Porting Signal Processing Algorithms to CuPy for precision measurement

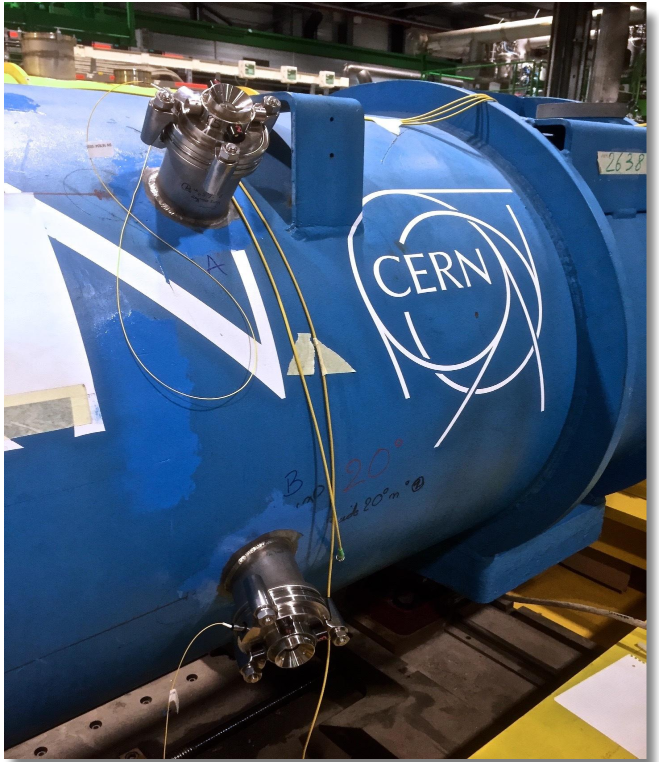
Porting Signal Processing Algorithms to CuPy for precision measurement

Acknowledgements: D.Cobas, M.Lipinski, M.Sosin, P.Peronnard, T.Gingold, C.Franco, T.Wlostowski (CERN)

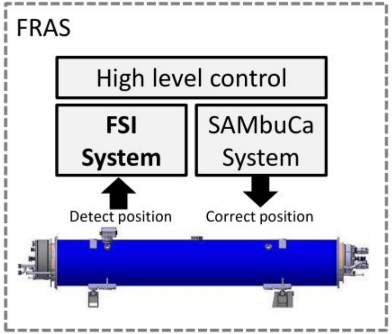
Outline

- 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 upto 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



https://home.cern/news/news/accelerators/aligning-hl-lhc-magnets-interferometry

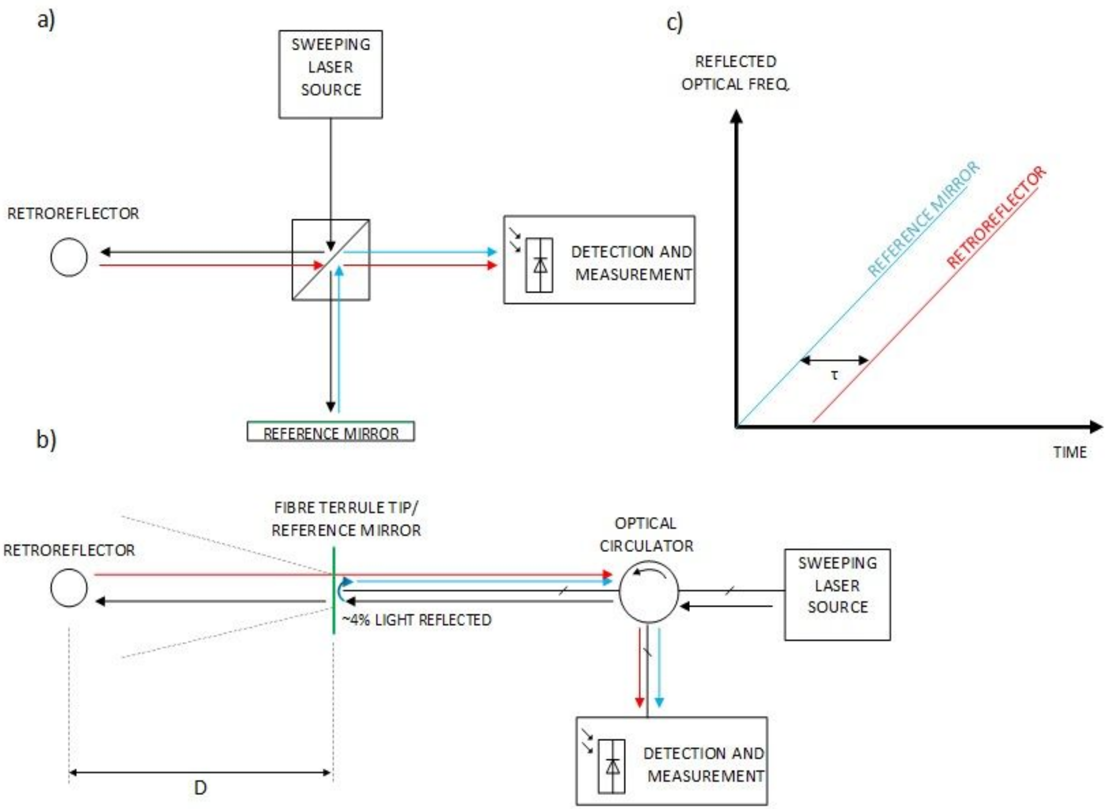
Frequency Scanning Interferometry System

- 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
τ - time delay between signals
α - sweep rate of the laser



Frequency Scanning Interferometry System

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

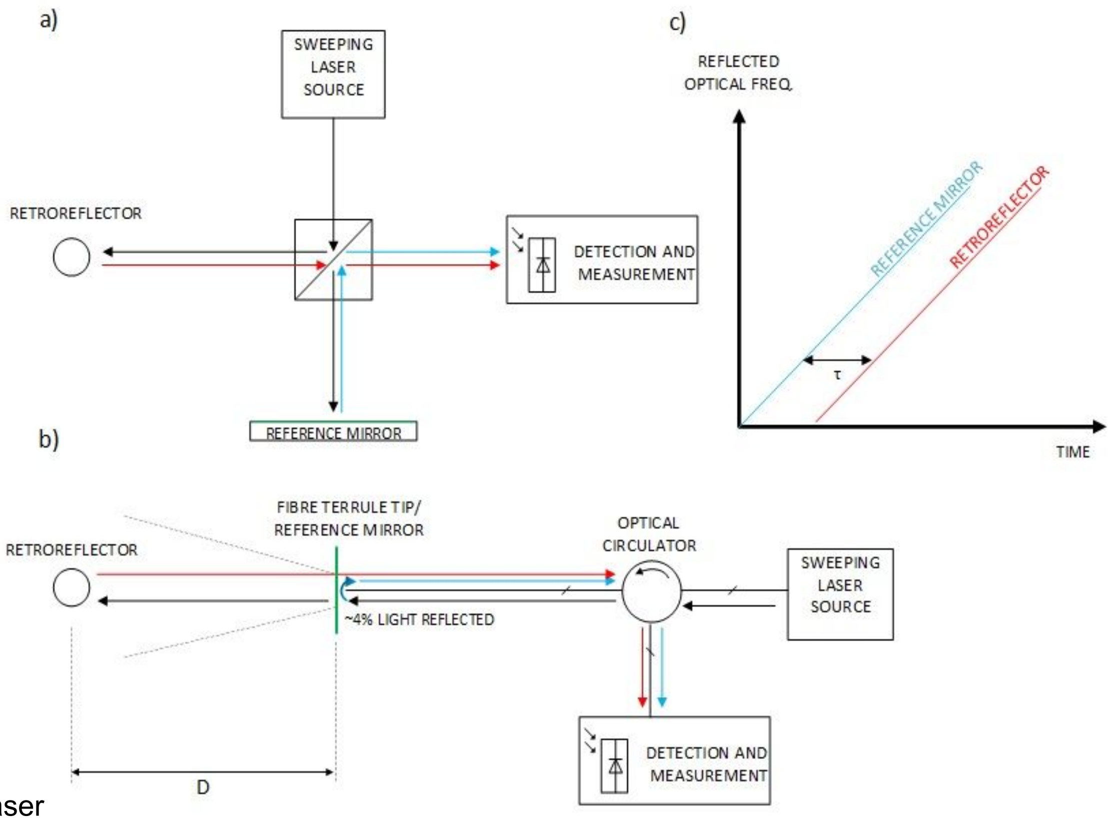
- Reference beam and the beam reflected from the target are recombined, creating an interference signal -

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

A - magnitude of the signal
τ - time delay between signals
α - sweep rate of the laser

- Distance D is calculated - $D = c\frac{N}{2\Delta\nu n}$

$\Delta\nu$ - 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

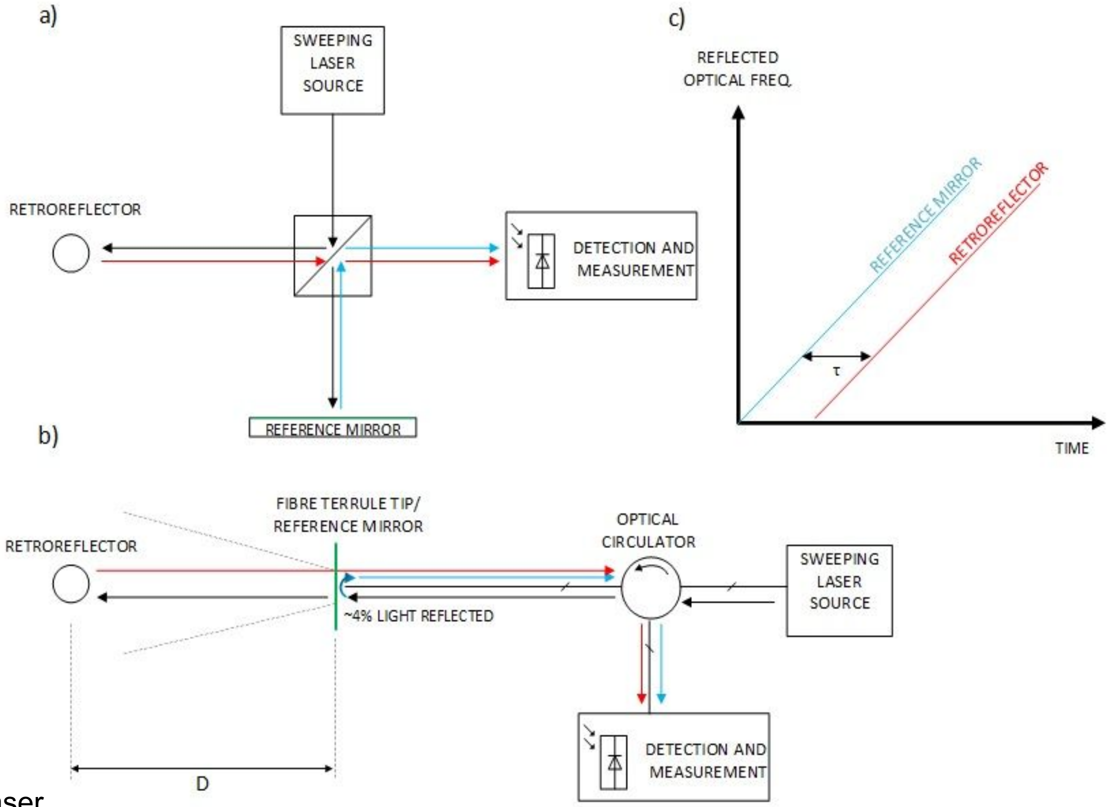
- Reference beam and the beam reflected from the target are recombined, creating an interference signal -

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

A - magnitude of the signal
τ - time delay between signals
α - sweep rate of the laser

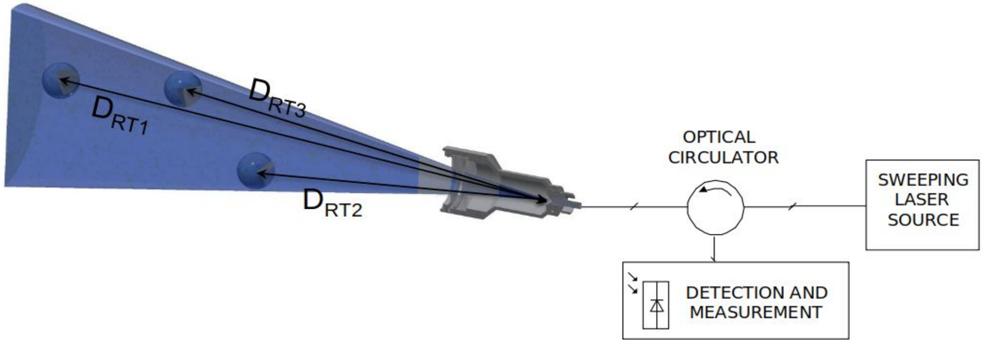
- Distance D is calculated - $D = c\dfrac{N}{2\Delta\nu n}$

$\Delta\nu$ - 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)



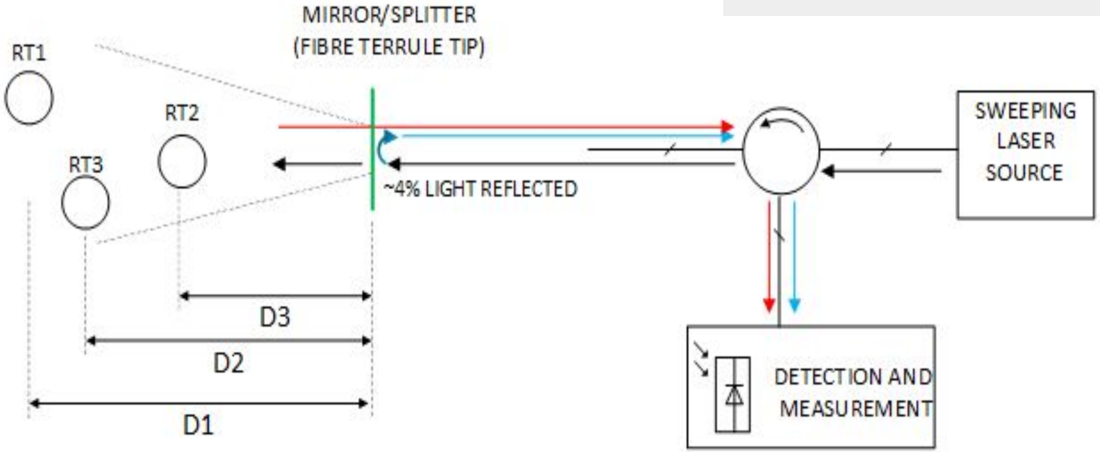
Introduction to Frequency Scanning Interferometry (FSI) systems - M. Sosin, J. Rutkowski

Frequency Scanning Interferometry System



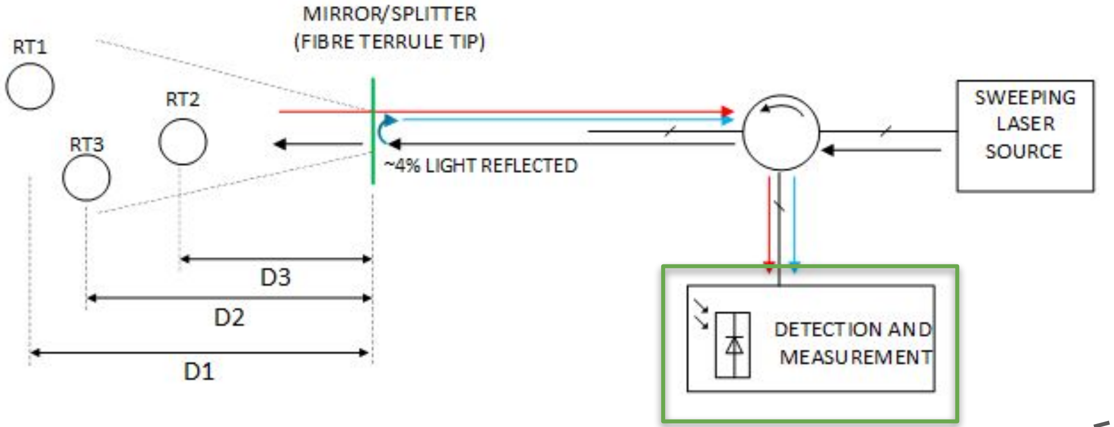
- Multi-Target Frequency Scanning Interferometry system

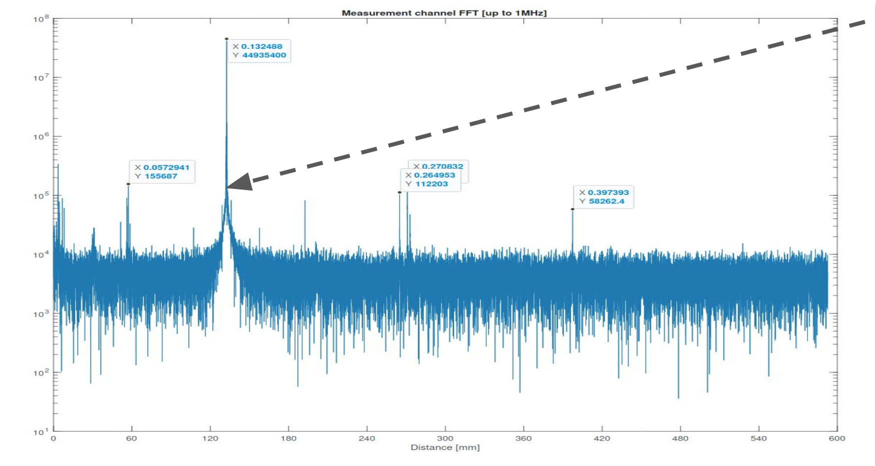
$$I(t,\tau) = A_1 \cdot \cos[2\pi(\alpha\tau_1 t + f_0 \tau_1)] + A_2 \cdot \cos[2\pi(\alpha\tau_2 t + f_0 \tau_2)...$$



A1, A2 - magnitude of the signal
τ - time delay between signals
α - sweep rate of the laser

Frequency Scanning Interferometry System



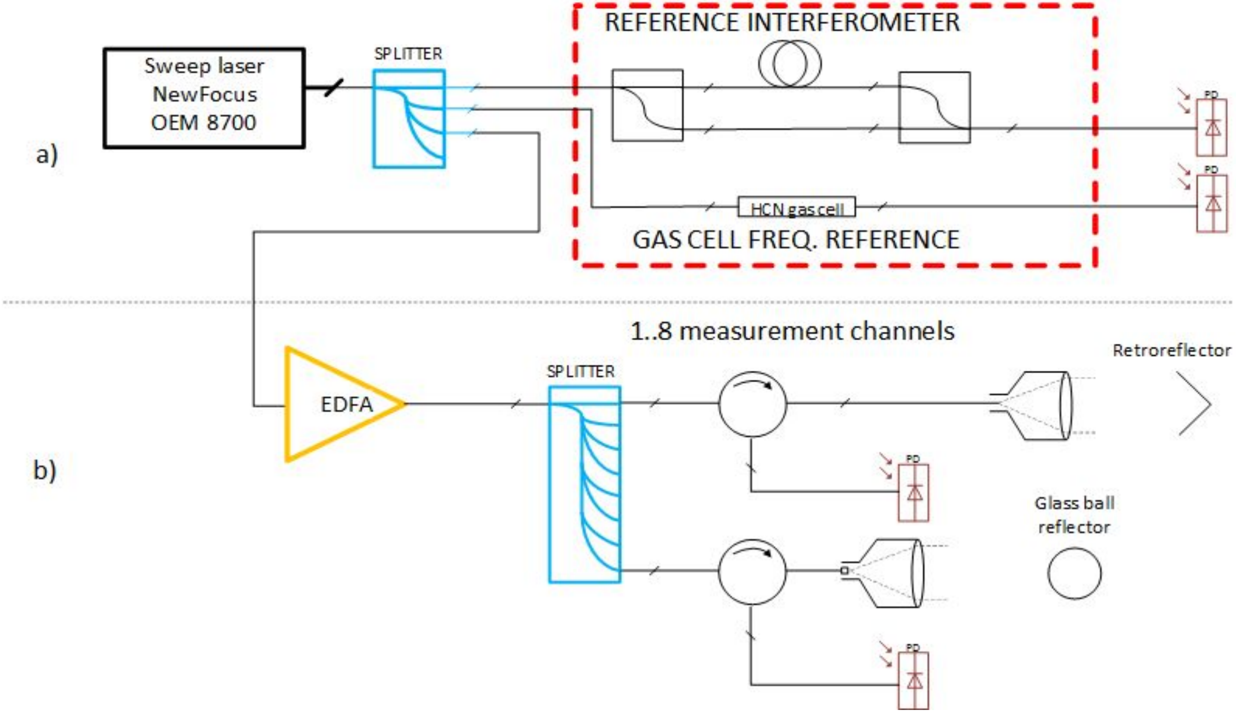


- Multi-Target Frequency Scanning Interferometry system
- Fourier Transform based analysis to obtain final distance

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

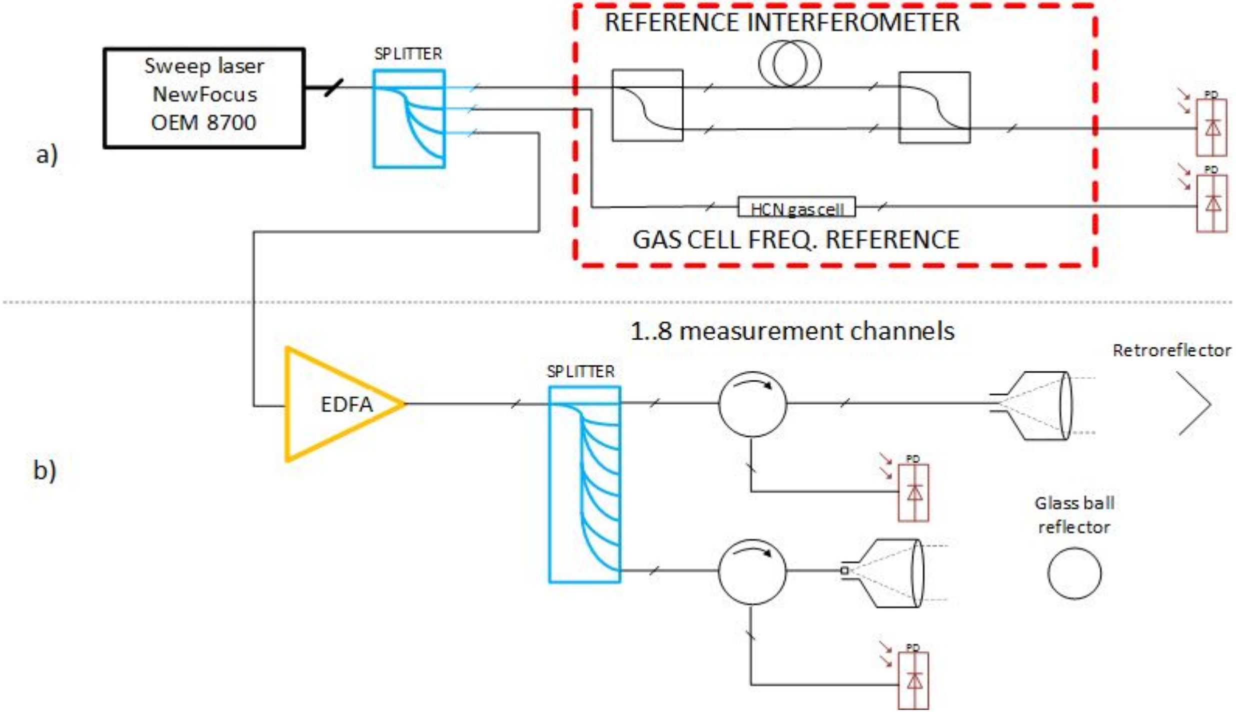
α – is a sweep rate of the laser ($\alpha = dv/dt$)
n – refractive index of light transmission medium
c – speed of light

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



For known length L -

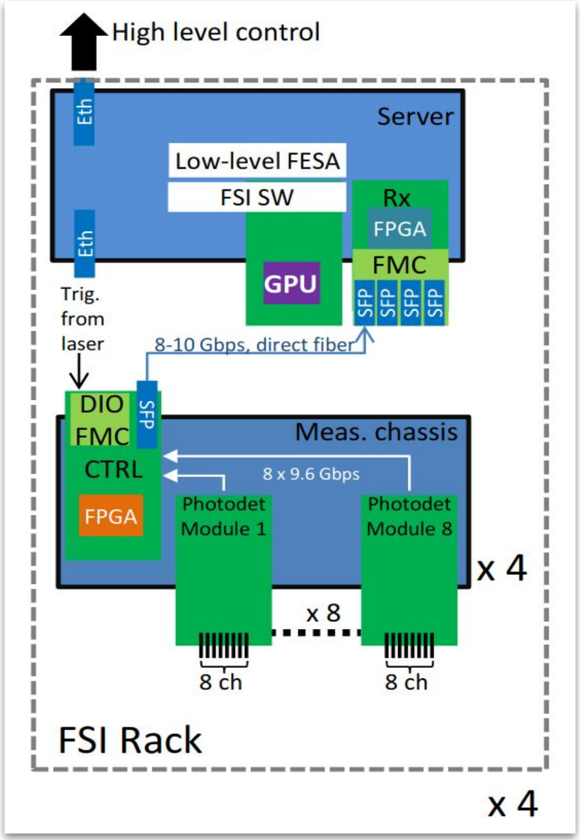
$$L = c\frac{m}{2\Delta\nu n}$$

$\Delta\nu$ - 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

$$D = c\frac{N}{2\Delta\nu n}$$ becomes, $$D = L\frac{N}{m}$$

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

Frequency Scanning Interferometry System



FSI Test Setup

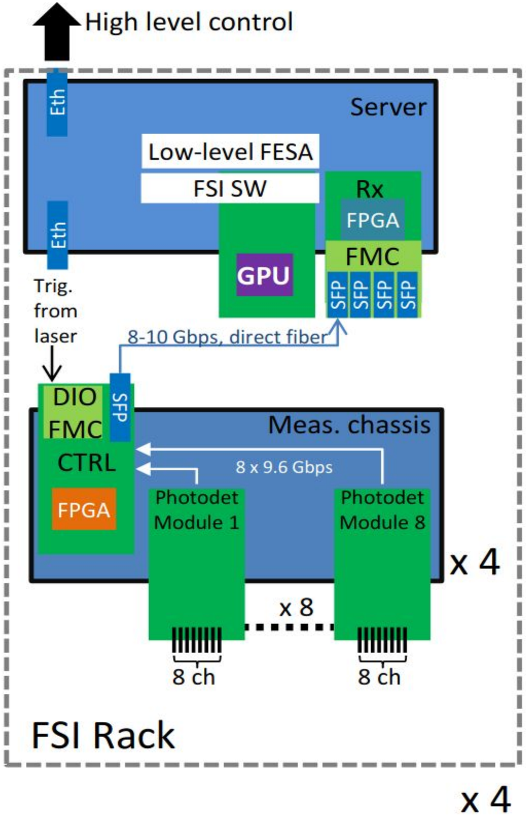
FSI Photodetector Module

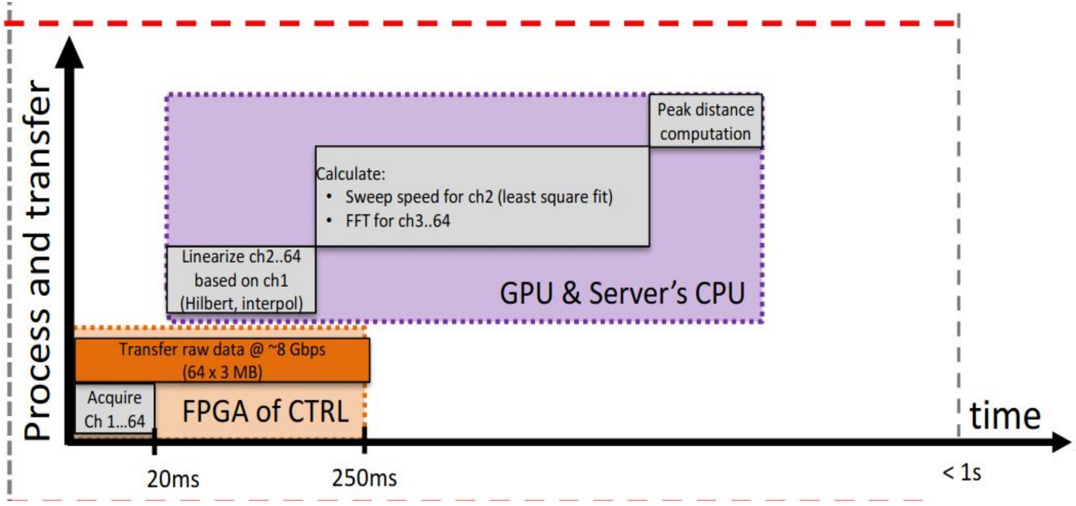
GPU: Nvidia RTX 3060

Outline

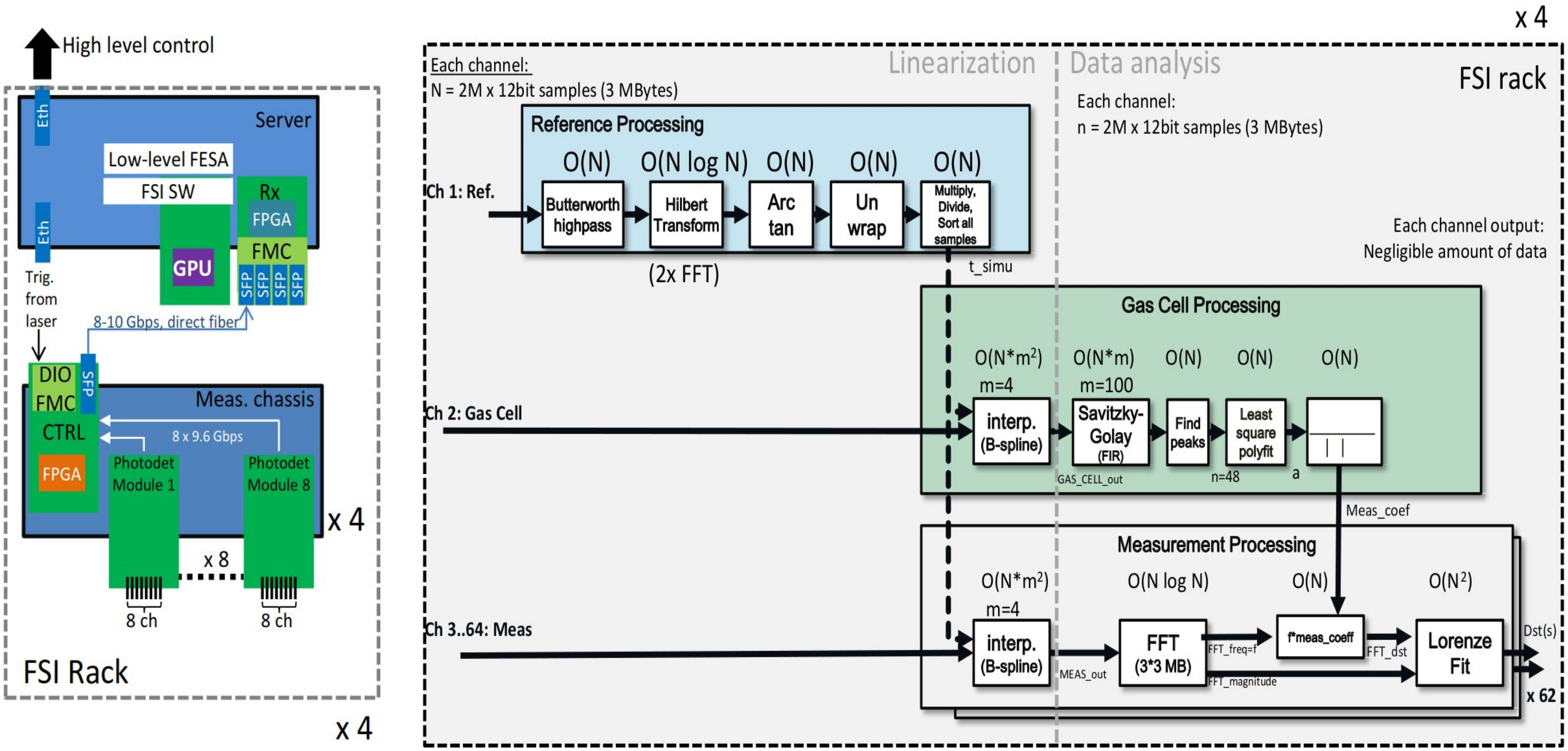
- Frequency Scanning Interferometry System
- Signal Processing in FSI
- CuPy and Signal Processing
  - Butterworth Filter
  - Hilbert Transform
  - Savitzky-Golay Filter
- Outlook

Signal Processing in Frequency Scanning Interferometry

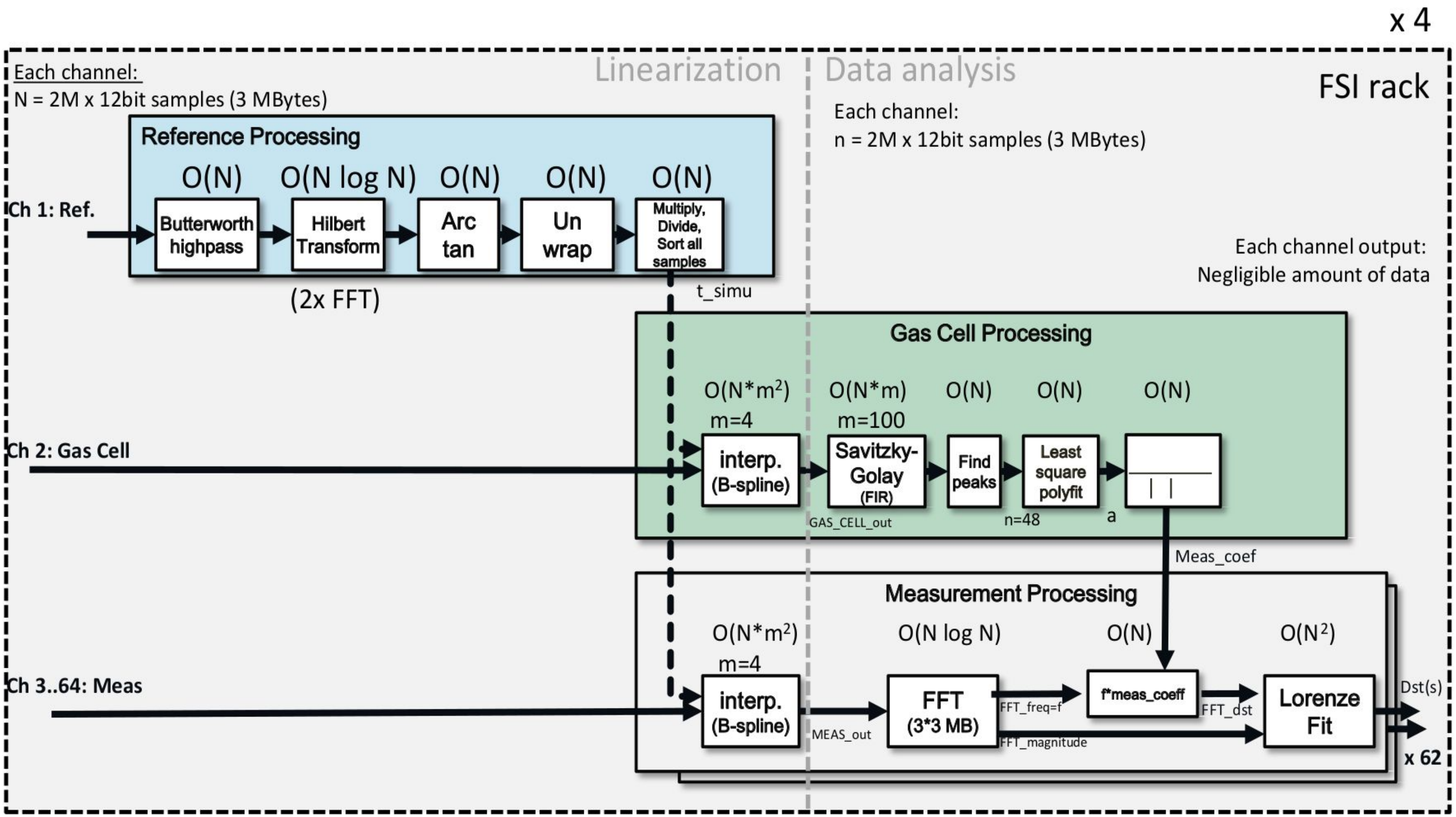




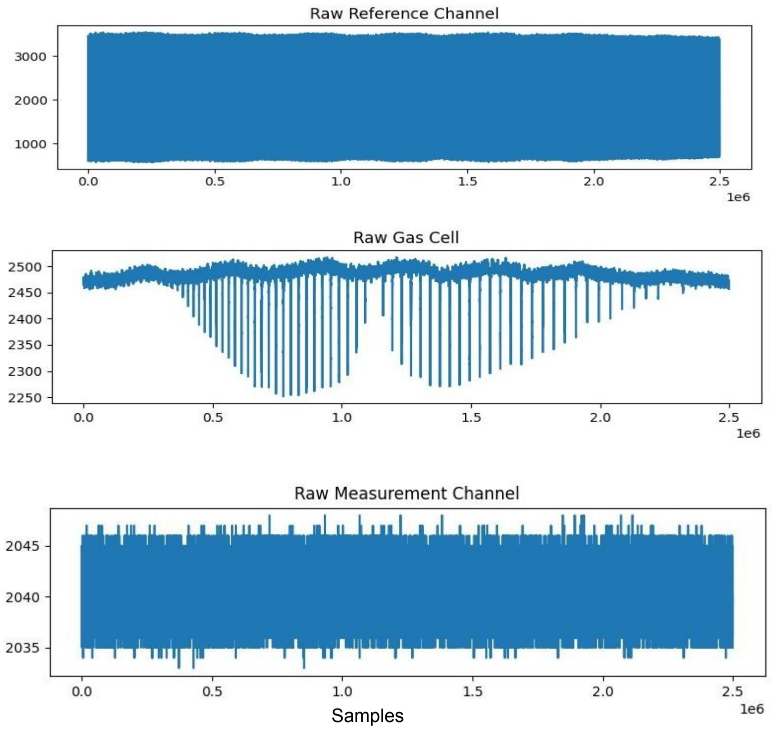
Signal Processing in Frequency Scanning Interferometry

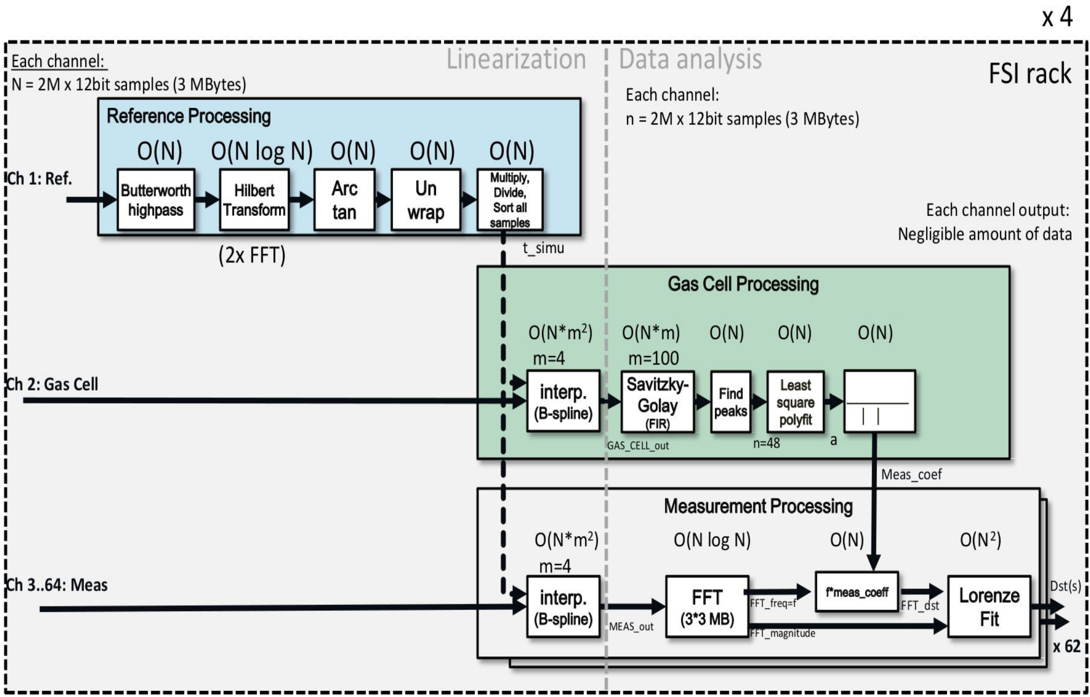


Signal Processing in Frequency Scanning Interferometry



Signal Processing in Frequency Scanning Interferometry





Outline

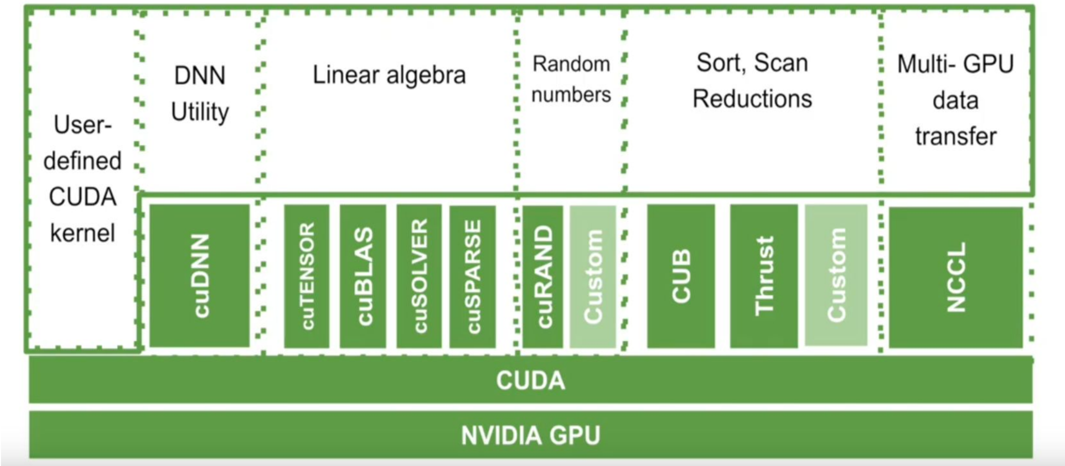
- Frequency Scanning Interferometry System
- Signal Processing in FSI
- CuPy and Signal Processing
  - Butterworth Filter
  - Hilbert Transform
  - Savitzky-Golay Filter
- Outlook

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

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



https://cupy.dev/

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 - *https://docs.cupy.dev/en/stable/reference/comparison.html*

CuPy

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 - *https://docs.cupy.dev/en/stable/reference/comparison.html*
- Open Source and distributed under MIT License
- Easy to start with and scale and test
- Develop custom Kernels using JIT - NUMBA

CuPy

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

  https://docs.cupy.dev/en/stable/reference/scipy.html#

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

https://docs.cupy.dev/en/stable/reference/scipy.html#

- Cusignal - RAPIDS

https://docs.rapids.ai/api/cusignal/stable/api.html

CuPy and Signal Processing Algorithms

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

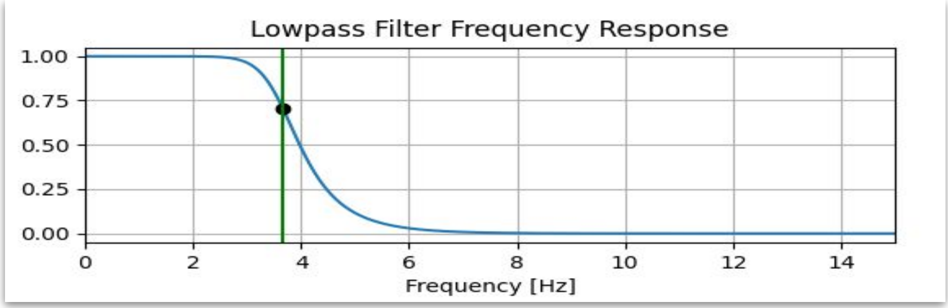
- Butterworth filter provides maximum flat response in passband i.e least ripple

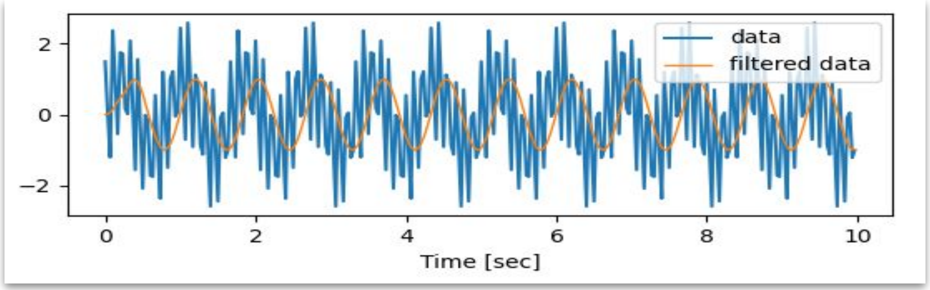
- Transfer Function of Butterworth Filter:

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

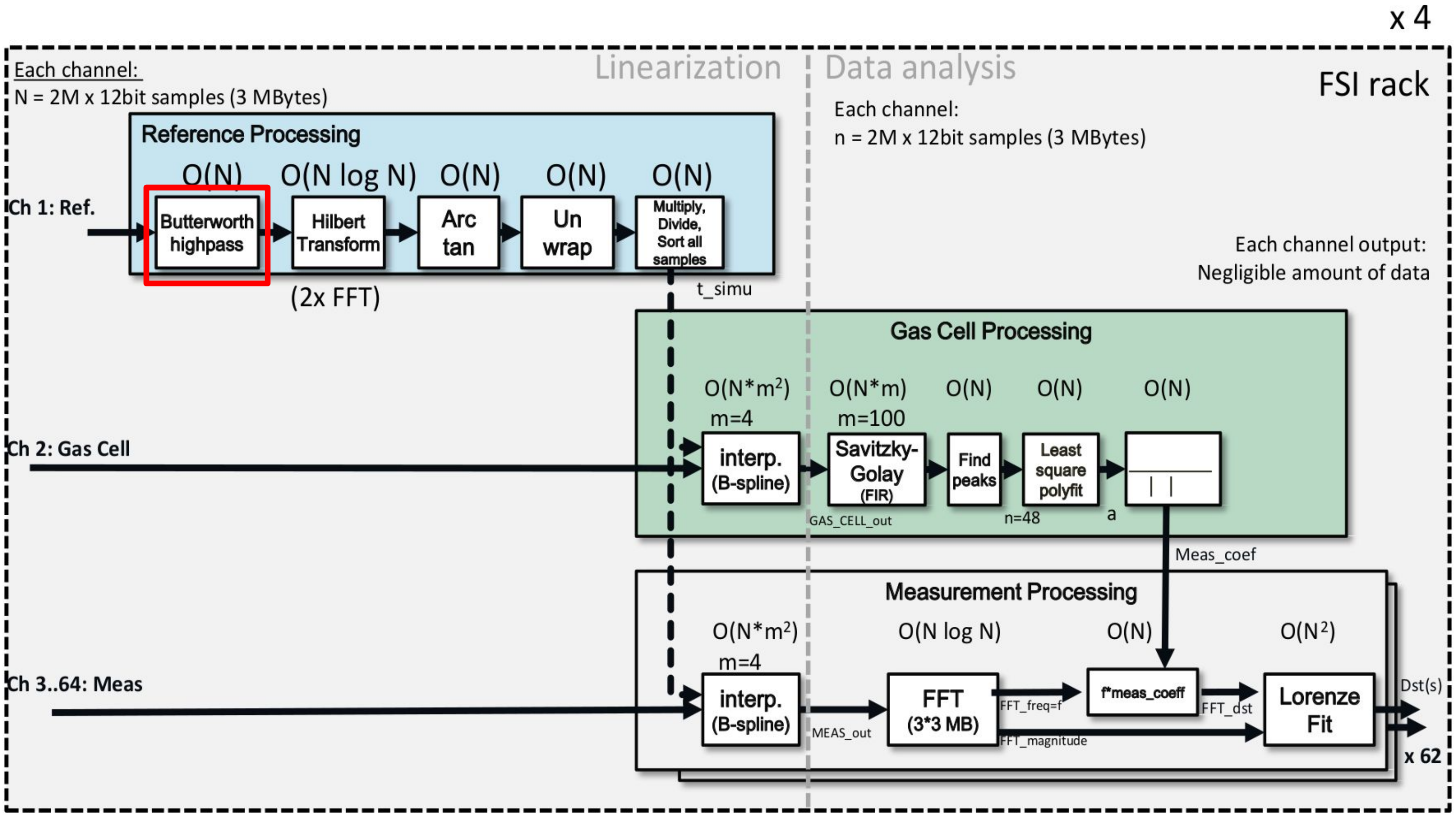
w_c = cut-off frequency

N = Order of Filter





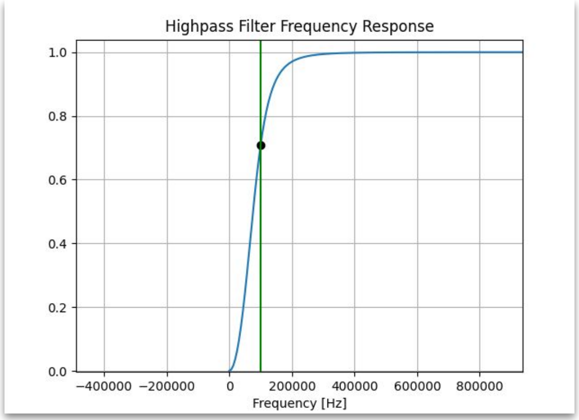
Butterworth High Pass Filter



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

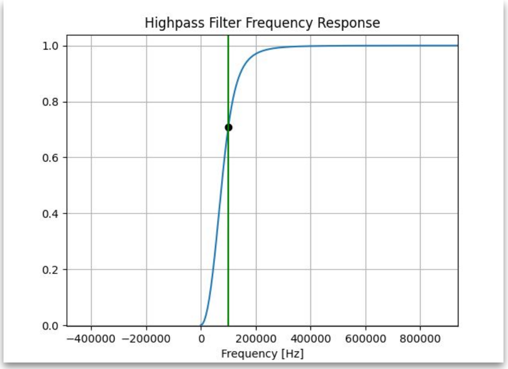
```
nyq = 0.5 * fs
normal_cutoff = cutoff / nyq
b, a = butter(order, normal_cutoff, btype='high', analog=False)
```

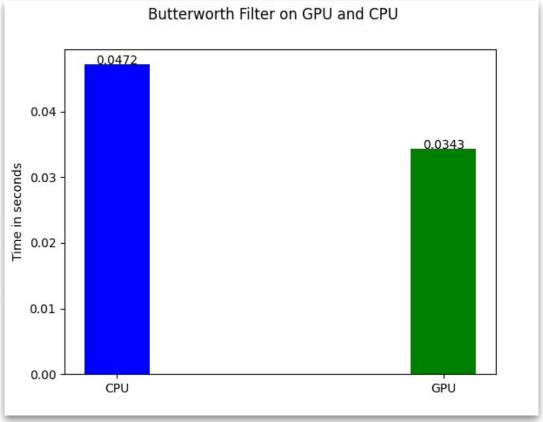
2] Apply using lfilter

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

3] Apply using FFT

```
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)
```



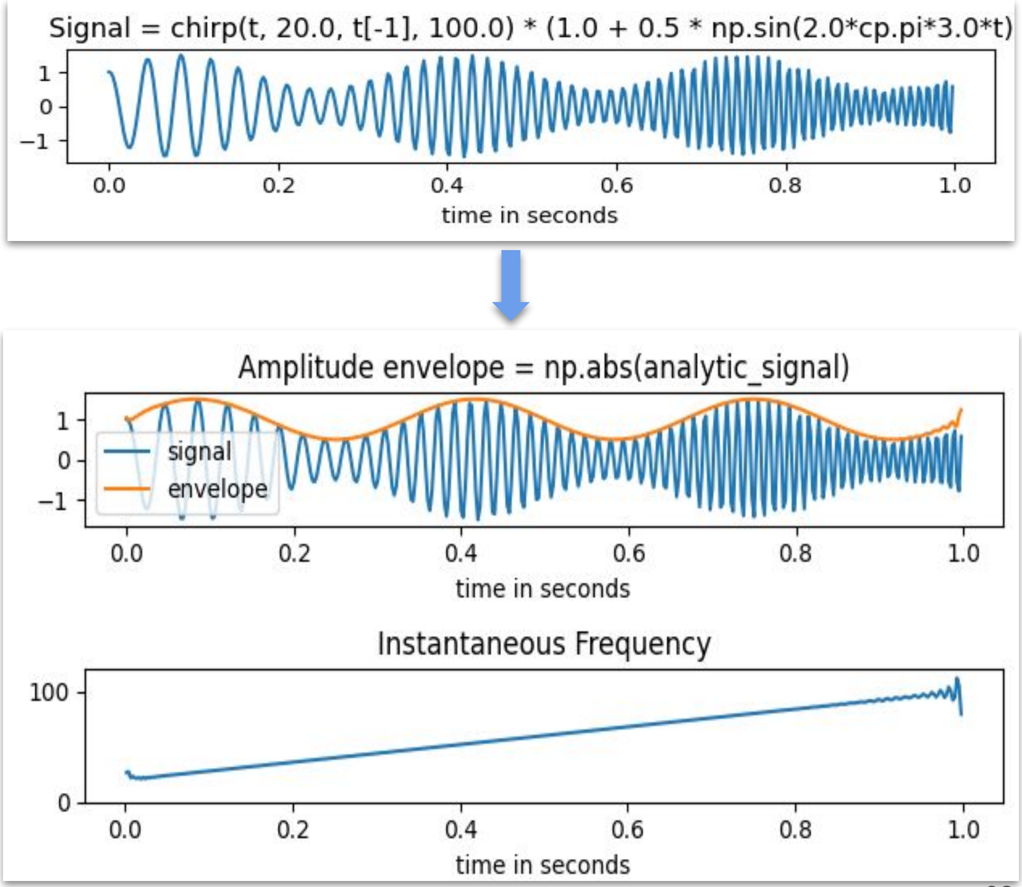


Outline

- 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

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

2] Rotate the Fourier Coefficients to obtain imaginary part

```
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
```

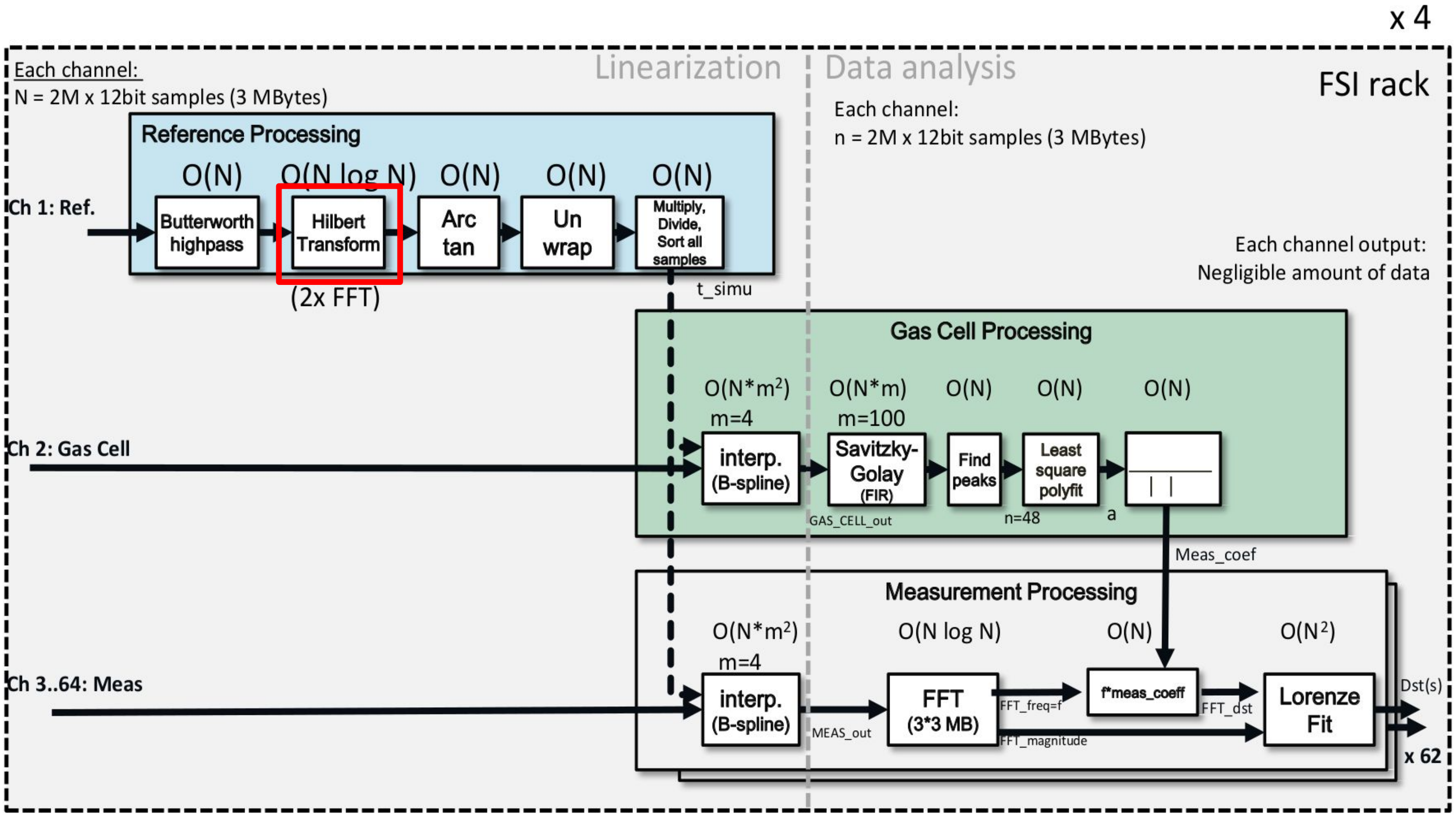
```
if x.ndim > 1:
      ind = [cp.newaxis] *
x.ndim
      ind[axis] = slice(None)
      h = h[tuple(ind)]
```

3] Compute Inverse Fourier Transform to get the Analytic Signal

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

4] Calculate Instantaneous Frequency and phase

Hilbert Transform

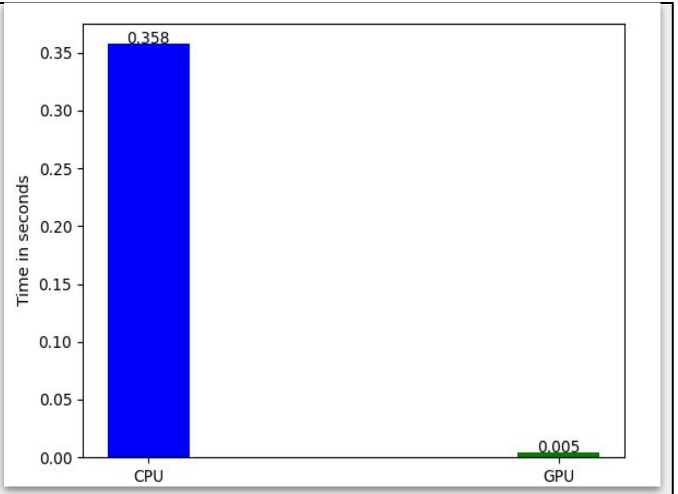


Hilbert Transform of Reference Cell Data

```
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
```

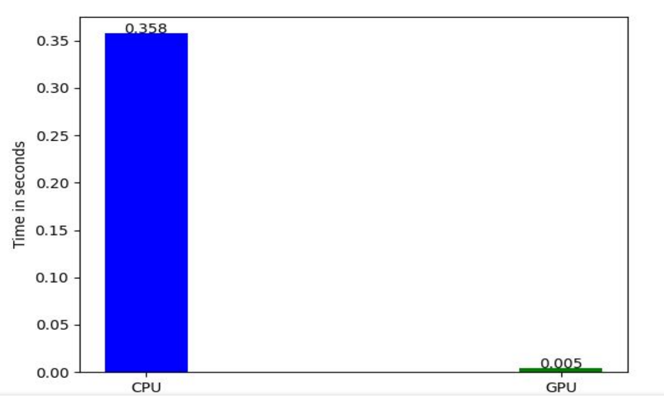
Performance Analysis: Hilbert Transform

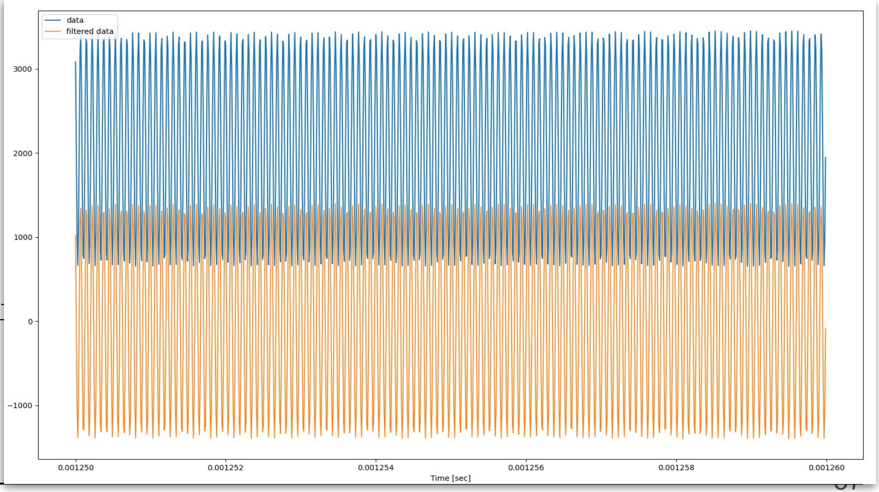
```
def DataLinearize(Tinterval, REF_IFM, plot='false') :

      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
```



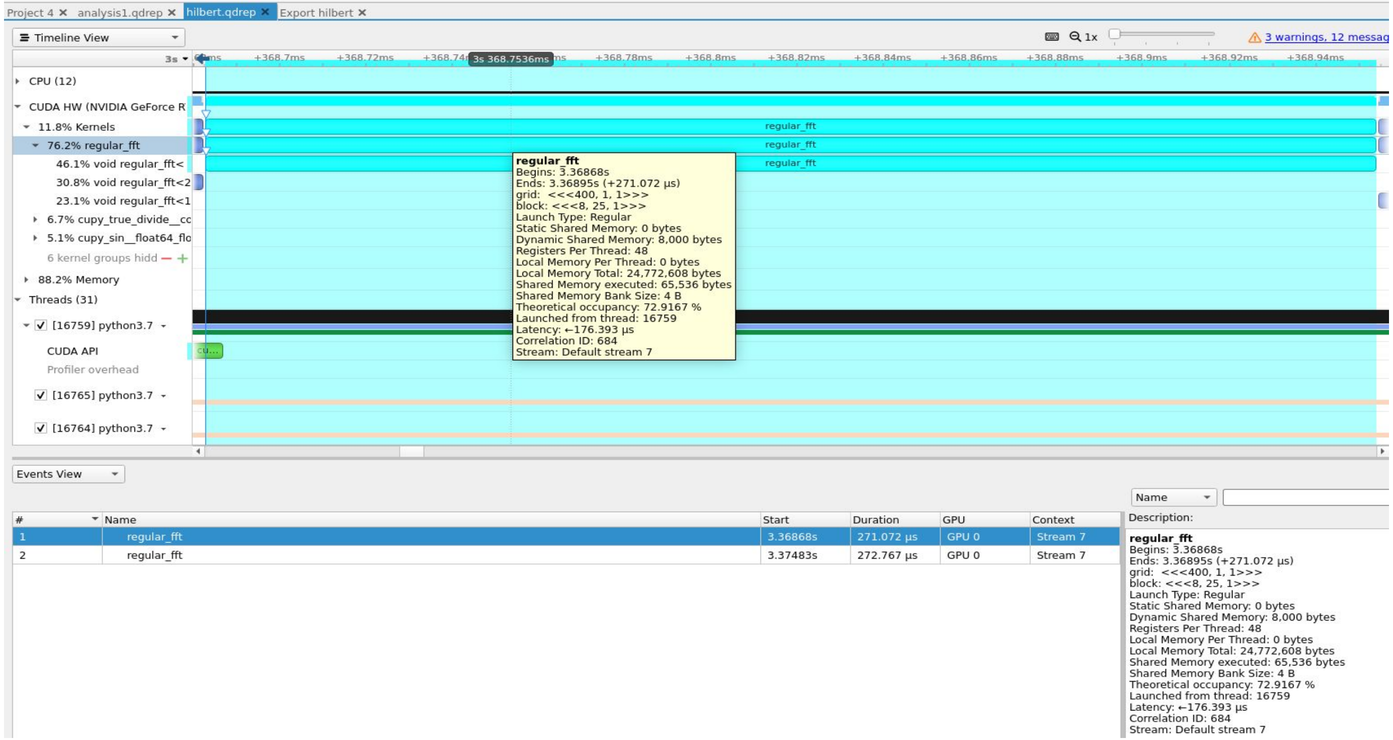
Performance Analysis: Hilbert Transform

```
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
      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

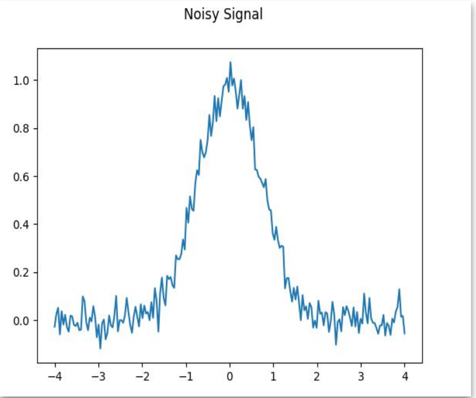
- Frequency Scanning Interferometry System
- Signal Processing in FSI
- CuPy and Signal Processing
  - Butterworth Filter
  - Hilbert Transform
  - Savitzky-Golay Filter
- Outlook

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.

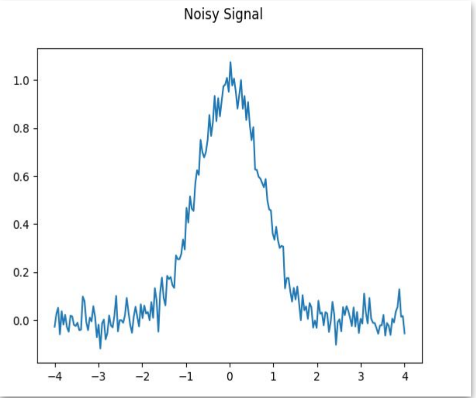
- Find least-square fit for each window and replace each data point with coefficient of that polynomial

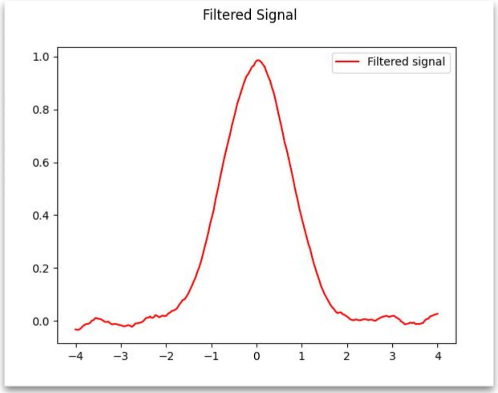
- But the smoothed output obtained by fitting polynomial to each window is equivalent to convolution of convolution coefficients(weighting coefficients) with each window/segment

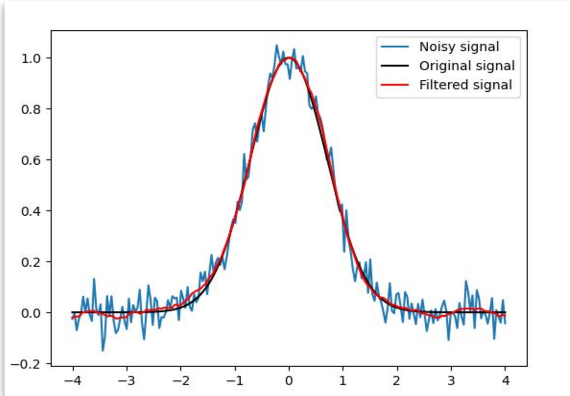


Savitzky-Golay Filter

- 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 = cp.array([[k**i for i in range(order+1)] for k in range(-half_window, half_window+1)])
m = cp.linalg.pinv(b)
m = cp.multiply(m[deriv] , cp.multiply(cp.power(rate,deriv), factorial(deriv)))
```

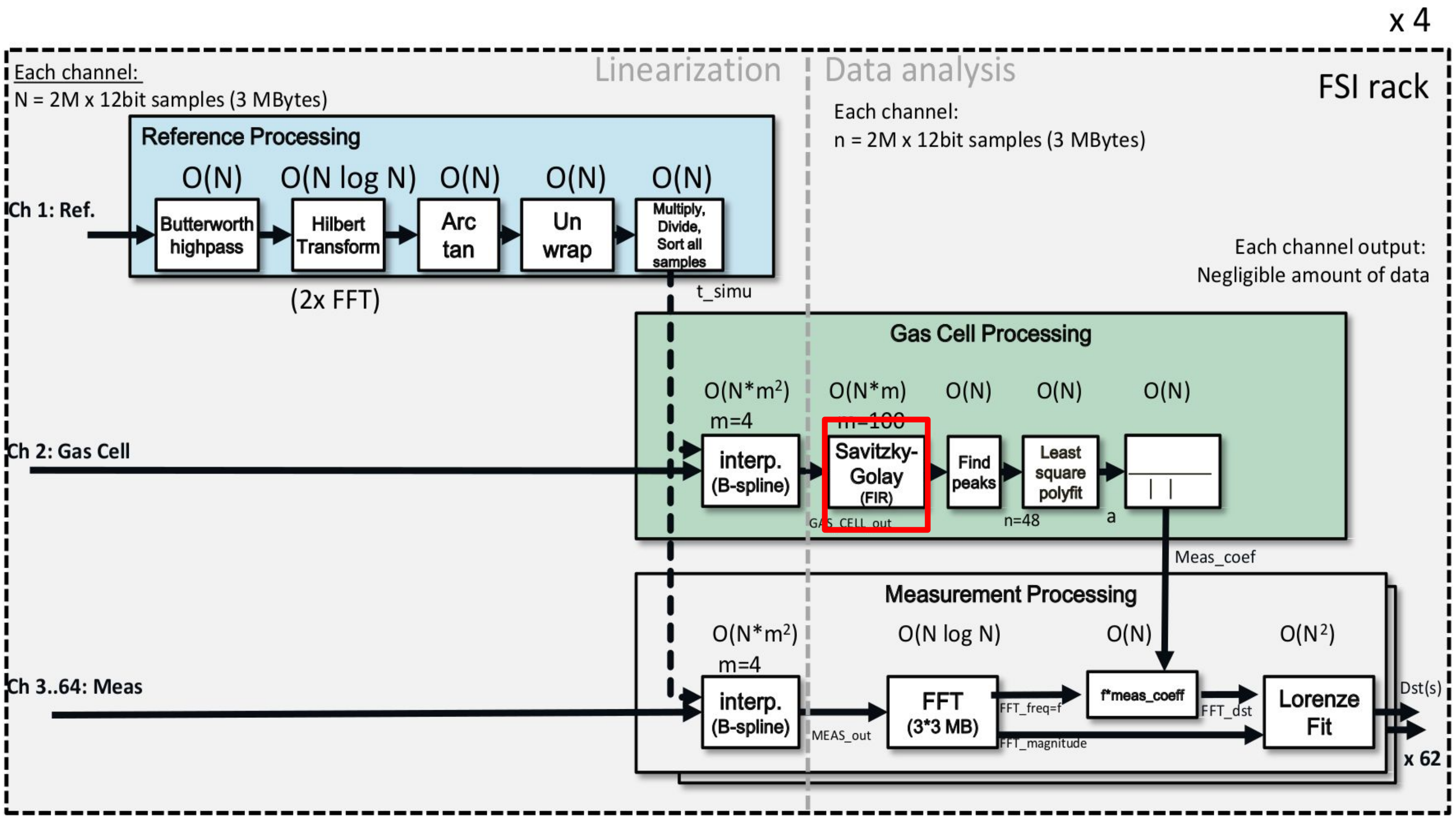
2] Pad the signal at the extremes

```
extr_begin = y[0] - cp.abs( y[1:half_window+1][::-1] - y[0] )
extr_end   = y[-1] + cp.abs(y[-half_window-1:-1][::-1] - y[-1])
y = cp.concatenate((extr_begin, y, extr_end))
```

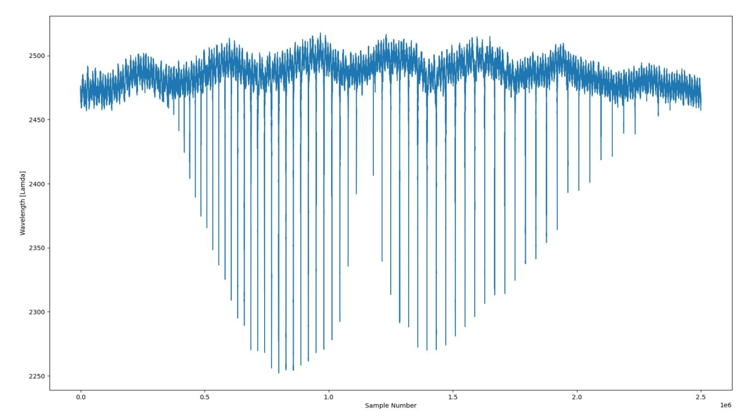
3] Convolve signal with calculated coefficients

```
result = cp.convolve( m[::-1], y, mode='valid')
```

Applying Savitzky-Golay Filter to Gas Cell Data

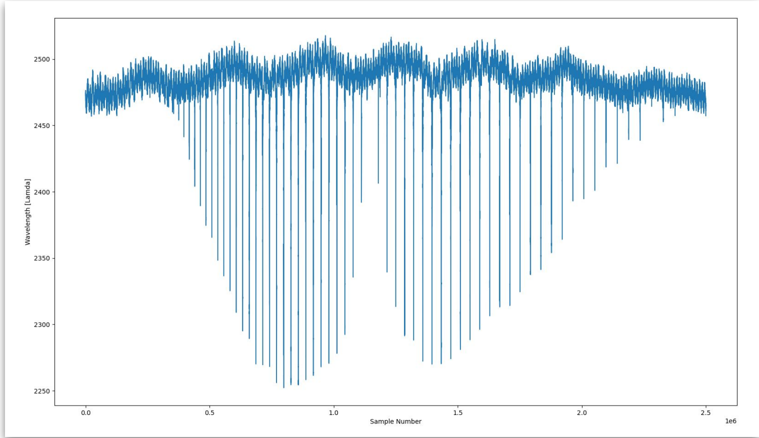


Applying Savitzky-Golay Filter to Gas Cell Data

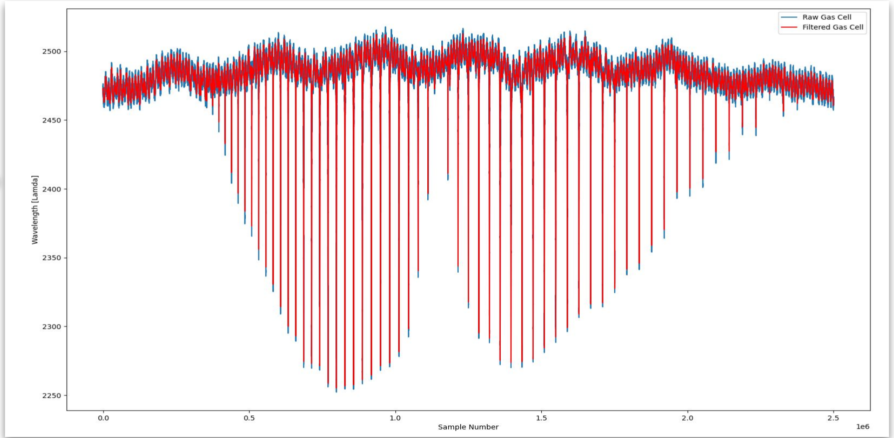


Spectrum of Hydrogen Cyanide (HCN) SRM2519a absorption gas cell -used to track the “true” frequency of the sweeping laser

Applying Savitzky-Golay Filter to Gas Cell Data

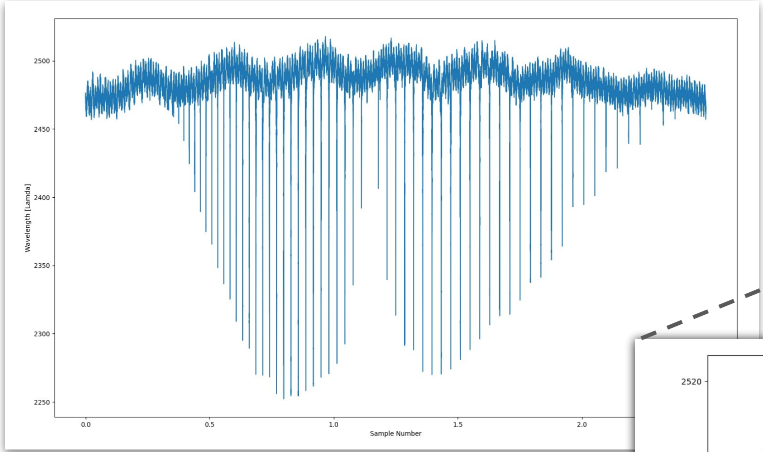


Gas Cell Spectrum

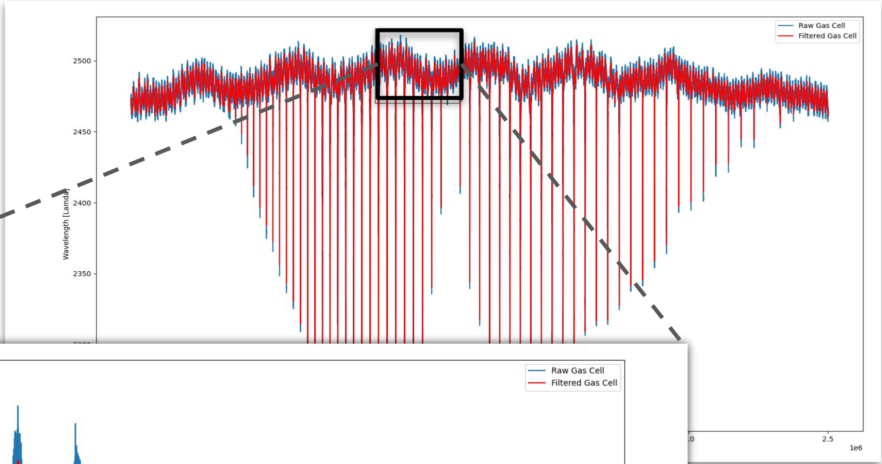


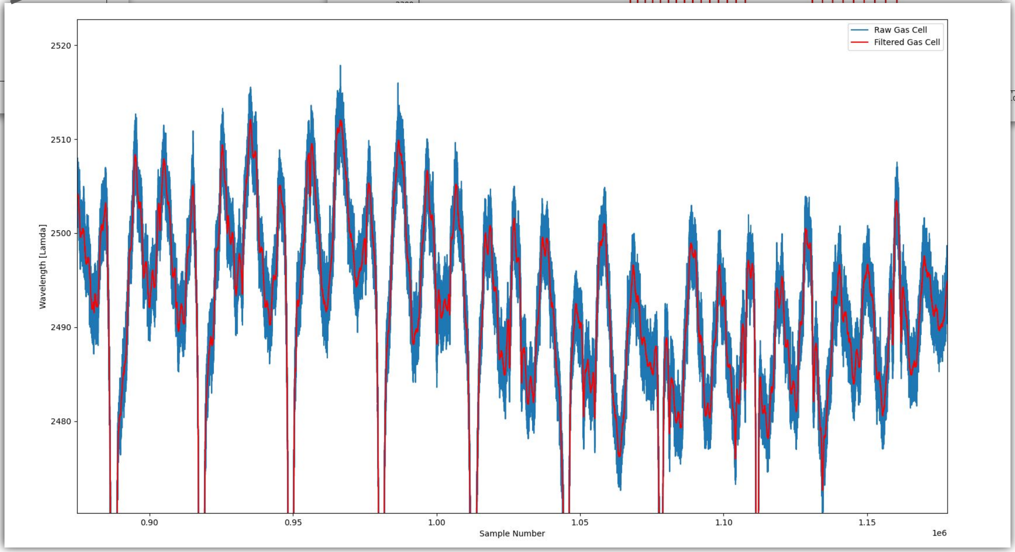
Filtered Gas Cell

Applying Savitzky-Golay Filter to Gas Cell Data



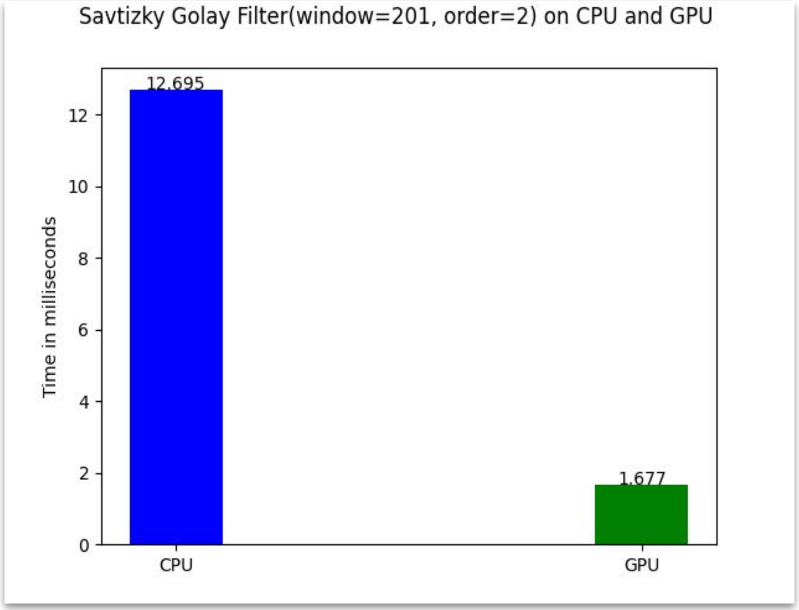
Gas Cell Spectrum





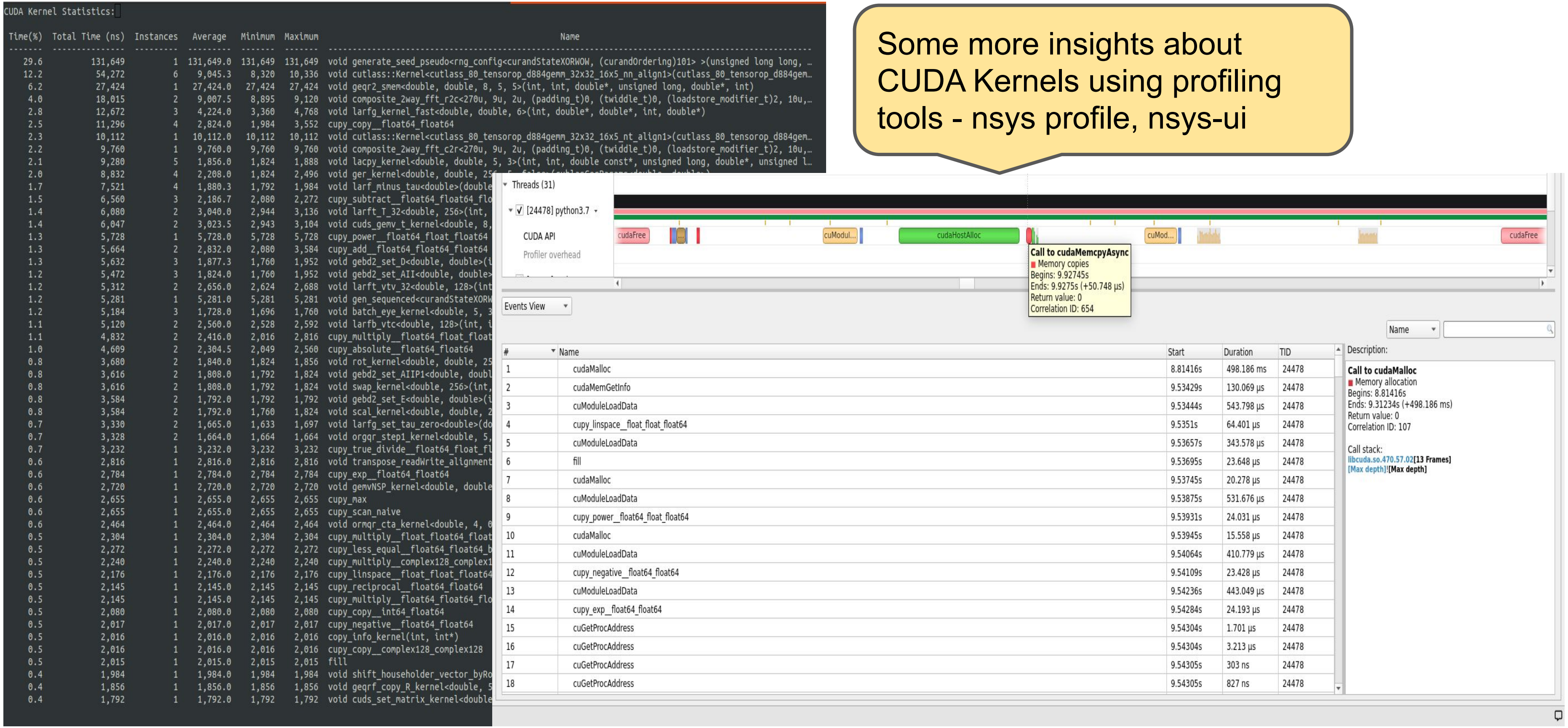
Performance Analysis: Savitzky-Golay Filter

```
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

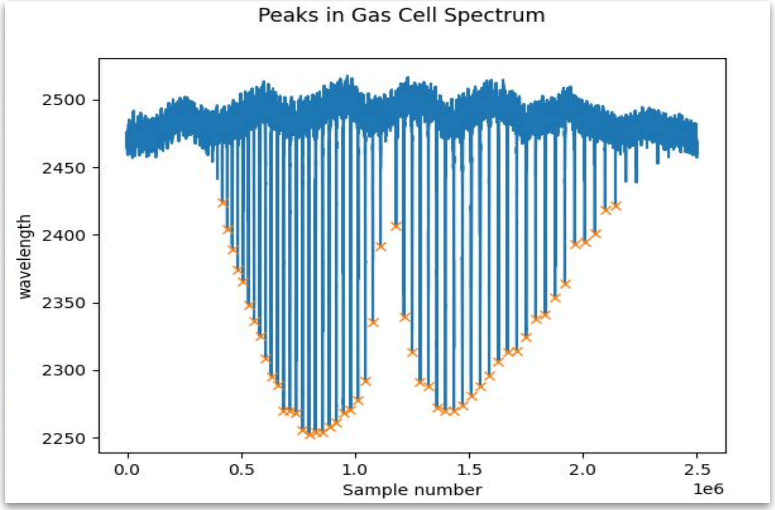
Some more insights about CUDA Kernels using profiling tools - nsys profile, nsys-ui

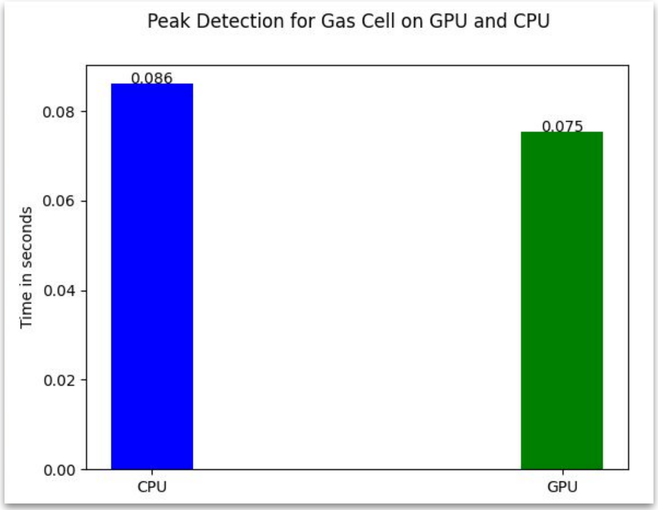


CUDA Kernel Statistics:

Time(%)	Total Time (ns)	Instances	Average	Minimum	Maximum	Name
29.6	131,649	1	131,649.0	131,649	131,649	void generate_seed_pseudo<rng_config<curandStateXORWOW, (curandOrdering)101> >(unsigned long long, …
12.2	54,272	6	9,045.3	8,320	10,336	void cutlass::Kernel<cutlass_80_tensorop_d884gemm_32x32_16x5_nn_align1>(cutlass_80_tensorop_d884gem…
6.2	27,424	1	27,424.0	27,424	27,424	void geqr2_smem<double, double, 8, 5, 5>(int, int, double*, unsigned long, double*, int)
4.0	18,015	2	9,007.5	8,895	9,120	void composite_2way_fft_r2c<270u, 9u, 2u, (padding_t)0, (twiddle_t)0, (loadstore_modifier_t)2, 10u,…
2.8	12,672	3	4,224.0	3,360	4,768	void larfg_kernel_fast<double, double, 6>(int, double*, double*, int, double*)
2.5	11,296	4	2,824.0	1,984	3,552	cupy_copy__float64_float64
2.3	10,112	1	10,112.0	10,112	10,112	void cutlass::Kernel<cutlass_80_tensorop_d884gemm_32x32_16x5_nt_align1>(cutlass_80_tensorop_d884gem…
2.2	9,760	1	9,760.0	9,760	9,760	void composite_2way_fft_c2r<270u, 9u, 2u, (padding_t)0, (twiddle_t)0, (loadstore_modifier_t)2, 10u,…
2.1	9,280	5	1,856.0	1,824	1,888	void lacpy_kernel<double, double, 5, 3>(int, int, double const*, unsigned long, double*, unsigned l…
2.0	8,832	4	2,208.0	1,824	2,496	void ger_kernel<double, double, 25
1.7	7,521	4	1,880.3	1,792	1,984	void larf_minus_tau<double>(double
1.5	6,560	3	2,186.7	2,080	2,272	cupy_subtract__float64_float64_flo
1.4	6,080	2	3,040.0	2,944	3,136	void larft_T_32<double, 256>(int,
1.4	6,047	2	3,023.5	2,943	3,104	void cuds_gemv_t_kernel<double, 8,
1.3	5,728	1	5,728.0	5,728	5,728	cupy_power__float64_float_float64
1.3	5,664	2	2,832.0	2,080	3,584	cupy_add__float64_float64_float64
1.3	5,632	3	1,877.3	1,760	1,952	void gebd2_set_D<double, double>(i
1.2	5,472	3	1,824.0	1,760	1,952	void gebd2_set_AII<double, double>
1.2	5,312	2	2,656.0	2,624	2,688	void larft_vtv_32<double, 128>(int
1.2	5,281	1	5,281.0	5,281	5,281	void gen_sequenced<curandStateXORW
1.2	5,184	3	1,728.0	1,696	1,760	void batch_eye_kernel<double, 5, 3
1.1	5,120	2	2,560.0	2,528	2,592	void larfb_vtc<double, 128>(int, i
1.1	4,832	2	2,416.0	2,016	2,816	cupy_multiply__float64_float_float
1.0	4,609	2	2,304.5	2,049	2,560	cupy_absolute__float64_float64
0.8	3,680	2	1,840.0	1,824	1,856	void rot_kernel<double, double, 25
0.8	3,616	2	1,808.0	1,792	1,824	void gebd2_set_AIP1<double, doubl
0.8	3,616	2	1,808.0	1,792	1,824	void swap_kernel<double, 256>(int,
0.8	3,584	2	1,792.0	1,792	1,792	void gebd2_set_E<double, double>(i
0.8	3,584	2	1,792.0	1,760	1,824	void scal_kernel<double, double, 2
0.7	3,330	2	1,665.0	1,633	1,697	void larfg_set_tau_zero<double>(do
0.7	3,328	2	1,664.0	1,664	1,664	void orgqr_step1_kernel<double, 5,
0.7	3,232	1	3,232.0	3,232	3,232	cupy_true_divide__float64_float_fl
0.6	2,816	1	2,816.0	2,816	2,816	void transpose_readWrite_alignment
0.6	2,784	1	2,784.0	2,784	2,784	cupy_exp__float64_float64
0.6	2,720	1	2,720.0	2,720	2,720	void gemvNSP_kernel<double, double
0.6	2,655	1	2,655.0	2,655	2,655	cupy_max
0.6	2,655	1	2,655.0	2,655	2,655	cupy_scan_naive
0.6	2,464	1	2,464.0	2,464	2,464	void ormqr_cta_kernel<double, 4, 0
0.5	2,304	1	2,304.0	2,304	2,304	cupy_multiply__float_float64_float
0.5	2,272	1	2,272.0	2,272	2,272	cupy_less_equal__float64_float64_b
0.5	2,240	1	2,240.0	2,240	2,240	cupy_multiply__complex128_complex1
0.5	2,176	1	2,176.0	2,176	2,176	cupy_linspace__float_float_float64
0.5	2,145	1	2,145.0	2,145	2,145	cupy_reciprocal__float64_float64
0.5	2,145	1	2,145.0	2,145	2,145	cupy_multiply__float64_float64_flo
0.5	2,080	1	2,080.0	2,080	2,080	cupy_copy__int64_float64
0.5	2,017	1	2,017.0	2,017	2,017	cupy_negative__float64_float64
0.5	2,016	1	2,016.0	2,016	2,016	copy_info_kernel(int, int*)
0.5	2,016	1	2,016.0	2,016	2,016	cupy_copy__complex128_complex128
0.5	2,015	1	2,015.0	2,015	2,015	fill
0.4	1,984	1	1,984.0	1,984	1,984	void shift_householder_vector_byRo
0.4	1,856	1	1,856.0	1,856	1,856	void geqrf_copy_R_kernel<double, 5
0.4	1,792	1	1,792.0	1,792	1,792	void cuds_set_matrix_kernel<double

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.

```
def detect_peaks_gpu(x, mph=None, mpd=1, threshold=0)
```

Some more Signal Processing Routines

A comparison of cupy and numpy implementations on 2.5 million data points sample(time in seconds) :

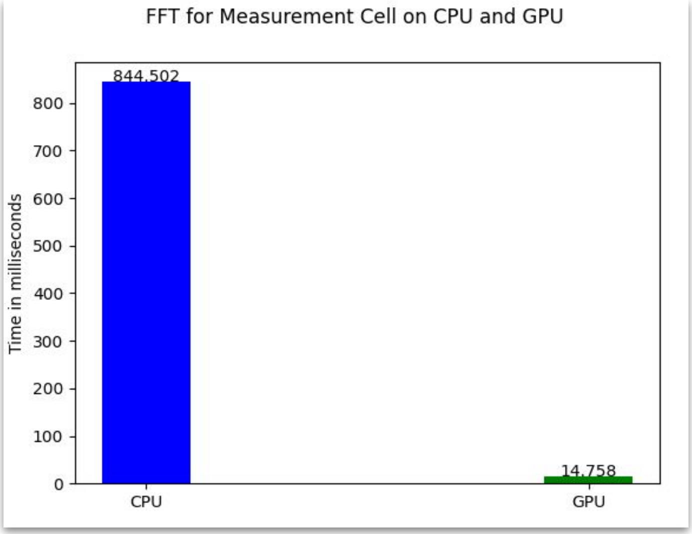
Routines		Numpy	CuPy	Speed
Interpolation:	np.interp -> cp.interp	0.055173	0.001341	41.4x
Unwrap:	np.unwrap->cp.unwrap	0.143882	0.015522	9.2x
Convolution:	np.convolve->cp.convolve	0.326742	0.014102	23.1x
Angle:	np.angle-> cp.angle	0.165760	0.004315	38.41x
Sort:	np.sort->cp.sort	0.071232	0.002608	27.3x
Absolute:	np.abs->cp.abs	0.005381	0.004910	1.09x

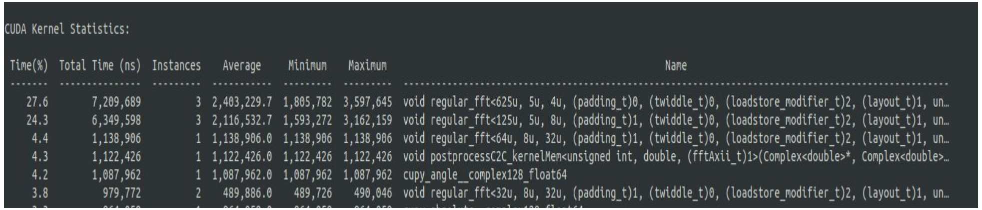
Fast-Fourier Transform in CuPy vs SciPy

cupyx.scipy.fft(x[, n, axis, norm, overwrite_x, plan])

- access advanced routines that cuFFT like get_plan_fft()
- Improve performance and behavior of the FFT routines - *https://docs.cupy.dev/en/stable/user_guide/fft.html*

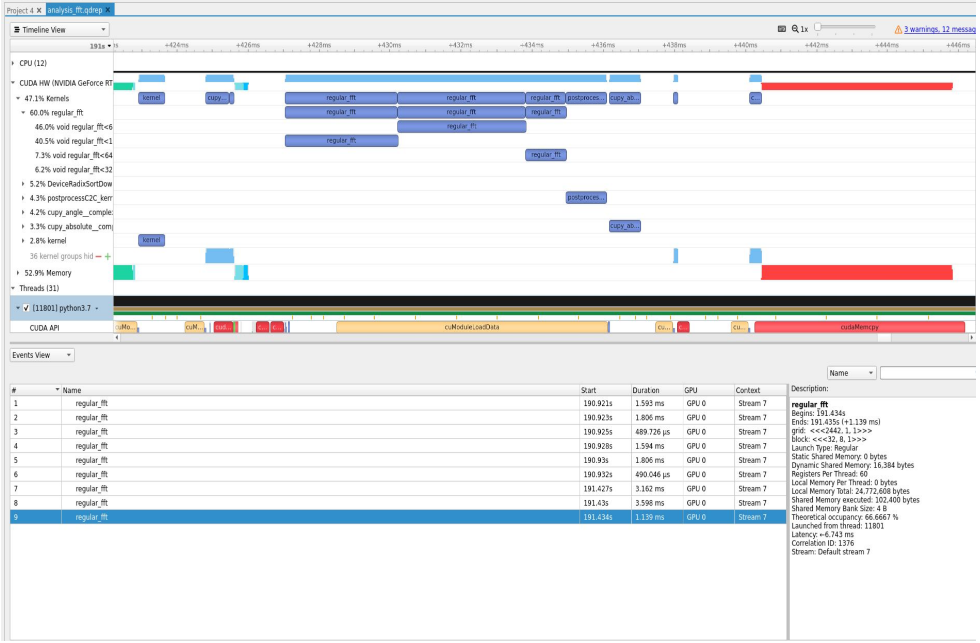
```
Y = cp.fft.rfft(Meas_Linear, int(len(Meas_Linear)))
```





CUDA Kernel Statistics:

Time(%)	Total Time (ns)	Instances	Average	Minimum	Maximum	Name
27.6	7,209,689	3	2,403,229.7	1,805,782	3,597,645	void regular_fft<625u, 5u, 4u, (padding_t)0, (twiddle_t)0, (loadstore_modifier_t)2, (layout_t)1, un…
24.3	6,349,598	3	2,116,532.7	1,593,272	3,162,159	void regular_fft<125u, 5u, 8u, (padding_t)1, (twiddle_t)0, (loadstore_modifier_t)2, (layout_t)1, un…
4.4	1,138,906	1	1,138,906.0	1,138,906	1,138,906	void regular_fft<64u, 8u, 32u, (padding_t)1, (twiddle_t)0, (loadstore_modifier_t)2, (layout_t)1, un…
4.3	1,122,426	1	1,122,426.0	1,122,426	1,122,426	void postprocessC2C_kernelMem<unsigned int, double, (fftAxii_t)1>(Complex<double>*, Complex<double>…
4.2	1,087,962	1	1,087,962.0	1,087,962	1,087,962	cupy_angle__complex128_float64
3.8	979,772	2	489,886.0	489,726	490,046	void regular_fft<32u, 8u, 32u, (padding_t)1, (twiddle_t)0, (loadstore_modifier_t)2, (layout_t)1, un…



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