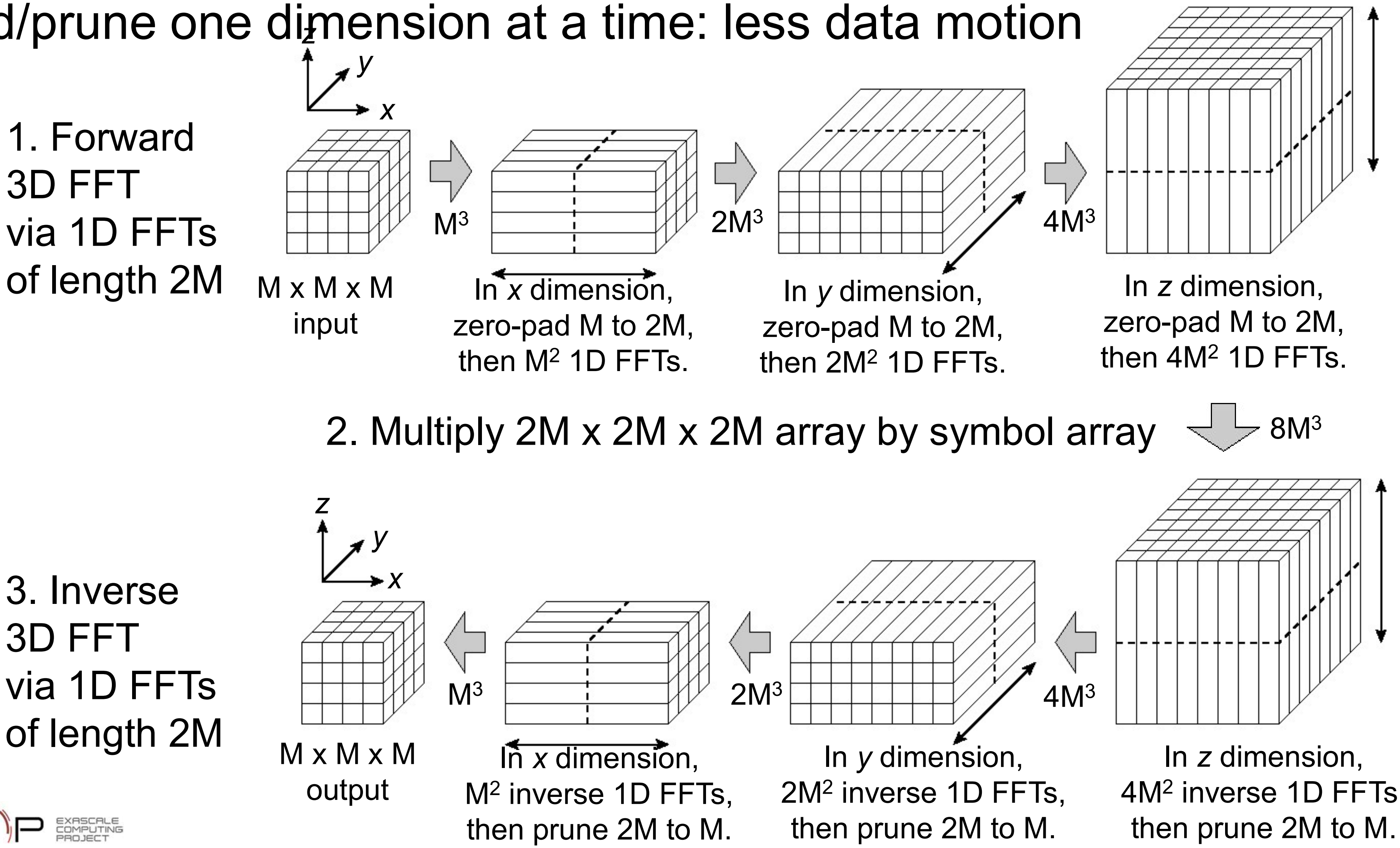
$3 \cdot (2M)^2$ inverse 1D FFTs of length 2M

3. Do <u>inverse 3D FFT of size 2M x 2M x 2M</u>, then prune to M x M x M output.

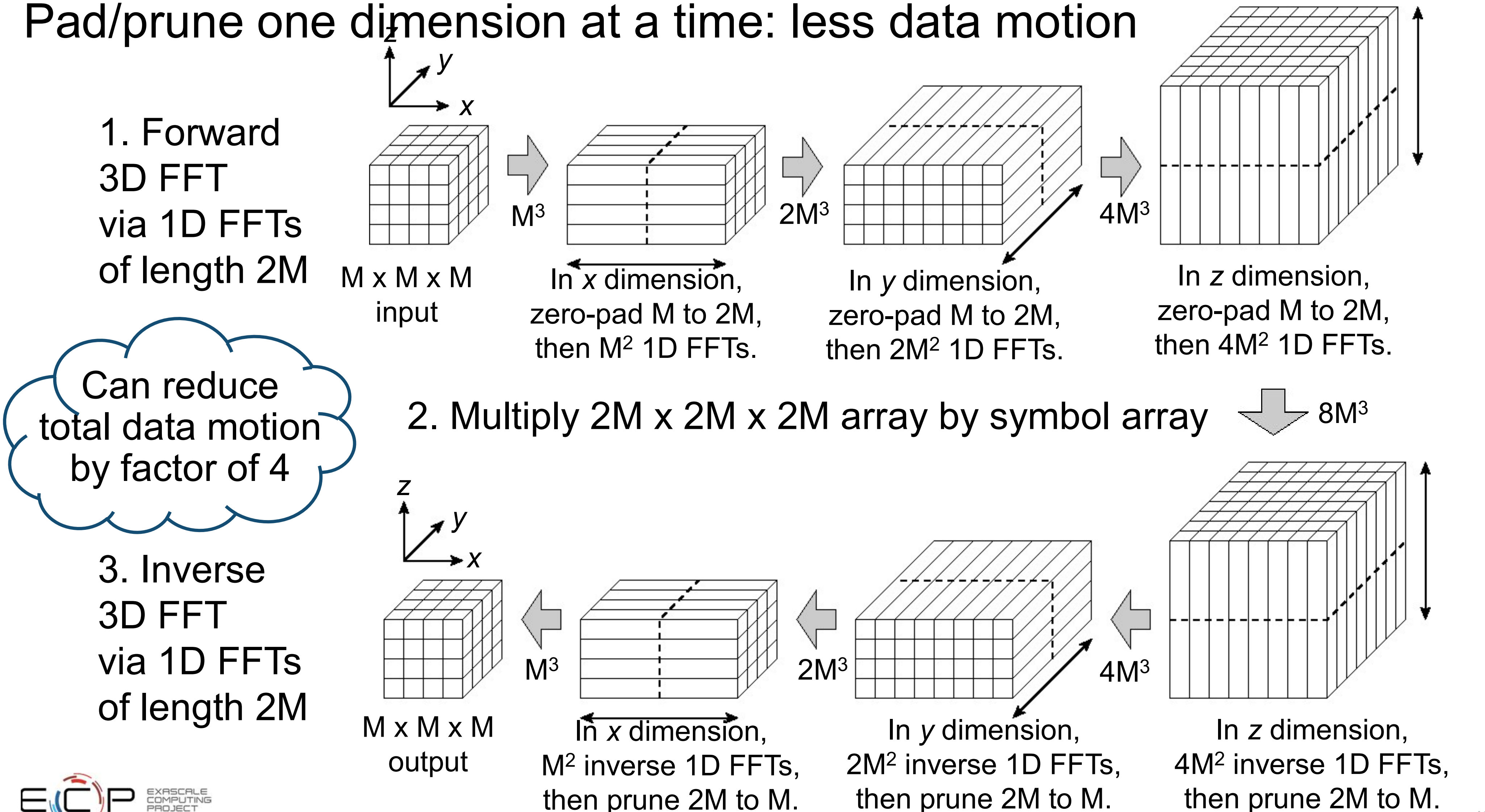


EXASCALE COMPUTING PROJECT

Pad/prune one dimension at a time: less data motion



Pad/prune one dimension at a time: less data motion



Implementation of integrated algorithm for free-space 3D convolution kernel in Spinifel

Original Spinifel:

```
ugrid_ups = cupy.zeros((2*M,)*3,
    dtype=uvect.dtype)

ugrid_ups[:M, :M, :M] = ugrid

F_ugrid_ups = cupy.fft.fftn(
    cupy.fft.ifftshift(ugrid_ups))

F_ugrid_conv_out_ups = F_ugrid_ups *
    F_ugrid_conv_

ugrid_conv_out_ups =
    cupy.fft.fftshift(cupy.fft.ifftn(
        F_ugrid_conv_out_ups))

ugrid_conv_out =
    ugrid_conv_out_ups[:M, :M, :M]
```

FFTX replacement:

```
ugrid_conv_out =
    fftx.convo.mdrfsconv(ugrid,
        F_ugrid_conv_)
```

The `fftx.convo` module is in the **python-package-fftx** repo.

Implementation of integrated algorithm for free-space 3D convolution kernel in Spinifel

Original Spinifel:

```
ugrid_ups = cupy.zeros((2*M,)*3,
    dtype=uvect.dtype)

ugrid_ups[:M, :M, :M] = ugrid

F_ugrid_ups = cupy.fft.fftn(
    cupy.fft.ifftshift(ugrid_ups))

F_ugrid_conv_out_ups = F_ugrid_ups *
    F_ugrid_conv_

ugrid_conv_out_ups =
    cupy.fft.fftshift(cupy.fft.ifftn(
        F_ugrid_conv_out_ups))

ugrid_conv_out =
    ugrid_conv_out_ups[:M, :M, :M]
```

FFTX replacement:

```
ugrid_conv_out =
    fftx.convo.mdrfsconv(ugrid,
        F_ugrid_conv_)
```

The `fftx.convo` module is in the **python-package-fftx** repo.

Implementation of integrated algorithm for free-space 3D convolution kernel in Spinifel

Original Spinifel:

```
ugrid_ups = cupy.zeros((2*M,)*3,
    dtype=uvect.dtype)

ugrid_ups[:M, :M, :M] = ugrid

F_ugrid_ups = cupy.fft.fftn(
    cupy.fft.ifftshift(ugrid_ups))

F_ugrid_conv_out_ups = F_ugrid_ups *
    F_ugrid_conv_

ugrid_conv_out_ups =
    cupy.fft.fftshift(cupy.fft.ifftn(
        F_ugrid_conv_out_ups))

ugrid_conv_out =
    ugrid_conv_out_ups[:M, :M, :M]
```

FFTX replacement:

```
ugrid_conv_out =
    fftx.convo.mdrfsconv(ugrid,
        F_ugrid_conv_)
```

The `fftx.convo` module is in the **python-package-fftx** repo.

Rather than simply replacing calls to `cupy.fft` with calls to `fftx.fft`, SPIRAL operates on an *integrated algorithm*, optimizing over whole kernel.

Implementation of integrated algorithm for free-space 3D convolution kernel in Spinifel

Original Spinifel:

```
ugrid_ups = cupy.zeros((2*M,)*3,
    dtype=uvect.dtype)

ugrid_ups[:M, :M, :M] = ugrid

F_ugrid_ups = cupy.fft.fftn(
    cupy.fft.ifftshift(ugrid_ups))

F_ugrid_conv_out_ups = F_ugrid_ups *
    F_ugrid_conv_

ugrid_conv_out_ups =
    cupy.fft.fftshift(cupy.fft.ifftn(
        F_ugrid_conv_out_ups))

ugrid_conv_out =
    ugrid_conv_out_ups[:M, :M, :M]
```

FFTX replacement:

```
ugrid_conv_out =
    fftx.convo.mdrfsconv(ugrid,
        F_ugrid_conv_)
```

The `fftx.convo` module is in the **python-package-fftx** repo.

Rather than simply replacing calls to `cupy.fft` with calls to `fftx.fft`, SPIRAL operates on an *integrated algorithm*, optimizing over whole kernel.

FFTX integrated algorithm for this kernel gives **4.3x** speedup over original Spinifel with CuPy when running on crusher @ OLCF.

Inside `fftx.convo.mdrfsconv`

`fftx/convo.py`:

```
try:
    import cupy as cp
import snowwhite as sw
from snowwhite.mdrfsconvsolver import *
…
def mdrfsconv(src, symbol, dst=None):
    global _solver_cache
    platform = SW_CPU
    if sw.get_array_module(src) == cp:
        platform = SW_HIP if sw.has_ROCm()
else SW_CUDA
    opts = { SW_OPT_PLATFORM : platform }
    N = list(src.shape)[1]
    t = 'd'
    if src.dtype.name == 'float32':
        opts[SW_OPT_REALCTYPE] = 'float'
        t = 's'
    ckey = t + '_mdrfsconv_' + str(N)
    solver = _solver_cache.get(ckey, 0)
    if solver == 0:
        problem = MdrfsconvProblem(N)
        solver  = MdrfsconvSolver(problem, opts)
        _solver_cache[ckey] = solver
    result = solver.solve(src, symbol, dst)
    return result
…
```

`snowwhite/mdrfsconvsolver.py`:

```
…
        print("Load(fftx);", file = script_file)
        print("ImportAll(fftx);", file = script_file)
        print("", file = script_file)
        if self._genCuda:
            print("conf := LocalConfig.fftx.confGPU();", file = script_file)
        elif self._genHIP:
            print ( 'conf := FFTXGlobals.defaultHIPConf();', file = script_file )
        else:
            print("conf := LocalConfig.fftx.defaultConf();", file = script_file)
        print("", file = script_file)
        print('t := let(symvar := var("sym", TPtr(TReal)),', file = script_file)
        print("    TFCall(", file = script_file)
        print("        Compose([", file = script_file)
        for i in range(len(self._callGraph)):
            print("            " + self._callGraph[i], file = script_file)
        print("        ]),", file = script_file)
        print('        rec(fname := "' + nameroot + '", params := [symvar])',
          file = script_file)
        print("    )", file = script_file)
        print(");", file = script_file)
        print("", file = script_file)
        print("opts := conf.getOpts(t);", file = script_file)
…
```

Inside `fftx.convo.mdrfsconv`

`fftx/convo.py`:

```
try:
    import cupy as cp
import snowwhite as sw
from snowwhite.mdrfsconvsolver import *
…
def mdrfsconv(src, symbol, dst=None):
    global _solver_cache
    platform = SW_CPU
    if sw.get_array_module(src) == cp:
        platform = SW_HIP if sw.has_ROCm()
else SW_CUDA
    opts = { SW_OPT_PLATFORM : platform }
    N = list(src.shape)[1]
    t = 'd'
    if src.dtype.name == 'float32':
        opts[SW_OPT_REALCTYPE] = 'float'
        t = 's'
    ckey = t + '_mdrfsconv_' + str(N)
    solver = _solver_cache.get(ckey, 0)
    if solver == 0:
        problem = MdrfsconvProblem(N)
        solver  = MdrfsconvSolver(problem, opts)
        _solver_cache[ckey] = solver
    result = solver.solve(src, symbol, dst)
    return result
…
```

`snowwhite/mdrfsconvsolver.py`:

```
…
    print("Load(fftx);", file = script_file)
    print("ImportAll(fftx);", file = script_file)
    print("", file = script_file)
    if self._genCuda:
        print("conf := LocalConfig.fftx.confGPU();", file = script_file)
    elif self._genHIP:
        print ( 'conf := FFTXGlobals.defaultHIPConf();', file = script_file )
    else:
        print("conf := LocalConfig.fftx.defaultConf();", file = script_file)
    print("", file = script_file)
    print('t := let(symvar := var("sym", TPtr(TReal)),', file = script_file)
    print("    TFCall(", file = script_file)
    print("        Compose([", file = script_file)
    for i in range(len(self._callGraph)):
        print("            " + self._callGraph[i], file = script_file)
    print("        ]),", file = script_file)
    print('        rec(fname := "' + nameroot + '", params := [symvar])',
       file = script_file)
    print("    )", file = script_file)
    print(");", file = script_file)
    print("", file = script_file)
    print("opts := conf.getOpts(t);", file = script_file)
…
```

`fftx.conv.mdrfsconv` on array of size 81x81x81 writes out SPIRAL script:

`dMdrfsconv_81x81x81_hip.g`:

```
Load(fftx);
ImportAll(fftx);
conf := FFTXGlobals.defaultHIPConf();
t := let(symvar := var("sym", TPtr(TReal)),
    TFCall(
        Compose([
            ExtractBox([162,162,162], [[81..161],[81..161],[81..161]]),
            IMDPRDFT([162,162,162], 1),
            RCDiag(FDataOfs(symvar, 4304016, 0)),
            MDPRDFT([162,162,162], -1),
            ZeroEmbedBox([162,162,162], [[81..161],[81..161],[81..161]])
        ]),
        rec(fname := "dMdrfsconv_81x81x81_hip", params := [symvar])
    )
);
opts := conf.getOpts(t);
opts.wrapCFuncs := true;
tt := opts.tagIt(t);
c := opts.fftxGen(tt);
PrintTo("dMdrfsconv_81x81x81_hip.cpp", opts.prettyPrint(c));
```

SPIRAL reads this, writes out HIP code.

EXASCALE COMPUTING PROJECT

Inside `fftx.convo.mdrfsconv`

`fftx/convo.py`:

```
try:
    import cupy as cp
import snowwhite as sw
from snowwhite.mdrfsconvsolver import *
…
def mdrfsconv(src, symbol, dst=None):
    global _solver_cache
    platform = SW_CPU
    if sw.get_array_module(src) == cp:
        platform = SW_HIP if sw.has_ROCm()
else SW_CUDA
    opts = { SW_OPT_PLATFORM : platform }
    N = list(src.shape)[1]
    t = 'd'
    if src.dtype.name == 'float32':
        opts[SW_OPT_REALCTYPE] = 'float'
        t = 's'
    ckey = t + '_mdrfsconv_' + str(N)
    solver = _solver_cache.get(ckey, 0)
    if solver == 0:
        problem = MdrfsconvProblem(N)
        solver  = MdrfsconvSolver(problem, opts)
        _solver_cache[ckey] = solver
    result = solver.solve(src, symbol, dst)
    return result
…
```

`snowwhite/mdrfsconvsolver.py`:

```
…
    print("Load(fftx);", file = script_file)
    print("ImportAll(fftx);", file = script_file)
    print("", file = script_file)
    if self._genCuda:
        print("conf := LocalConfig.fftx.confGPU();", file = script_file)
    elif self._genHIP:
        print ( 'conf := FFTXGlobals.defaultHIPConf();', file = script_file )
    else:
        print("conf := LocalConfig.fftx.defaultConf();", file = script_file)
    print("", file = script_file)
    print('t := let(symvar := var("sym", TPtr(TReal)),', file = script_file)
    print("    TFCall(", file = script_file)
    print("        Compose([", file = script_file)
    for i in range(len(self._callGraph)):
        print("            " + self._callGraph[i], file = script_file)
    print("        ]),", file = script_file)
    print('        rec(fname := "' + nameroot + '", params := [symvar])',
      file = script_file)
    print("    )", file = script_file)
    print(");", file = script_file)
    print("", file = script_file)
    print("opts := conf.getOpts(t);", file = scrip
…
```

`fftx.conv.mdrfsconv`
on array of size 81x81x81
writes out SPIRAL script:

`dMdrfsconv_81x81x81_hip.g`:

```
Load(fftx);
ImportAll(fftx);
conf := FFTXGlobals.defaultHIPConf();
t := let(symvar := var("sym", TPtr(TReal)),
    TFCall(
        Compose([
            ExtractBox([162,162,162], [[81..161],[81..161],[81..161]]),
            IMDPRDFT([162,162,162], 1),
            RCDiag(FDataOfs(symvar, 4304016, 0)),
            MDPRDFT([162,162,162], -1),
            ZeroEmbedBox([162,162,162], [[81..161],[81..161],[81..161]])
        ]),
        rec(fname := "dMdrfsconv_81x81x81_hip", params := [symvar])
    )
);
opts := conf.getOpts(t);
opts.wrapCFuncs := true;
tt := opts.tagIt(t);
c := opts.fftxGen(tt);
PrintTo("dMdrfsconv_81x81x81_hip.cpp", opts.prettyPrint(c));
```

prune
inverse 3D FFT
multiply by array
forward 3D FFT
zero-pad

SPIRAL reads this, writes out HIP code.

Inside `fftx.convo.mdrfsconv`

`fftx/convo.py`:

```
try:
   import cupy as cp
import snowwhite as sw
from snowwhite.mdrfsconvsolver import *
…
def mdrfsconv(src, symbol, dst=None):
   global _solver_cache
   platform = SW_CPU
   if sw.get_array_module(src) == cp:
      platform = SW_HIP if sw.has_ROCm()
else SW_CUDA
   opts = { SW_OPT_PLATFORM : platform }
   N = list(src.shape)[1]
   t = 'd'
   if src.dtype.name == 'float32':
      opts[SW_OPT_REALCTYPE] = 'float'
      t = 's'
   ckey = t + '_mdrfsconv_' + str(N)
   solver = _solver_cache.get(ckey, 0)
   if solver == 0:
      problem = MdrfsconvProblem(N)
      solver  = MdrfsconvSolver(problem, opts)
      _solver_cache[ckey] = solver
   result = solver.solve(src, symbol, dst)
   return result
…
```

`snowwhite/mdrfsconvsolver.py`:

```
…
    print("Load(fftx);", file = script_file)
    print("ImportAll(fftx);", file = script_file)
    print("", file = script_file)
    if self._genCuda:
       print("conf := LocalConfig.fftx.confGPU();", file = script_file)
    elif self._genHIP:
       print ( 'conf := FFTXGlobals.defaultHIPConf();', file = script_file )
    else:
       print("conf := LocalConfig.fftx.defaultConf();", file = script_file)
    print("", file = script_file)
    print('t := let(symvar := var("sym", TPtr(TReal)),', file = script_file)
    print("    TFCall(", file = script_file)
    print("        Compose([", file = script_file)
    for i in range(len(self._callGraph)):
       print("            " + self._callGraph[i], file = script_file)
    print("        ]),", file = script_file)
    print('        rec(fname := "' + nameroot + '", params := [symvar])',
      file = script_file)
    print("    )", file = script_file)
    print(");", file = script_file)
    print("", file = script_file)
    print("opts := conf.getOpts(t);", file = scrip
…
```

prune
inverse 3D FFT
multiply by array
forward 3D FFT
zero-pad

`fftx.conv.mdrfsconv`
on array of size 81x81x81
writes out SPIRAL script:

`dMdrfsconv_81x81x81_hip.g`:

```
Load(fftx);
ImportAll(fftx);
conf := FFTXGlobals.defaultHIPConf();
t := let(symvar := var("sym", TPtr(TReal)),
   TFCall(
      Compose([
         ExtractBox([162,162,162], [[81..161],[81..161],[81..161]]),
         IMDPRDFT([162,162,162], 1),
         RCDiag(FDataOfs(symvar, 4304016, 0)),
         MDPRDFT([162,162,162], -1),
         ZeroEmbedBox([162,162,162], [[81..161],[81..161],[81..161]])
      ]),
      rec(fname := "dMdrfsconv_81x81x81_hip", params := [symvar])
   )
);
opts := conf.getOpts(t);
opts.wrapCFuncs := true;
tt := opts.tagIt(t);
c := opts.fftxGen(tt);
PrintTo("dMdrfsconv_81x81x81_hip.cpp", opts.prettyPrint(c));
```

Implementing an integrated algorithm in FFTX requires SPIRAL expertise.
Talk to us if you're interested in such an implementation for kernels in your application.

SPIRAL reads this, writes out HIP code.

Reminder: where to get the FFTX Python interface

Repositories and installation instructions here:

- `https://github.com/spiral-software/python-package-snowwhite`
- `https://github.com/spiral-software/python-package-fftx`

Acknowledgements

- This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

Integrated algorithm reduces traffic by a factor of 4

Free-space convolution using black-box 3D FFT for size 2M x 2M x 2M:

	flops	reads	writes
TOTAL	**24**M^2∗FFT1D(2M) + 8M^3∗c	**64**M^3	**56**M^3

Integrated algorithm combining 1D FFTs in *z* with multiplication by symbol:

	flops	reads	writes
Forward 1D FFT in *x* Forward 1D FFT in *y*	M^2∗FFT1D(2M) 2M^2∗FFT1D(2M)	M^3 2M^3	2M^3 4M^3
1D FFTs in *z*, and symbol	2∗4M^2∗FFT1D(2M) + $(2M)^3$∗c	4M^3 + M^3	4M^3
Inverse 1D FFT in *y* Inverse 1D FFT in *x*	2M^2∗FFT1D(2M) M^2∗FFT1D(2M)	4M^3 2M^3	2M^3 M^3
TOTAL	**14**M^2∗FFT1D(2M) + 8M^3∗c	**14**M^3	**13**M^3

*Demo on crusher (**this will not be a slide!**)*

```
mkdir spiral-python-home
pushd spiral-python-home
git clone https://github.com/spiral-software/python-package-snowwhite.git snowwhite
git clone https://github.com/spiral-software/python-package-fftx.git fftx
pwd
ls
popd

export PYTHONPATH=$PYTHONPATH:/ccs/home/petermc/spiral-python-home
export SPIRAL_HOME=/ccs/home/petermc/spiral-software
export PATH=$SPIRAL_HOME/bin:$PATH
which spiral

mkdir demo
pushd demo

python
import fftx
import cupy
N = 64; dims = tuple([N, N, N])
```

*Continuation of demo on crusher (**not a slide!**)*

```
arr = cupy.random.random(dims) + 1j*cupy.random.random(dims)
farr_cupy = cupy.fft.fftn(arr)
farr_fftx = fftx.fft.fftn(arr)
cupy.max(abs(farr_cupy - farr_fftx))
cupy.max(abs(farr_cupy - farr_fftx)) / cupy.max(abs(farr_cupy))
# Again on a new array of the same size; much faster.
arr = cupy.random.random(dims) + 1j*cupy.random.random(dims)
farr_cupy = cupy.fft.fftn(arr)
farr_fftx = fftx.fft.fftn(arr)
cupy.max(abs(farr_cupy - farr_fftx)) / cupy.max(abs(farr_cupy))

# Run python again to show that the size is remembered.
python
import fftx
import cupy
N = 64; dims = tuple([N, N, N])
arr = cupy.random.random(dims) + 1j*cupy.random.random(dims)
farr_cupy = cupy.fft.fftn(arr)
farr_fftx = fftx.fft.fftn(arr)
cupy.max(abs(farr_cupy - farr_fftx))
```

*Continuation of demo on crusher (**not a slide!**)*

```
pushd ~/spiral-python-home
pushd fftx/examples/
python time-fftn.py
python time-fftn.py 64,64,64 F d GPU
python time-fftn.py 64,64,64 F d GPU
```