RAPIDS

cuSignal - GPU Accelerating SciPy Signal with Numba and CuPy

Adam Thompson | Senior Solutions Architect | adamt@nvidia.com
@adamlikesai

Contributors: Matthew Nicely, Graham Markall, Brad Rees

SciPy Signal – Polyphase Resampler

```
import numpy as np
from scipy import signal

start = 0
stop = 10
num_samps = int(1e8)
resample_up = 2
resample_down = 3

cx = np.linspace(start, stop, num_samps, endpoint=False)
cy = np.cos(-cx**2/6.0)

%%timeit
cf = signal.resample_poly(cy, resample_up, resample_down, window=('kaiser', 0.5))
```

2x Xeon E5-2600: 2.36 seconds

cuSignal – Polyphase Resampler

```
import cupy as cp
import cusignal

# Optional: Precompile custom CUDA kernels to eliminate JIT overhead on first run
cusignal.precompile_kernels()

start = 0
stop = 10
num_samps = int(1e8)
resample_up = 2
resample_down = 3

cx = cp.linspace(start, stop, num_samps, endpoint=False)
cy = cp.cos(-cx**2/6.0)

%%timeit
cf = cusignal.resample_poly(cy, resample_up, resample_down, window=('kaiser', 0.5))
```

NVIDIA V100: 8.29 milliseconds, **285x SciPy Signal**

Minimal API Changes, Same Results

```
import cupy as cp
import cusignal

# Optional: Precompile custom CUDA kernels to eliminate JIT overhead on first run
cusignal.precompile_kernels()

start = 0
stop = 10
num_samps = int(1e8)
resample_up = 2
resample_down = 3

cx = cp.linspace(start, stop, num_samps, endpoint=False)
cy = cp.cos(-cx**2/6.0)

%%timeit
cf = cusignal.resample_poly(cy, resample_up, resample_down, window=('kaiser', 0.5))
```

Mean Squared Error between cuSignal and SciPy Signal = 2.297e-31

Speed of Light Performance – V100

timeit (7 runs); Benchmarked with ~1e8 sample signals, float64

Method	Scipy Signal (ms)	cuSignal (ms)	Speedup (xN)
fftconvolve	27300	85.1	320.8
correlate	4020	47.4	84.8
resample	14700	45.9	320.2
resample_poly	2360	8.29	284.6
welch	4870	45.5	107.0
spectrogram	2520	23.3	108.1
convolve2d	8410	9.92	847.7

Learn more about cuSignal functionality and performance by browsing the notebooks

cuSignal – Selected Algorithms

GPU-accelerated SciPy Signal

Convolution
Convolve/Correlate
FFT Convolve
Convolve/Correlate 2D

Filtering and Filter Design
Resampling – Polyphase, Upfirdn, Resample
Hilbert/Hilbert 2D
Wiener
Firwin

Waveform Generation
Chirp
Square
Gaussian Pulse

Window Functions
Kaiser
Blackman
Hamming
Hanning

Spectral Analysis
Periodogram
Welch
Spectrogram

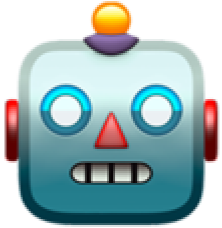
Wavelets

Peak Finding

More to come!

Signal Processing + GPUs: Two Fundamental Needs

Fast filtering, FFTs, correlations, convolutions, resampling, etc to process increasingly larger bandwidths of signals at increasingly fast rates and do increasingly cool stuff we couldn't do before



Artificial Intelligence techniques applied to spectrum sensing, signal identification, spectrum collaboration, and anomaly detection

Enabling Online Signal Processing

```
import cusignal
import numpy as np

num_samps = int(1e8)

# Create shared memory between CPU and GPU; similar to np.empty
sig = cusignal.get_shared_mem(num_samps, dtype=np.float64)

print('GPU Pointer: ', sig.__cuda_array_interface__['data'])
print('CPU Pointer: ', sig.__array_interface__['data'])

>>> GPU Pointer:  (47394527903744, False)
>>> CPU Pointer:  (47394527903744, False)
```

`cusignal.get_shared_mem` wraps Numba's `mapped_array` functionality within the CUDA module and enables a direct memory access between CPU and GPU, reducing I/O overhead

cuSignal support of the `__cuda_array_interface__` allows for zero-copy data movement between compatible GPU libraries (CuPy, Numba, RAPIDS, PyTorch, etc)

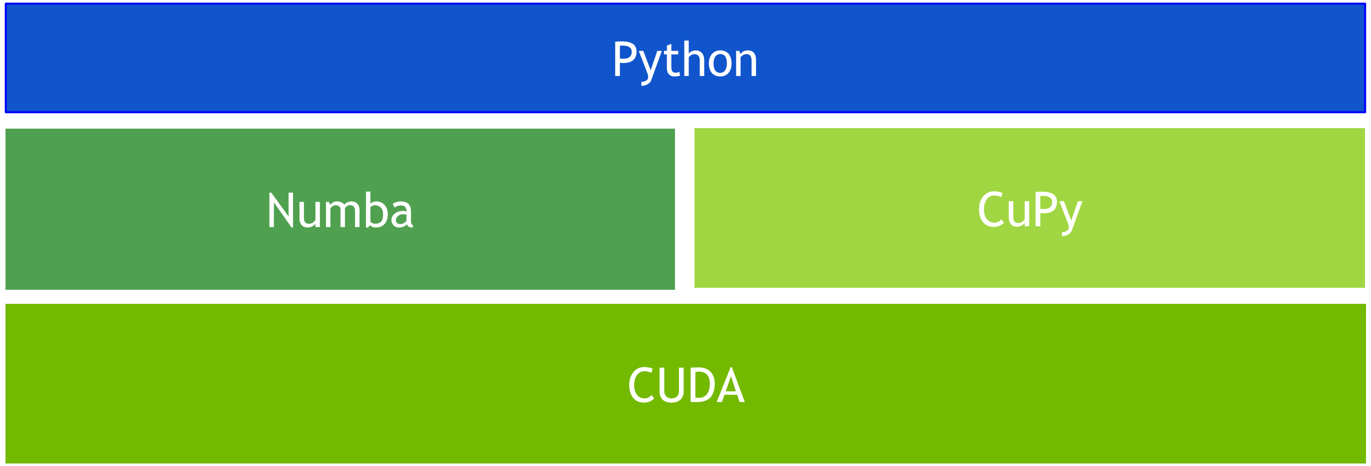
Deepwave Digital exhibits real-time signal collection and polyphase resampling at 62.5 MSps with cuSignal and their AIR-T Software Defined Radio platform that includes an NVIDIA Jetson TX2

Zero-Copy Connection to PyTorch

```
import cusignal
import cupy as cp
import torch

num_samps = int(1e8)

# Create shared memory between CPU and GPU; similar to np.empty
sig = cusignal.get_shared_mem(num_samps, dtype=cp.float64)
sig[:] = cp.random.randn(num_samps)

print('GPU Pointer: ', sig.__cuda_array_interface__['data'])

# Move to PyTorch
torch_sig = torch.as_tensor(sig, device='cuda')

print('Torch Pointer: ', torch_sig.__cuda_array_interface__['data'])

>>> GPU Pointer:  (47442242306048, False)
>>> Torch Pointer: (47442242306048, False)
```

cuSignal Technology Stack

An Innovative Approach to Pure Python GPU Library Development



Unlike other RAPIDS libraries, cuSignal is purely developed in Python
with custom CUDA Kernels written with Numba and CuPy (notice no Cython layer).

How We Built cuSignal (And How To Do It Yourself!)

Drop-in GPU Library Replacements

NumPy -> CuPy
Pandas -> cuDF
Scikit-Learn -> cuML
Network-X -> cuGraph

Pros:
Trivial code change
“Free” Performance

Cons:
Potentially sub-optimal
Limited control

Custom Numba CUDA Kernels

Leverage JIT compilation and Numba’s CUDA support to quickly build and test custom CUDA kernels with a Pythonic API

Pros:
Quickly build custom features
Boilerplate code

Cons:
JIT compilation overhead
Excess register pressure

Custom Raw CUDA Kernels

To match native CUDA speeds, wrap raw CUDA kernels in CuPy; precompile and cache kernel to avoid JIT overhead

Pros:
Matches CUDA C++ speed
No excess SW layer

Cons:
Limited debugging tools
Basically C/C++
Support multiple dtypes

Drop-in Replacements: 1D Hilbert Transform

SciPy Signal

```
import numpy as np
from scipy import fft as sp_fft

def hilbert(x, N=None, axis=-1):
    x = np.asarray(x)
    if np.iscomplexobj(x):
        raise ValueError("x must be real.")
    if N is None:
        N = x.shape[axis]
    if N <= 0:
        raise ValueError("N must be positive.")

    Xf = sp_fft.fft(x, N, axis=axis)
    h = np.zeros(N)
    if N % 2 == 0:
        h[0] = h[N // 2] = 1
        h[1:N // 2] = 2
    else:
        h[0] = 1
        h[1:(N + 1) // 2] = 2

    if x.ndim > 1:
        ind = [np.newaxis] * x.ndim
        ind[axis] = slice(None)
        h = h[tuple(ind)]
    x = sp_fft.ifft(Xf * h, axis=axis)
    return x
```

cuSignal

```
import cupy as cp
from cupy import fft as sp_fft

def hilbert(x, N=None, axis=-1):
    x = cp.asarray(x)
    if cp.iscomplexobj(x):
        raise ValueError("x must be real.")
    if N is None:
        N = x.shape[axis]
    if N <= 0:
        raise ValueError("N must be positive.")

    Xf = sp_fft.fft(x, N, axis=axis)
    h = cp.zeros(N)
    if N % 2 == 0:
        h[0] = h[N // 2] = 1
        h[1:N // 2] = 2
    else:
        h[0] = 1
        h[1:(N + 1) // 2] = 2

    if x.ndim > 1:
        ind = [cp.newaxis] * x.ndim
        ind[axis] = slice(None)
        h = h[tuple(ind)]
    x = sp_fft.ifft(Xf * h, axis=axis)
    return x
```

NVIDIA V100, 1e8 float64 samples, **230x speedup**

Custom Numba CUDA Kernels: Lombscargle

SciPy Signal (_spectral.pyx)

```
import numpy as np
cimport numpy as np
cimport cython

@cython.boundscheck(False)
def _lombscargle(np.ndarray[np.float64_t, ndim=1] x,
                np.ndarray[np.float64_t, ndim=1] y,
                np.ndarray[np.float64_t, ndim=1] freqs):

    for i in range(freqs.shape[0]):

        # Code not shown

        for j in range(x.shape[0]):

            # Code not shown

        pgram[i] = 0.5 * (((c_tau * xc + s_tau * xs)**2 /
            (c_tau2 * cc + cs_tau * cs + s_tau2 * ss)) +
            ((c_tau * xs - s_tau * xc)**2 /
            (c_tau2 * ss - cs_tau * cs + s_tau2 * cc)))
```

cuSignal

```
import cupy as cp
from numba import cuda

@cuda.jit(fastmath=True)
def _numba_lombscargle(x, y, freqs, pgram, y_dot):

    F = cuda.grid(1)
    strideF = cuda.gridsize(1)

    if not y_dot[0]:
        yD = 1.0
    else:
        yD = 2.0 / y_dot[0]

    for i in range(F, freqs.shape[0], strideF):

        # Code not shown

        for j in range(x.shape[0]):

            # Code not shown

        pgram[i] = 0.5 * (((c_tau * xc + s_tau * xs)**2 /
            (c_tau2 * cc + cs_tau * cs + s_tau2 * ss)) +
            ((c_tau * xs - s_tau * xc)**2 /
            (c_tau2 * ss - cs_tau * cs + s_tau2 * cc)))
```

NVIDIA V100, 1e4 float64 samples, **322x speedup**

Custom Raw CUDA Kernels: Lombscargle

SciPy Signal (_spectral.pyx)

```
import numpy as np
cimport numpy as np
cimport cython

@cython.boundscheck(False)
def _lombscargle(np.ndarray[np.float64_t, ndim=1] x,
                 np.ndarray[np.float64_t, ndim=1] y,
                 np.ndarray[np.float64_t, ndim=1] freqs):

    for i in range(freqs.shape[0]):

        # Code not shown

        for j in range(x.shape[0]):

            # Code not shown

        pgram[i] = 0.5 * (((c_tau * xc + s_tau * xs)**2 /
            (c_tau2 * cc + cs_tau * cs + s_tau2 * ss)) +
            ((c_tau * xs - s_tau * xc)**2 /
            (c_tau2 * ss - cs_tau * cs + s_tau2 * cc)))
```

cuSignal

```
_cupy_lombscargle_src = Template(
  """
$header
extern "C" {
  __global__ void _cupy_lombscargle(const int x_shape,
      const int freqs_shape,
      const ${datatype} * __restrict__ x,
      const ${datatype} * __restrict__ y,
      const ${datatype} * __restrict__ freqs,
      ${datatype} * __restrict__ pgram,
      const ${datatype} * __restrict__ y_dot
      ) {
    const int tx {
      static_cast<int>( blockIdx.x * blockDim.x + threadIdx.x ) };
    const int stride { static_cast<int>( blockDim.x * gridDim.x ) };

    for ( int tid = tx; tid < freqs_shape; tid += stride ) {

      // Code not shown

      for ( int j = 0; j < x_shape; j++ ) {

        // Code not shown

       pgram[tid] = 0.5 * (((c_tau * xc + s_tau * xs)**2 /
           (c_tau2 * cc + cs_tau * cs + s_tau2 * ss)) +
           ((c_tau * xs - s_tau * xc)**2 /
           (c_tau2 * ss - cs_tau * cs + s_tau2 * cc)))
```

NVIDIA V100, 1e4 float64 samples, **386x speedup**

Future Direction

Focus on Performance

- Port CuPy Raw CUDA string template to new CuPy C++ template functionality
- Add support for more data types
- Experiment with loading precompiled .cubin files into CuPy
- Expand Numba/Raw CUDA support for functions currently leveraging pure CuPy

Expand Functionality

- GPU accelerated signal reader (SigMF, VITA 49, MIDAS, bin)
- Polyphase channelization
- Digital beamforming

Acknowledgements

Open GPU Data Science community and developers, particularly developers and maintainers of CuPy and Numba

Expedition Technology

Deepwave Digital

Luigi F. Cruz – Early cuSignal adoption with SDRs to demodulate 20MHz FM Spectrum

John Murray

And all contributors – past and future – to cuSignal

Get Started

Join the cuSignal movement!

https://github.com/rapidsai/cusignal

```
conda install -c rapidsai -c nvidia -c conda-forge \
    -c defaults cusignal=0.14 python=3.7 cudatoolkit=10.0
```