Porting Signal Processing Algorithms to CuPy for precision measurement
Porting Signal Processing Algorithms to CuPy for precision measurement

Outline

- Frequency Scanning Interferometry System
- Signal Processing in FSI
- CuPy and Signal Processing
  - Butterworth Filter
  - Hilbert Transform
  - Savitzky-Golay Filter
- Outlook
Frequency Scanning Interferometry System

- Frequency scanning interferometry measurement system for Full Remote Alignment System (FRAS), which can determine distance from measuring head to target up to micrometer precision in real time

- Monitoring the position of magnet and crab cavity cold masses inside their cryostats

- Based on Michelson Interferometry Principle and uses sweeping laser to identify distance of target system

Frequency Scanning Interferometry System

- Based on Michelson Interferometry Principle and uses sweeping laser to identify distance of target system

- Reference beam and the beam reflected from the target are recombined, creating an interference signal -

\[ I(t,\tau) = A \cdot \cos[2\pi(\alpha t + f_0 \tau)] \]

- A - magnitude of the signal
- \( \tau \) - time delay between signals
- \( \alpha \) - sweep rate of the laser
Frequency Scanning Interferometry System

- Based on Michelson Interferometry Principle and uses sweeping laser to identify distance of target system

- Reference beam and the beam reflected from the target are recombined, creating an interference signal -

\[ I(t,\tau) = A \cdot \cos[2\pi(\alpha \tau t + f_0 \tau)] \]

- Distance D is calculated -

\[ D = c \frac{N}{2\Delta v n} \]

\( \Delta v \) - change of the laser frequency during sweep
\( n \) - refractive index
\( c \) - speed of light
\( N \) - number of cycles of the signal measured during the laser sweep (above equation)
Frequency Scanning Interferometry System

- Based on Michelson Interferometry Principle and uses sweeping laser to identify distance of target system
- Reference beam and the beam reflected from the target are recombined, creating an interference signal:
  \[ I(t, \tau) = A \cdot \cos[2\pi(\alpha \tau t + f_0 \tau)] \]
  - A - magnitude of the signal
  - \( \tau \) - time delay between signals
  - \( \alpha \) - sweep rate of the laser
- Distance \( D \) is calculated:
  \[ D = c \frac{N \Delta v n}{2 \Delta v n} \]
  - \( \Delta v \) - change of the laser frequency during sweep
  - \( n \) - refractive index
  - \( c \) - speed of light
  - \( N \) - number of cycles of the signal measured during the laser sweep (above equation)

Introduction to Frequency Scanning Interferometry (FSI) systems - M. Sosin, J. Rutkowski
Frequency Scanning Interferometry System

- Multi-Target Frequency Scanning Interferometry system

\[ I(t, \tau) = A_1 \cdot \cos \left[ 2\pi (\alpha t + f_0 \tau_1) \right] + A_2 \cdot \cos \left[ 2\pi (\alpha t + f_0 \tau_2) \right] \ldots \]

- A1, A2 - magnitude of the signal
- \( \tau \) - time delay between signals
- \( \alpha \) - sweep rate of the laser
Frequency Scanning Interferometry System

- Multi-Target Frequency Scanning Interferometry system
- Fourier Transform based analysis to obtain final distance

\[ D_n = \frac{c \cdot f_{\text{beat}[m]}}{2 \cdot \frac{dv}{dt} \cdot n} \]

\( \alpha \) – is a sweep rate of the laser (\( \alpha = \frac{dv}{dt} \))
\( n \) – refractive index of light transmission medium
\( c \) – speed of light
Frequency Scanning Interferometry System

- FSI interferometer schematic - a) laser delivery and signal analysis b) measurement channels
- Reference Interferometer to identify laser sweep (α)
Frequency Scanning Interferometry System

- FSI interferometer schematic - a) laser delivery and signal analysis b) measurement channels
- Reference Interferometer to identify laser sweep ($\Delta \nu$) or ($\alpha$)

For known length $L$ -

$$L = \frac{c}{2\Delta \nu n}$$

$\Delta \nu$ - change of the laser frequency during sweep
$n$ - refractive index
$c$ - speed of light
$m$ - number of cycles of the signal measured during the laser sweep for length $L$

$$D = \frac{c}{2\Delta \nu} \text{ becomes, } D = \frac{N}{m}$$
Frequency Scanning Interferometry System

FSI Photodetector Module

FSI Test Setup

GPU: Nvidia RTX 3060
Outline

- Frequency Scanning Interferometry System
- Signal Processing in FSI
- CuPy and Signal Processing
  - Butterworth Filter
  - Hilbert Transform
  - Savitzky-Golay Filter
- Outlook
Signal Processing in Frequency Scanning Interferometry

High level control

Low-level FESA
FSI SW
Rx
FPGA
FMC

GPU

8-10 Gbps, direct fiber

Trig. from laser

DIO
FMC
CTRL
FPGA

Meas. chassis

8 x 9.6 Gbps

8 ch

8 ch

FSI Rack

x 4

Process and transfer

Peak distance computation

Calculate:
- Sweep speed for ch2 (least square fit)
- FFT for ch3..64

Linearize ch2..64 based on ch1 (Hilbert, interp)

Transfer raw data @ ~8 Gbps (64 x 3 MB)

Acquire Ch1..64

FPGA of CTRL

GPU & Server's CPU

x 4

20ms
250ms
< 1s

14
Signal Processing in Frequency Scanning Interferometry

Each channel:
N = 2M x 12bit samples (3 MBytes)

Linearization

Data analysis

Each channel:
n = 2M x 12bit samples (3 MBytes)

Gas Cell Processing

Measurement Processing

FSI Rack

Each channel output: Negligible amount of data
Signal Processing in Frequency Scanning Interferometry
Signal Processing in Frequency Scanning Interferometry

Each channel:
N = 2M x 12bit samples (3 MBytes)

Ch 1: Ref.
- Butterworth highpass
- Hilbert Transform
- Arc tan
- Un wrap
- Multiply, Divide, Sort all samples

(2x FFT)

Ch 2: Gas Cell
- Interp. (6-spline)
- Savitzky-Golay (FR)
- Find peaks
- Least square poly fit
- Gas Cell_out

Ch 3, 64: Meas
- Interp. (6-spline)
- FFT (33 MBytes)
- Lorence Fit
- Measure_coef

Gas Cell Processing

Linearization

Data analysis

Each channel:
- n = 2M x 12bit samples (3 MBytes)

Each channel output:
Negligible amount of data
Outline

- Frequency Scanning Interferometry System
- Signal Processing in FSI
- CuPy and Signal Processing
  - Butterworth Filter
  - Hilbert Transform
  - Savitzky-Golay Filter
- Outlook
CuPy

- It is an open-source matrix library accelerated with NVIDIA CUDA.
- It uses CUDA-related libraries including cuBLAS, cuDNN, cuRand, cuSolver, cuSPARSE, cuFFT, and NCCL to make full use of the GPU architecture.
- It is an open-source matrix library accelerated with NVIDIA CUDA.
- It uses CUDA-related libraries including cuBLAS, cuDNN, cuRand, cuSolver, cuSPARSE, cuFFT, and NCCL to make full use of the GPU architecture.
CuPy

- It is an open-source matrix library accelerated with NVIDIA CUDA.
- It uses CUDA-related libraries including cuBLAS, cuDNN, cuRand, cuSolver, cuSPARSE, cuFFT, and NCCL to make full use of the GPU architecture.
- Provides High performance N-dimensional array computation.
- Drop in replacement for Numpy -
CuPy

- It is an open-source matrix library accelerated with NVIDIA CUDA.
- It uses CUDA-related libraries including cuBLAS, cuDNN, cuRand, cuSolver, cuSPARSE, cuFFT, and NCCL to make full use of the GPU architecture.
- Provides High performance N-dimensional array computation.
- Open Source and distributed under MIT License
- Easy to start with and scale and test
- Develop custom Kernels using JIT - NUMBA
CuPy and Signal Processing Algorithms

Support for some of the Scipy routines is available:

- Discrete Fourier Transform
  - `fft`, `rfft`, `ifft`, `fft2`, `irfft`, `fftshift`

- Linear Algebra
  - `lu`, `eigsh`, `lsqr`

- Multidimensional Image processing
  - `gaussian_filter`, `laplace`, `convolve`, `grey_dilation`, `grey_erosion`

- Signal Processing
  - `fftconvolve`, `correlate`, `medfit`

- Sparse Matrices
  - ...... and many more

CuPy and Signal Processing Algorithms

Support for some of the Scipy routines is available:

- **Discrete Fourier Transform**
  - `fft`, `rfft`, `ifft`, `fft2`, `irfft`, `fftshift`

- **Linear Algebra**
  - `lu`, `eigsh`, `lsqr`

- **Multidimensional Image processing**
  - `gaussian_filter`, `laplace`, `convolve`, `grey_dilation`, `grey_erosion`

- **Signal Processing**
  - `fftconvolve`, `correlate`, `medfit`

- **Sparse Matrices** … and many more

[Cusignal - RAPIDS](https://docs.rapids.ai/api/cusignal/stable/api.html)
Considerations while porting to GPU:

1] Check the data format

2] Check number of Device to Host and Host to Device Memory Transactions

3] No recursion functions are present

4] GPU is good if you have large data set to process and have possibility of either Data parallelism or Task parallelism
Outline

- Frequency Scanning Interferometry System
- Signal Processing in FSI
- CuPy and Signal Processing
  - Butterworth Filter
  - Hilbert Transform
  - Savitzky-Golay Filter
- Outlook
Butterworth Filter

- To reduce the background noise and suppress the interfering signals by removing some frequencies - filters are used.

- The frequency range which is allowed: passband and the range which is suppressed is stopband.

- Butterworth filter provides maximum flat response in passband i.e least ripple.

- Transfer Function of Butterworth Filter:

\[ |H_b(jw)| = \frac{1}{\sqrt{1 + \left(\frac{w}{w_c}\right)^{2N}}} \]

\( w_c \) = cut-off frequency

\( N = \) Order of Filter
Butterworth High Pass Filter

Each channel:
N = 2M x 12bit samples (3 MBytes)

Ch 1: Ref.
- Butterworth highpass
- Hilbert Transform
- Arc tan
- Un wrap
- Multiply, Divide, Sort all samples

(2x FFT)

Ch 2: Gas Cell
- Interp. (B-spline)
- Savitzky-Golay (FIR)
- Find peaks
- Least square polyfit
- GAS_CELL_out
- n=48

Ch 3..64: Meas
- Interp. (B-spline)
- FFT (3*3 MB)
- FFT_freq
- FFT_magnitude
- FFT_out
- MEAS_out
- m=4

Linearization

Data analysis

Each channel:
n = 2M x 12bit samples (3 MBytes)

FSI rack

Each channel output:
Negligible amount of data

Measurement Processing

Gas Cell Processing

O(N)*m^2
m=4

O(N)*m
m=100

O(N)

O(N)

O(N)

O(N^2)

Dist(s)

Meas_coef
Butterworth High Pass Filter in CuPy

1] Calculate z,p,k for Lowpass analog prototype

```
z = cp.array([])
m = cp.arange(-N+1, N, 2)
p = -cp.exp(1j * pi * m / (2 * N))
k = 1
```

2] Pre-warp frequencies for Digital Filter

```
warped = 2 * fs * cp.tan(pi * Wn / fs)
```

3] Convert Lowpass analog prototype to Highpass, wo= cutoff frequency

```
z_hp = wo / z
p_hp = wo / p
z_hp = cp.append(z_hp, cp.zeros(degree))
k_hp = k * cp.real(cp.prod(-z) / cp.prod(-p))
```

4] Return digital filter parameters using Bilinear Transformation fs = 2.0*fs

```
z_z = (fs + z) / (fs - z)
p_z = (fs + p) / (fs - p)
z_z = cp.append(z_z, -cp.ones(degree))
k_z = k * cp.real(cp.prod(fs - z) / cp.prod(fs - p))
```

5] Convert to b/a form from z,p,k
Performance Analysis: Butterworth Filter

1] Calculate filter Transfer Function

```python
nyq = 0.5 * fs
normal_cutoff = cutoff / nyq
b, a = butter(order, normal_cutoff, btype='high', analog=False)
```

2] Apply using lfilter

```python
data = Reference_cell
ret = lfilter(b, a, data)
```

3] Apply using FFT

```python
delta = np.zeros(np.size(t))
delta[1] = 1;
filter_butter = lfilter(b, a, delta)
filter_butter = cp.array(filter_butter)
filter_fft = cupyx.scipy.fft.fft(filter_butter)
data_fft = cupyx.scipy.fft.fft(data)
res_fft = cp.multiply(data_fft, filter_fft)
res_fft = cp.array(res_fft)
res = cupyx.scipy.fft.irfft(res_fft)
```
Frequency Scanning Interferometry System

Signal Processing in FSI

CuPy and Signal Processing
- Butterworth Filter
- Hilbert Transform
- Savitzky-Golay Filter

Outlook
Hilbert Transform

- It is useful for calculating instantaneous attributes of a time series, especially the amplitude and the frequency.

- The instantaneous amplitude is the amplitude of the complex Hilbert transform and the instantaneous frequency is the time rate of change of the instantaneous phase angle.

- It returns Analytic Signal ‘x’

\[ x = x_r + jx_i \]

- \( x_r \) is the original data
- \( x_i \) and an imaginary part, which contains the Hilbert transform. The imaginary part is a version of the original real sequence with a 90° phase shift.
Hilbert Transform in CuPy

1] Compute Fast Fourier Transform of Real-valued Signal

```python
Xf = cupyx.scipy.fft.fft(x, N, axis=axis)
h = cp.zeros(N)
```

2] Rotate the Fourier Coefficients to obtain imaginary part

```python
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
```

```python
if x.ndim > 1:
    ind = [cp.newaxis] * x.ndim
    ind[axis] = slice(None)
    h = h[tuple(ind)]
```

3] Compute Inverse Fourier Transform to get the Analytic Signal

```python
x = cupyx.scipy.fft.ifft(Xf * h, axis=axis)
```

4] Calculate Instantaneous Frequency and phase
Hilbert Transform

Each channel:
\[ N = 2M \times 12\text{bit samples} = 3 \text{ MBytes} \]

Reference Processing
- Butterworth highpass
- Hilbert Transform
- Arc tan
- Un wrap

(2x FFT)

Data analysis
- Each channel:
  \[ n = 2M \times 12\text{bit samples} = 3 \text{ MBytes} \]

Gas Cell Processing
- \( O(N^2m) \), \( m=4 \)
- Savitzky-Golay (FIR)
- Least square polyfit
- Meas_coef

Measurement Processing
- \( O(N^2m^2) \), \( m=4 \)
- FFT
- Lorenze Fit
- Dst(s)

FSI rack
- Each channel output:
  Negligible amount of data

\[ x \times 4 \]
Hilbert Transform of Reference Cell Data

```python
def DataLinearize(Tinterval, REF_IFM, plot='false') :
    
    fs= 1/Tinterval
    REF_IFM = filterDataButterworthHighpass(REF_IFM, 100000, fs)
    t = cp.linspace(0.0, len(REF_IFM)*Tinterval, num=len(REF_IFM))
    start=time.time()
    analytic_signal = hilbert_gpu(REF_IFM, axis=0)
    Time_GPU_HT = time.time() - start
    REF_IFM = cp.asnumpy(REF_IFM)
    start=time.time()
    analytic_signal2 = hilbert(REF_IFM, axis=0)
    Time_CPU_HT = time.time() - start
    print("Time taken by CPU %s" %time.time()-start)
    
    phase = cp.angle(analytic_signal)
    instantaneous_phase = cp.unwrap(phase, axis=0)
    instantaneous_phase = cp.asnumpy(instantaneous_phase)
    del phase
    del analytic_signal

    f_theor = cp.max(instantaneous_phase)/(2*3.14*(Tinterval*len(instantaneous_phase)))
    t_simu = cp.array(instantaneous_phase/(2*3.14*f_theor))
    t_simu[0] = 0
    t_simu=cp.sort(t_simu)
    return t,t_simu,Time_GPU_HT,Time_CPU_HT
```
def DataLinearize(Tinterval, REF_IFM, plot='false') :
    fs= 1/Tinterval
    REF_IFM = filterDataButterworthHighpass(REF_IFM, 100000, fs)
    t = cp.linspace(0.0, len(REF_IFM)*Tinterval, num=len(REF_IFM))
    start=time.time()
    analytic_signal = hilbert_gpu(REF_IFM, axis=0)
    Time_GPU_HT = time.time() - start
    REF_IFM = cp.asnumpy(REF_IFM)
    start=time.time()
    analytic_signal2 = hilbert(REF_IFM, axis=0)
    Time_CPU_HT = time.time() - start
    print("Time taken by CPU %s" % (time.time()-start))

    phase = cp.angle(analytic_signal)
    instantaneous_phase = cp.unwrap(phase, axis=0)
    instantaneous_phase = cp.asnumpy(instantaneous_phase)
    del phase
    del analytic_signal

    f_theor = cp.max(instantaneous_phase)/(2*3.14*(Tinterval*len(instantaneous_phase)))
    t_simu = cp.array(instantaneous_phase/(2*3.14*f_theor))
    t_simu[0] = 0
    t_simu=cp.sort(t_simu)
    return t,t_simu,Time_GPU_HT,Time_CPU_HT
def DataLinearize(Tinterval, REF_IFM, plot='false'):
    fs= 1/Tinterval
    REF_IFM = filterDataButterworthHighpass(REF_IFM, 100000, fs)
    t = cp.linspace(0.0, len(REF_IFM)*Tinterval, num=len(REF_IFM))
    start=time.time()
    analytic_signal = hilbert_gpu(REF_IFM, axis=0)
    Time_GPU_HT = time.time() - start
    REF_IFM = cp.asnumpy(REF_IFM)
    start=time.time()
    analytic_signal2 = hilbert(REF_IFM, axis=0)
    Time_CPU_HT = time.time() - start
    print("Time taken by CPU %s" %(time.time()-start))
    
    phase = cp.angle(analytic_signal)
    instantaneous_phase = cp.unwrap(phase, axis=0)
    instantaneous_phase = cp.asnumpy(instantaneous_phase)
    del phase
    del analytic_signal
    
    f_theor = cp.max(instantaneous_phase)/(2*3.14*(Tinterval*len(instantaneous_phase)))
    t_simu = cp.array(instantaneous_phase/(2*3.14*f_theor))
    t_simu[0] = 0
    t_simu=cp.sort(t_simu)
    return t,t_simu,Time_GPU_HT,Time_CPU_HT
Performance Analysis: Hilbert Transform

Performance of FFT Cupy Kernel in timeline view
Outline

- Frequency Scanning Interferometry System
- Signal Processing in FSI
- CuPy and Signal Processing
  - Butterworth Filter
  - Hilbert Transform
  - Savitzky-Golay Filter
- Outlook
Savitzky-Golay Filter

- It is a digital filter that can be applied to a set of digital data points for smoothing the data without distorting the original signal tendency or to calculate the derivative of signal.

- Find least-square fit for each window and replace each data point with coefficient of that polynomial.

- But the smoothed output obtained by fitting polynomial to each window is equivalent to convolution of convolution coefficients (weighting coefficients) with each window/segment.
Savitzky-Golay Filter

- It is a digital filter that can be applied to a set of digital data points for smoothing the data without distorting the original signal tendency or to calculate the derivative of signal.
- Find least-square fit for each window and replace each data point with coefficient of that polynomial
- But the smoothed output obtained by fitting polynomial to each window is equivalent to convolution of ‘convolution coefficients(weighting coefficients)’ with each window/segment
Savitzky-Golay Filter in CuPy

1] Precompute the coefficients based on order and window length

\[
b = \text{cp.array([[k**i for i in range(order+1)] for k in range(-\text{half\_window}, half\_window+1)]}) \\
m = \text{cp.linalg.pinv}(b) \\
m = \text{cp.multiply}(m[\text{deriv}], \text{cp.multiply}(\text{cp.power}(\text{rate, deriv}), \text{factorial}(\text{deriv})))
\]

2] Pad the signal at the extremes

\[
\text{extr\_begin} = y[0] - \text{cp.abs}(y[1:half\_window+1][::-1] - y[0]) \\
\text{extr\_end} = y[-1] + \text{cp.abs}(y[-half\_window-1:-1][::-1] - y[-1]) \\
y = \text{cp.concatenate}((\text{extr\_begin}, y, \text{extr\_end}))
\]

3] Convolve signal with calculated coefficients

\[
\text{result} = \text{cp.convolve}(m[::-1], y, \text{mode}='valid')
\]
Applying Savitzky-Golay Filter to Gas Cell Data

Each channel:
N = 2M x 12bit samples (3 MBytes)

Linearization:
- Butterworth highpass
- Hilbert Transform
- Arc tan
- Un wrap

Data analysis:
- Multiply, Divide, Sort all samples

Gas Cell Processing:
- O(N*m)
- O(N*m)
- O(N)
- O(N)
- O(N)

Measurement Processing:
- O(N*m^2)
- O(N log N)
- O(N)
- O(N^2)

Ch 1: Ref.
- t_simu

Ch 2: Gas Cell
- Gas Cell Processing
- interp. (B-spline)
- Savitzky-Golay (FIR)
- Find peaks
- Least square polyfit

Ch 3..64: Meas
- Measurement Processing
- interp. (B-spline)
- FFT (3*3 MB)
- Lorenze Fit

FSI rack
Each channel:
n = 2M x 12bit samples (3 MBytes)
Each channel output: Negligible amount of data
Applying Savitzky-Golay Filter to Gas Cell Data

Spectrum of Hydrogen Cyanide (HCN) SRM2519a absorption gas cell - used to track the “true” frequency of the sweeping laser
Applying Savitzky-Golay Filter to Gas Cell Data

Gas Cell Spectrum

Filtered Gas Cell
Applying Savitzky-Golay Filter to Gas Cell Data

Gas Cell Spectrum
Performance Analysis: Savitzky-Golay Filter

```python
start=time.time()
savg_cpu = scipy.signal.savgol_filter(FilteredGasCell, 201, 2)
Time_CPU_SG=time.time()-start

FilteredGasCell = cp.array(FilteredGasCell)
start=time.time()
FilteredGasCell = savgol_filter_gpu(FilteredGasCell, window_size=201, order=2)
Time_GPU_SG=time.time()-start
```
Performance Analysis: Savitzky-Golay Filter

Some more insights about CUDA Kernels using profiling tools - nsys profile, nsys-ui
Peak Detection for Gas Cell

● There is no function like `scipy.signal.find_peaks()` in CuPy yet.

● Peak detection for Gas Cell on GPU is developed based on idea that a peak must be greater (or smaller) than its immediate neighbors, but the performance need to be improved.

```python
def detect_peaks_gpu(x, mph=None, mpd=1, threshold=0)
```
A comparison of `cupy` and `numpy` implementations on 2.5 million data points sample (time in seconds):

<table>
<thead>
<tr>
<th>Routines</th>
<th>Numpy</th>
<th>CuPy</th>
<th>Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interpolation:</td>
<td>np.interp -&gt; cp.interp</td>
<td>0.055173</td>
<td>0.001341</td>
</tr>
<tr>
<td>Unwrap:</td>
<td>np.unwrap-&gt;cp.unwrap</td>
<td>0.143882</td>
<td>0.015522</td>
</tr>
<tr>
<td>Convolution:</td>
<td>np.convolve-&gt;cp.convolve</td>
<td>0.326742</td>
<td>0.014102</td>
</tr>
<tr>
<td>Angle:</td>
<td>np.angle-&gt; cp.angle</td>
<td>0.165760</td>
<td>0.004315</td>
</tr>
<tr>
<td>Sort:</td>
<td>np.sort-&gt;cp.sort</td>
<td>0.071232</td>
<td>0.002608</td>
</tr>
<tr>
<td>Absolute:</td>
<td>np.abs-&gt;cp.abs</td>
<td>0.005381</td>
<td>0.004910</td>
</tr>
</tbody>
</table>
cupyx.scipy.fft(x[, n, axis, norm, overwrite_x, plan])

- access advanced routines that cuFFT like `get_plan_fft()`

\[
Y = \text{cp.fft.rfft(Meas\_Linear, int(len(Meas\_Linear)))}
\]
Outline

- Frequency Scanning Interferometry System
- Signal Processing in FSI
- CuPy and Signal Processing
  - Butterworth Filter
  - Hilbert Transform
  - Savitzky-Golay Filter
- Outlook
Outlook: What lies ahead?

- CuPy - a great library to start and test processing on GPU and expand to Signal Processing
- More performance tests and analysis to do with multiple channels and ultimately to improve performance
- Move to more CuPy based custom Kernels
- Upstreaming developments to CuPy repository
Outlook: What lies ahead?

- CuPy - a great library to start and test processing on GPU and expand to Signal Processing

- More performance tests and analysis to do with multiple channels and ultimately to improve performance

- Move to more CuPy based custom Kernels

- Upstreaming developments to CuPy repository

Now this is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning - Winston Churchill