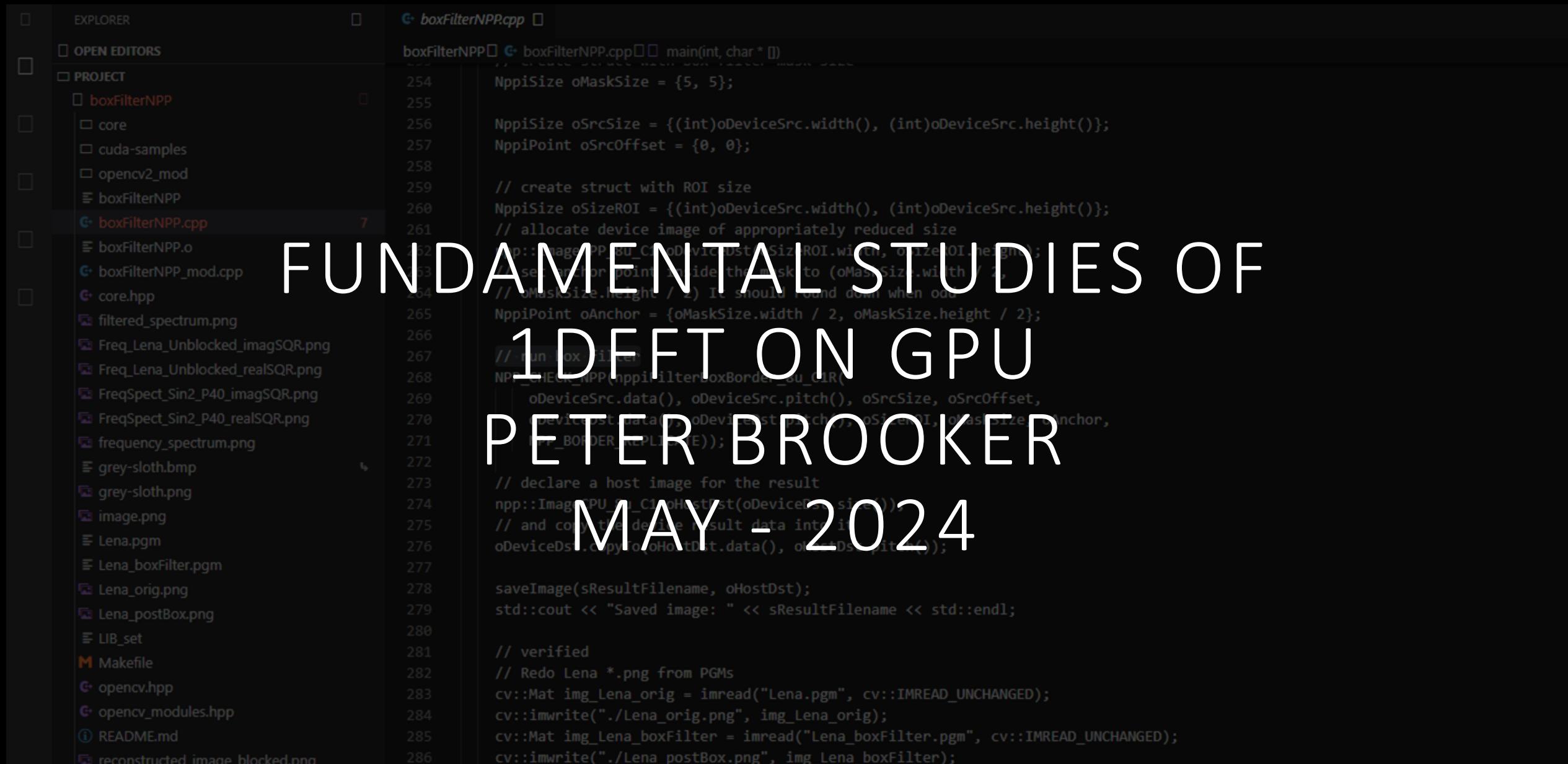


FUNDAMENTAL STUDIES OF 1DFFT ON GPU

PETER BROOKER

MAY - 2024

2. Using the cuFFT API

This chapter provides a general overview of the cuFFT library API. For more cc
Reference. Users are encouraged to read this chapter before continuing wit

The Discrete Fourier transform (DFT) maps a complex-valued vector x_k (*time*

$$X_k = \sum_{n=0}^{N-1} x_n e^{-2\pi i \frac{kn}{N}}$$

where X_k is a complex-valued vector of the same size. This is known as a *forv*
the transform is an *inverse* transform. Depending on N, different algorithms

The cuFFT API is modeled after FFTW, which is one of the most popular and
configuration mechanism called a *plan* that uses internal building blocks to op
GPU hardware selected. Then, when the *execution* function is called, the actu
advantage of this approach is that once the user creates a plan, the library re
without recalculation of the configuration. This model works well for cuFFT be
and GPU resources, and the plan interface provides a simple way of reusing c

Computing a number `BATCH` of one-dimensional DFTs of size `NX` using cuFF

```
#define NX 256
#define BATCH 10
#define RANK 1
...
{
    cufftHandle plan;
    cufftComplex *data;
    ...
    cudaMalloc((void**)&data, sizeof(cufftComplex)*NX*BATCH);
    cufftPlanMany(&plan, RANK, NX, &iembed, istride, idist,
        &oembed, ostride, odist, CUFFT_C2C, BATCH);
    ...
    cufftExecC2C(plan, data, data, CUFFT_FORWARD);
    cudaDeviceSynchronize();
    ...
    cufftDestroy(plan);
    cudaFree(data);
}
```

2.1. Accessing cuFFT

The cuFFT and cuFFTW libraries are available as shared libraries. They consist

OUTLINE

- Purpose of freq spectrum study
- Fast Fourier Transform Background
- Single frequency test signals
- Top Hat input signal test case
- Top Hat test case with high frequencies blocked
- Batch signal FFT processing demonstration
- Conclusion

PURPOSE OF FREQ STECTRUM STUDY

FUNDAMENTAL STUDIES OF 1DFFT ON GPU

PURPOSE OF FREQ SPECTRUM STUDY

- It is possible to create 1D signal test cases where the FFT sum can be explicitly performed.
- Analyze the 1D freq spectrum output for these known test cases to gain a fundamental understanding of the form of the 1D frequency spectrum output by 1D cuFFT
- Use this knowledge to create a frequency filter to block out all but the lowest frequencies of a top hat input in order to reproduce the expected ringing in the inverse FFT of the freq spectrum after the filter

FAST FOURIER TRANSFORM BACKGROIUND

FFT PROCESSING ADDED TO BOXFILTERNPP.CPP

DFT MATH FOR SPATIAL FOURIER TRANSFORM

- i = SQRT(-1)
- k = spatial frequency = 1/Lambda
- Definition:
  - x_j= 0,1,…,N-1
- Defn: Smallest λ = 1
  - k=1 ← largest k
  - (should actually be 4)
  - (0,pi/4,pi/2,3pi/2,2pi)
- Largest lambda = N-1
  - k = 1/(N-1)←smallest k
- Definition:
  - k_m = 0/N,1/N,…,(N-1)/N

$$F(k)=\int_{-\infty}^{\infty} f(x)\, e^{-2\pi i k x}\, dx.$$

mathworld.wolfram.com/FourierTransform.html

$$F(k) \sim F(k_m) = \sum_{j=0}^{N-1} \Delta x_j * f(x_j) * e^{-i2\pi k_m x_j}$$

$$= \sum_{j=0}^{N-1} (1) * f(x_j) * e^{-i2\pi \frac{m}{N} j}$$

Let f(xj) = $e^{+i2\pi \frac{p}{N} j}$

Then $F(k_m) = \sum_{j=0}^{N-1} e^{+i2\pi \frac{p}{N} j}\ e^{-i2\pi \frac{m}{N} j}$

$$= \sum_{j=0}^{N-1} e^{+i2\pi \frac{(p-m)}{N} j}$$

$$= \begin{cases} N, \ if\ p = m \\ 0, if\ p \neq m \\ (see\ geometric\ series) \end{cases}$$

MORE DFT MATH FOR SPATIAL FOURIER TRANSFORM

- Consider wavelength Lambda
- Cosine starts at 1 goes to -1 and then comes back to 1
- At least 3 pts are needed
- ➔ Lambda >= 2x sampling freq
- ➔ Lambda >= 2 grid pts
- k_m = 20 ➔ k=20*(1/40)=1/2
  - Lambda = 1/k = 1/(1/2)=2

- For n = N/2 + m, F(n) calculation is a slight modification of F(m) calculation

$$F(m) = \sum_{j=0}^{N-1} f(x_j) * e^{-i2\pi\frac{m}{N}j}$$

Consider m < N/2

$$F\left(\frac{N}{2}+m\right) = \sum_{j=0}^{N-1} f(x_j) * e^{-i2\pi\frac{\left(\frac{N}{2}+m\right)}{N}j}$$

$$= \sum_{j=0}^{N-1} f(x_j) * e^{-i2\pi\frac{\left(\frac{N}{2}+m\right)}{N}j}$$

$$= \sum_{j=0}^{N-1} f(x_j) e^{-i2\pi\frac{N}{2N}j} * e^{-i2\pi\frac{m}{N}j}$$

$$= \sum_{j=0}^{N-1} f(x_j) e^{-i\pi j} * e^{-i2\pi\frac{m}{N}j}$$

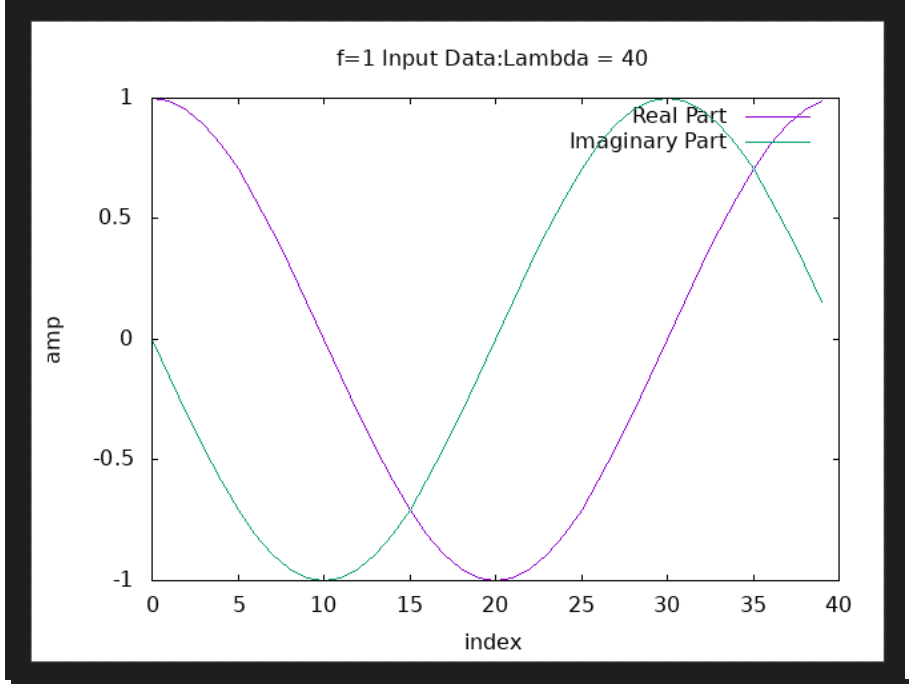
SINGLE FREQUENCY TEST SIGNALS

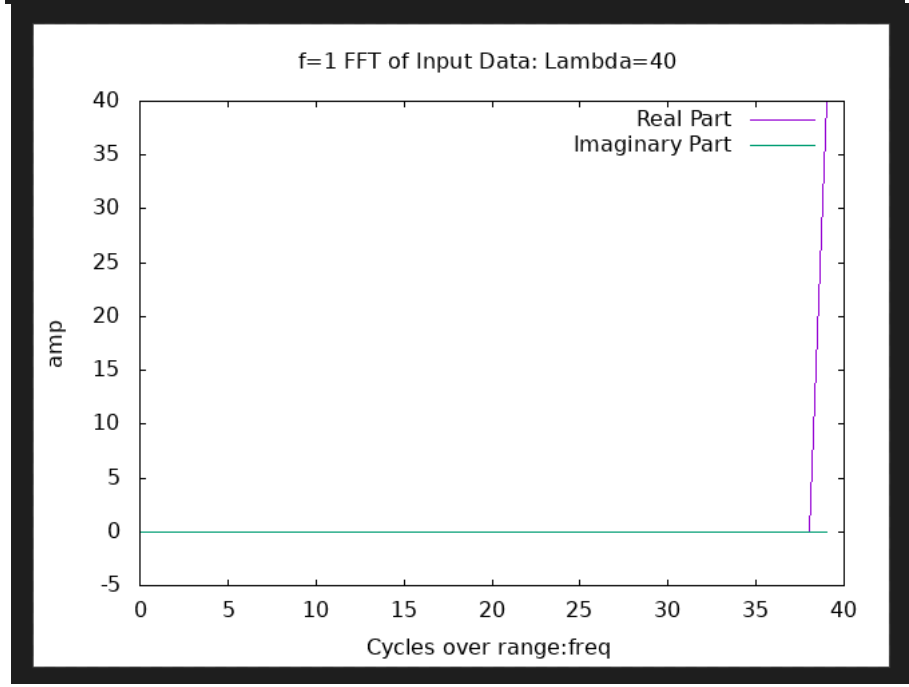
FUNDAMENTAL STUDIES OF 1DFFT ON GPU

```
f = 1; // Lambda = N/f = 40/1 = 40 <== 1 cycle over the range

// Define the input data on the host
for (int i = 0; i < N; ++i)
{
  h_data[i].x = cos(-2 * M_PI * f * i / N);
  h_data[i].y = sin(-2 * M_PI * f * i / N);
}
```

FREQ = 1, LAMBDA = 40

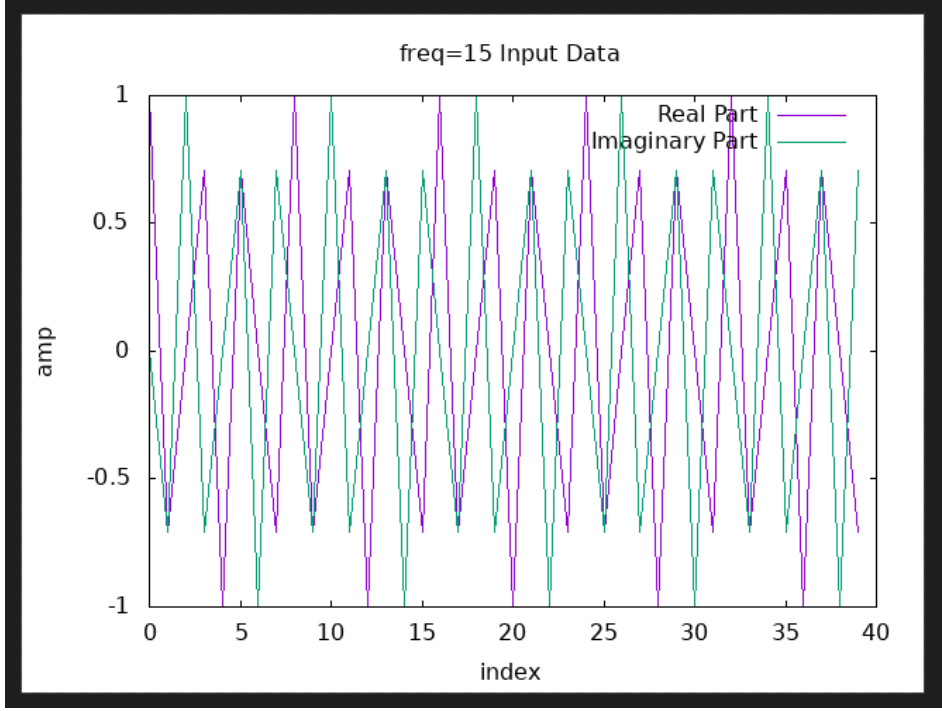
- Frequency = 1 means only one cycle over the range = 40
- Lowest frequency possible
- Only 1 lambda over range=40
- Highest frequencies are in the middle.
- Low freq are at the beginning and end.
- Only a low freq at freq=39=N - 1
- Amplitude of spectrum = 40 = N

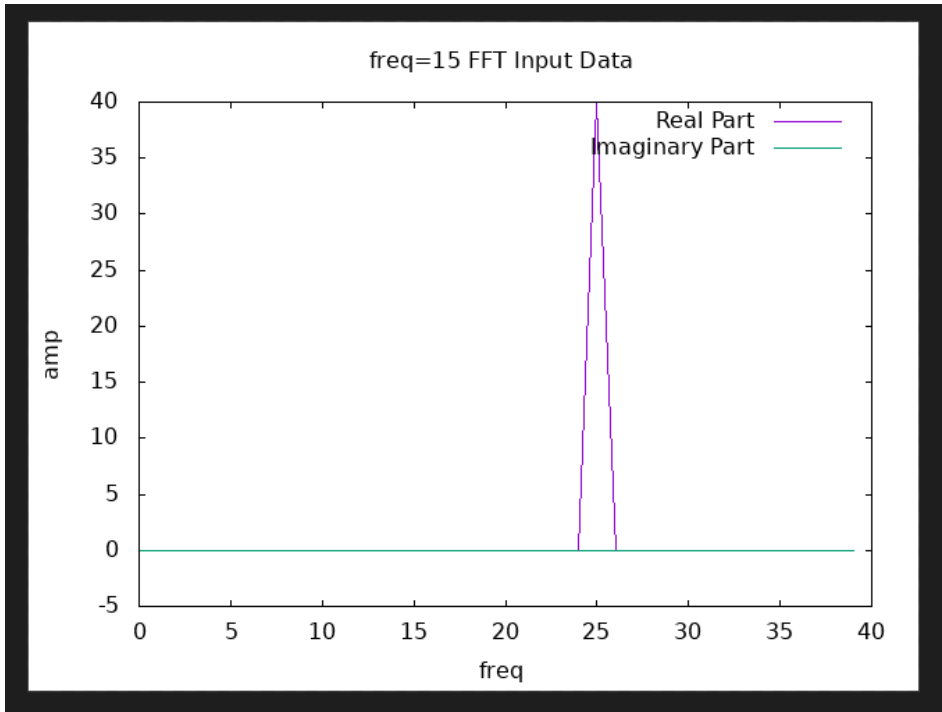




FREQ = 15, LAMBDA = 40/15

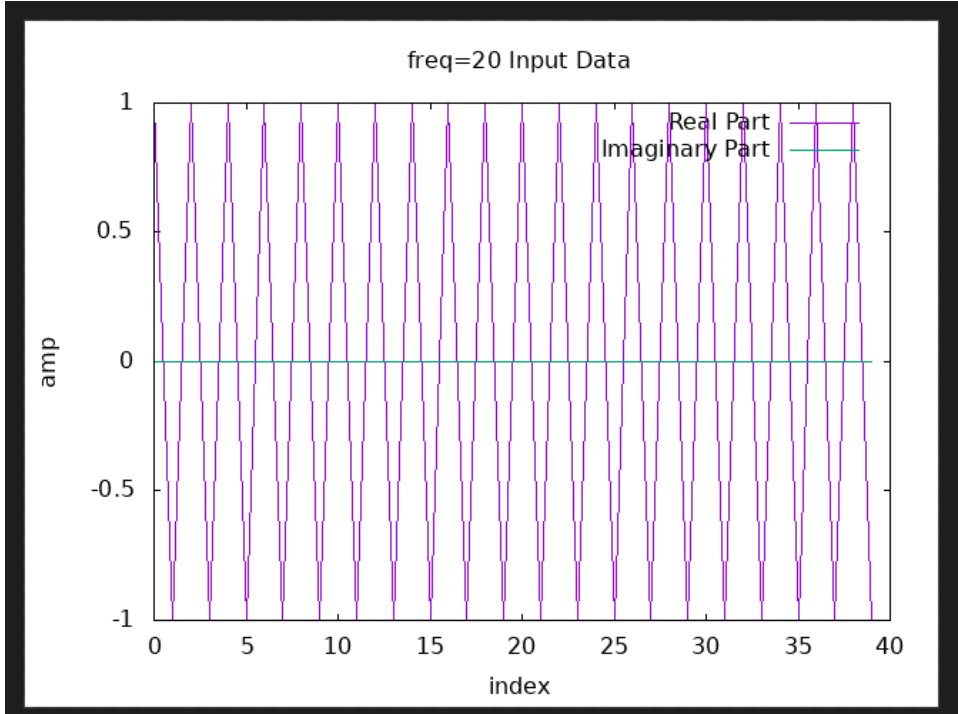
- Freq = 15 ➔ Lambda = N/f = 40/15 = 2.7
- Lambda is not on the grid. Plot looks terrible
- Only a single peak in freq at 25
- Freq is 5 away from middle of N/2 = 20
- Is this the pattern?:
  - N/2 – f + N/2 = 20 + (20 – 15)
  - = 20 + 5
  - = 25
- Why is there not a peak also at 15?

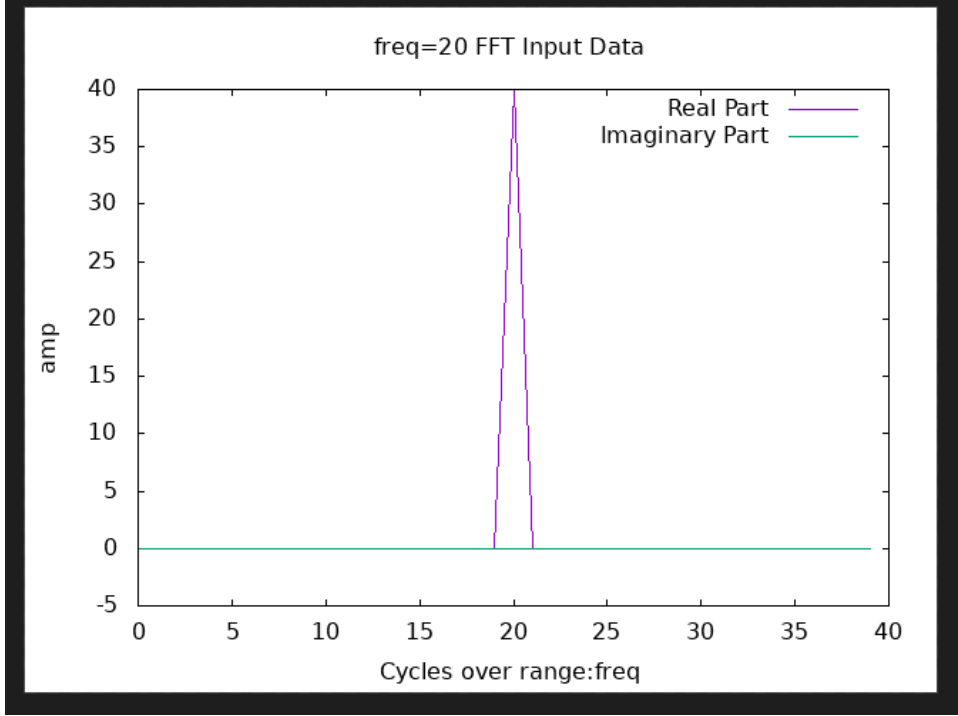




FREQ = 20, LAMBDA = 40/20 = 2 UNITS

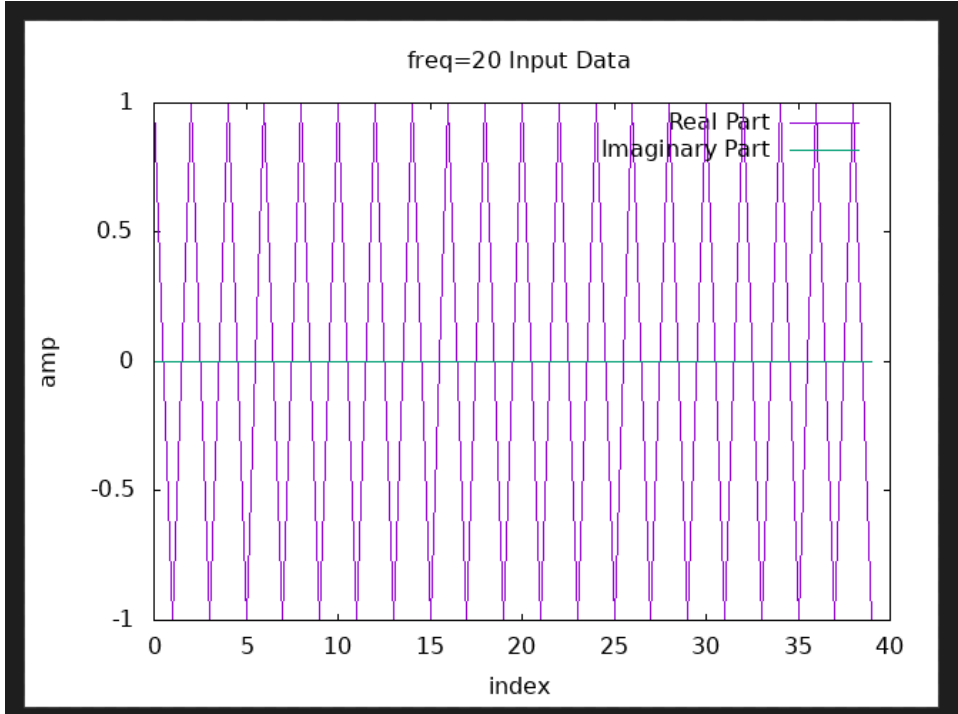
- Lambda fits on the grid.
- Imag = sin(theta) & sin theta is zero at grid pts
- Single peak in freq spectrum at f = 20, same as what was specified

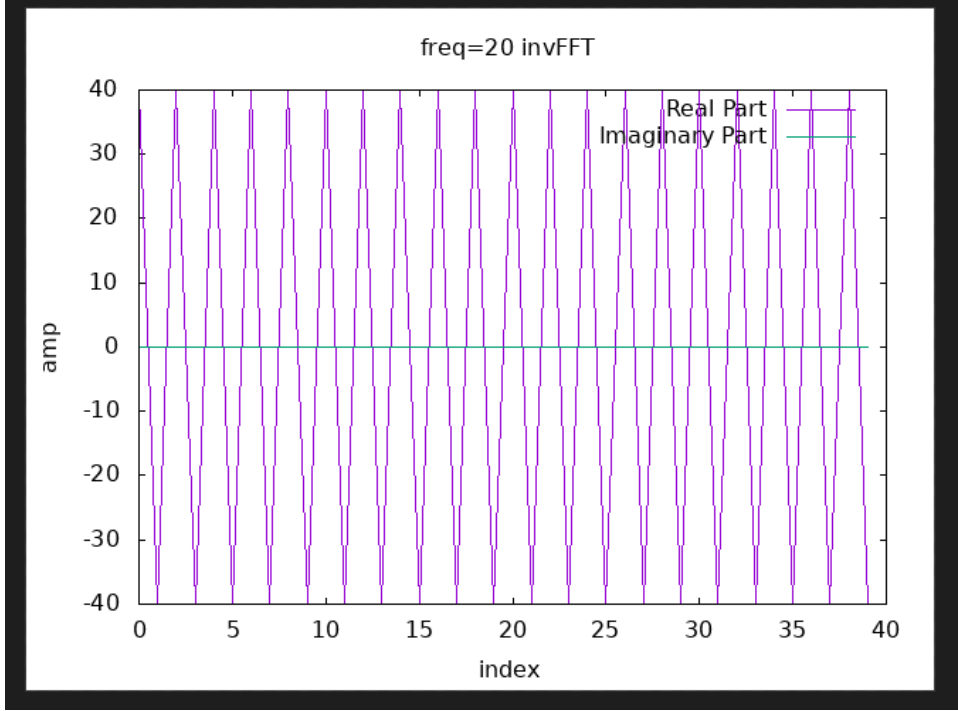




FREQ = 20, INPUT & INVERSE FFT

- Input signal goes from -1 to +1
- invFFT goes from -40 to + 40
- Cufft does not divide by N for invFFT
- invFFT is input signal * N.



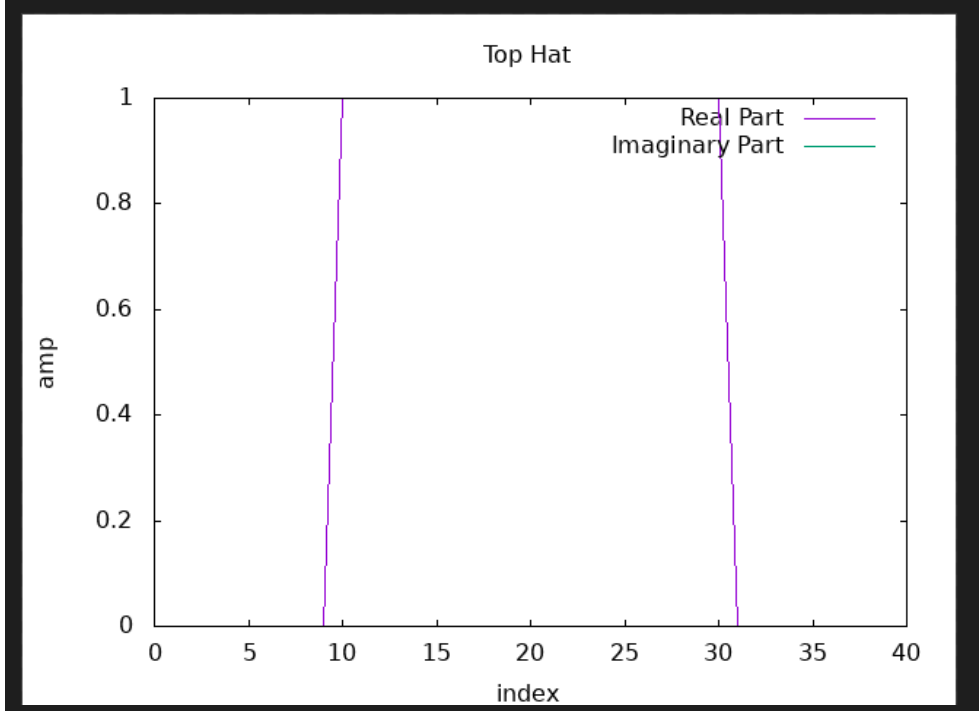


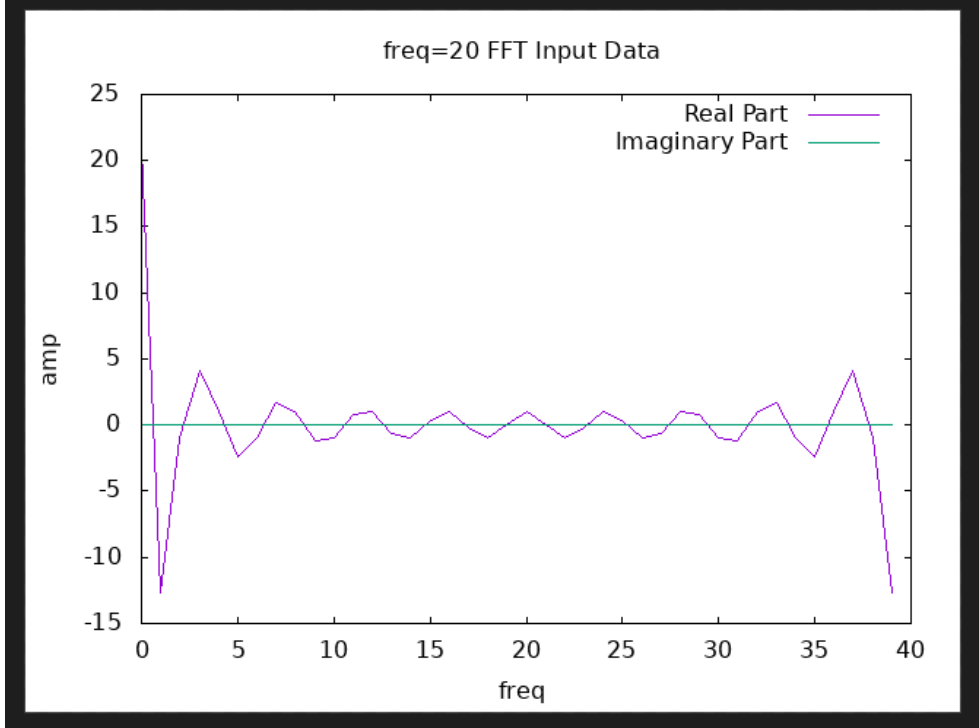
TOP HAT INPUT SIGNAL TEST CASE

FUNDAMENTAL STUDIES OF 1DFFT ON GPU

TOP HAT, WIDTH = 20

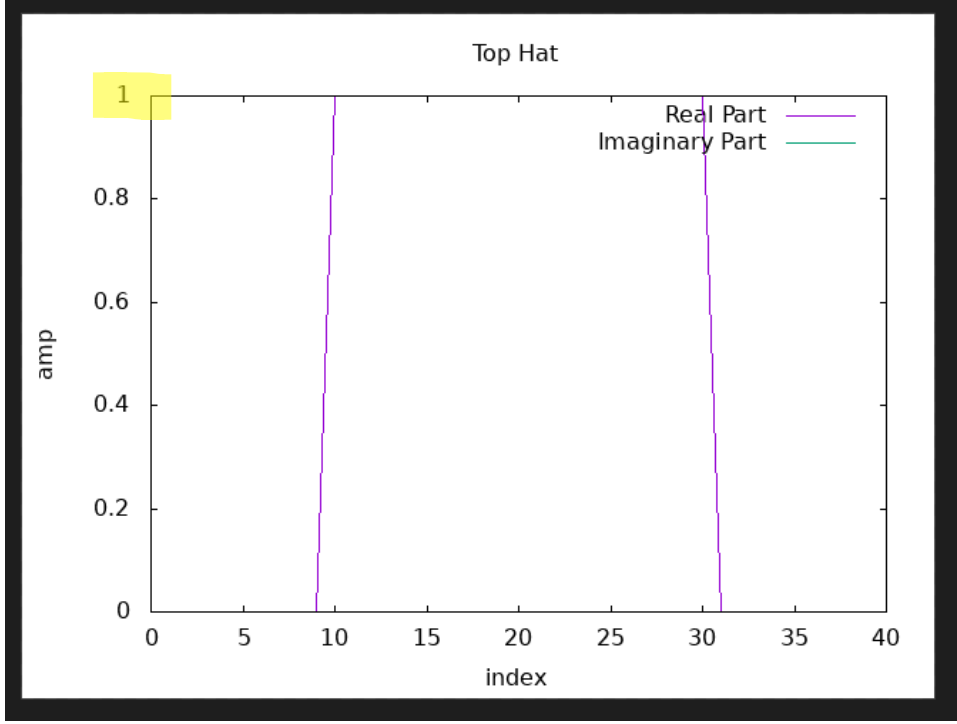
- Real Signal = 1 between 10 and 30 including 10 and 30
- Imag signal = 0 always
- Unlike previously, freq spectrum has values below freq = 20

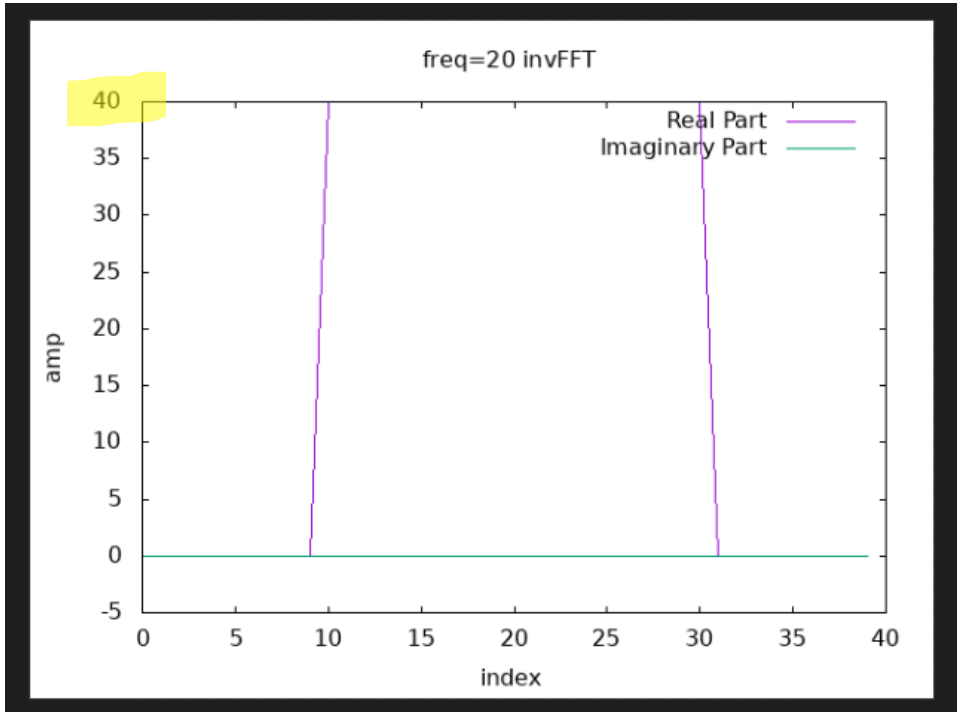




TOP HAT, WIDTH = 20, INVERSE FFT

- Input signal reproduced perfectly except for scale
- invFFT is the input signal multiplied by N=40
- cuFFT leaves out the divide by N





TOP HAT TEST CASE WITH HIGH FREQUENCIES BLOCKED

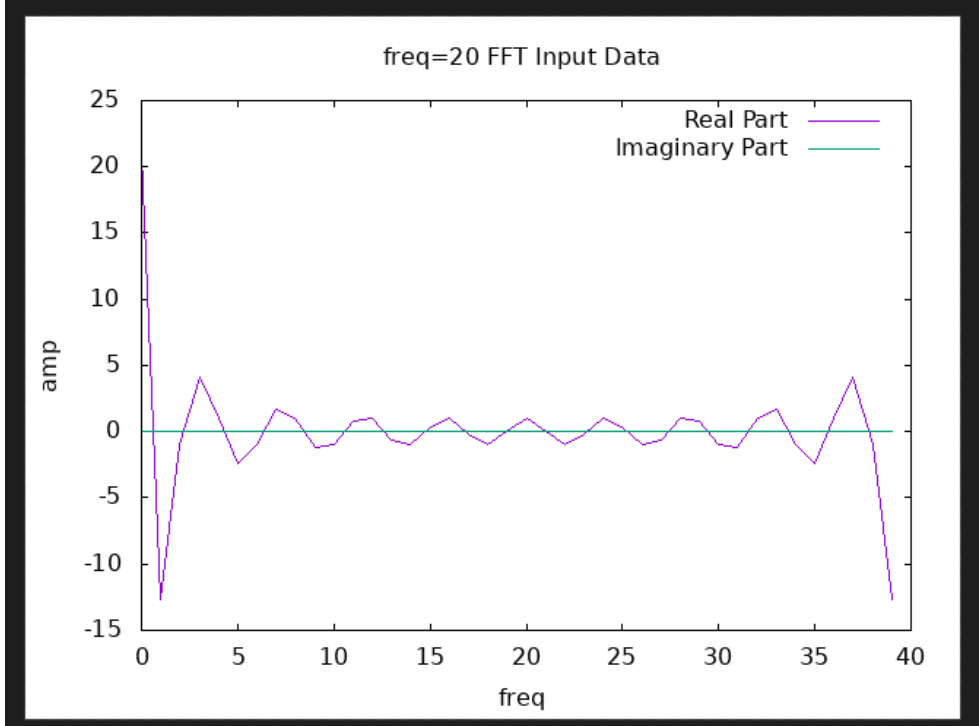
```
// Initialize filter array
for (int i = 0; i < N; ++i)
{
  if (i <= 5 || i >= 35)
  {
    h_filter[i].x = 1.0f;
    h_filter[i].y = 0.0f;
  }
  else
  {
    h_filter[i].x = 0.0f;
    h_filter[i].y = 0.0f;
  }
}
```

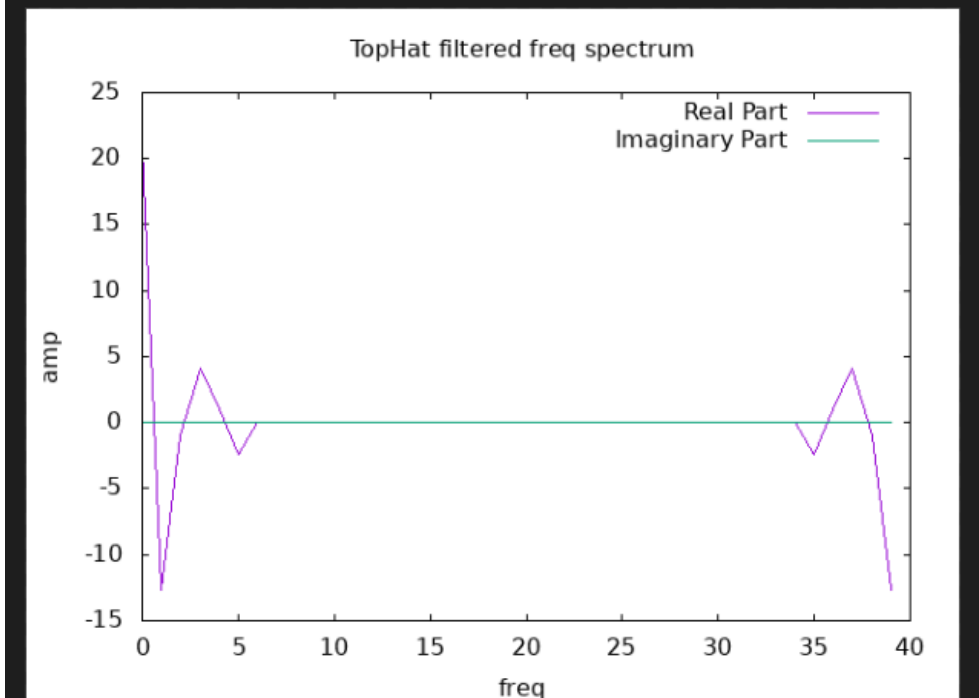
FUNDAMENTAL STUDIES OF 1DFFT ON GPU

TOP HAT HIGH FREQUENCY FILTER

- Frequencies are highest at N/2 = 20
- Blocked all frequencies between 5 & 35
- Copied h_filter to device and set up kernel for the multiply

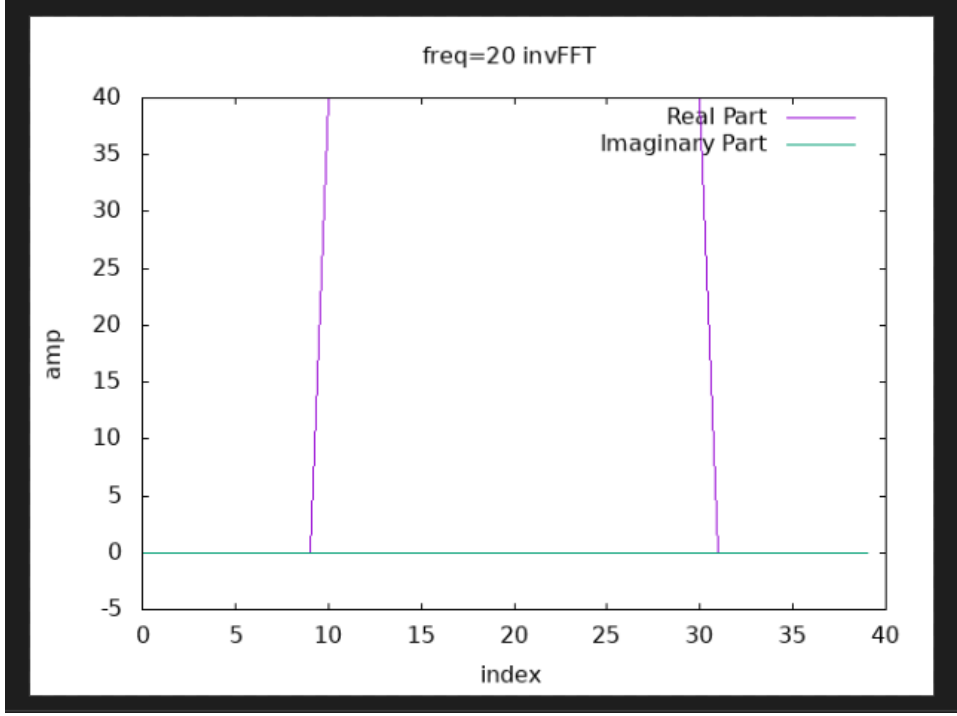
```
__global__ void applyFilter(cufftComplex *d_freq, const cufftComplex *filter)
{
  int idx = blockIdx.x * blockDim.x + threadIdx.x;

  // Multiply the real and imaginary parts of FFT data by the corresponding pa
  d_freq[idx].x *= filter[idx].x;
  d_freq[idx].y *= filter[idx].x; // Using filter[idx].x since filter[idx].y
}
```

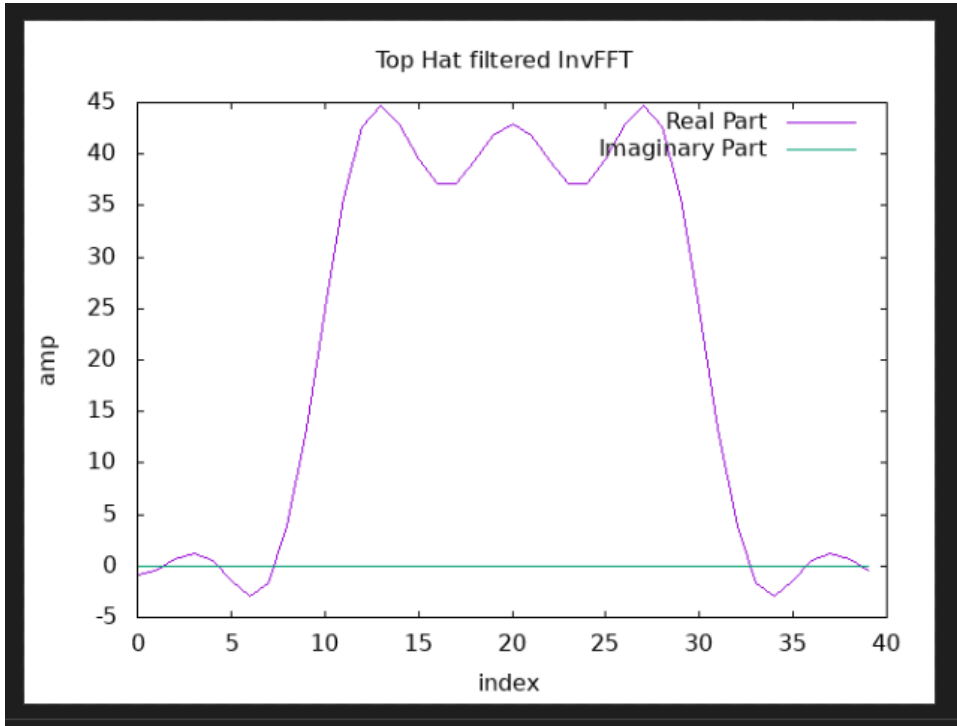




INV TOP HAT WITH AND WITHOUT HIGH FREQUENCY BLOCKING FILTER

- Blocking the higher frequencies generates the signal with ringing…as expected





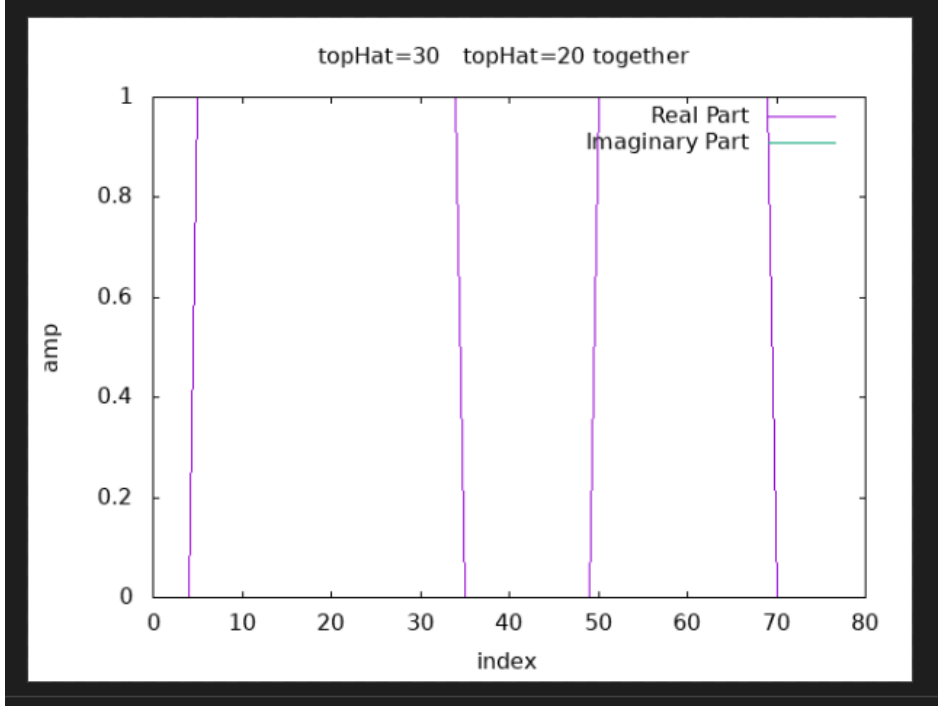
BATCH SIGNAL FFT PROCESSING DEMONSTRATION

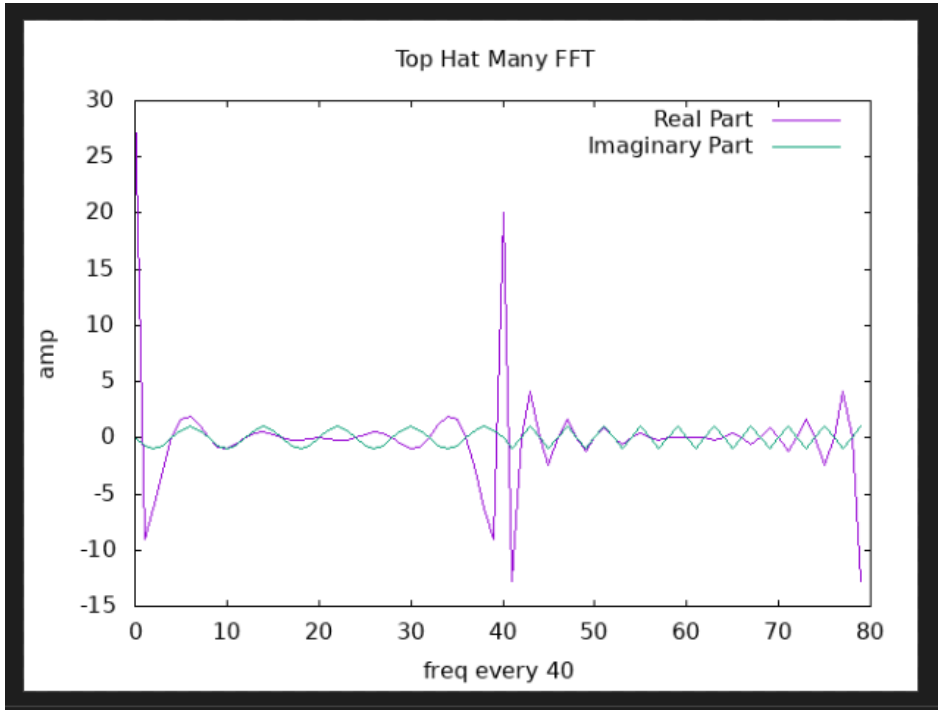
FUNDAMENTAL STUDIES OF 1DFFT ON GPU

```
const int batch = 2;                    // Number of FFTs to perform

// Create a cufftHandle for FFT plan
int n[1] = {N}; // Size of each dimension of the input data
cufftHandle planM;
cufftPlanMany(&planM, 1, n, nullptr, 1, N, nullptr, 1, N, CUFFT_C2C, batch);

// Execute FFT on the batch of signals
cufftExecC2C(planM, d_largeArray, d_largeArray, CUFFT_FORWARD); // Forward FF
```

PLAN MANY BATCH FFT

- Top Hat Width=30 and Top Hat width = 20 are placed in a single cufftComplex array
- cufftPlanMany is used instead of cufftPlan1D
- Single call process both signal and stores spectrum into single array
- In principle, you can load up thousands of cases and in a single call process all of them.





CONCLUSION

FFT PROCESSING ADDED TO BOXFILTERNPP.CPP

CONCLUSIONS

- This detailed study of the working of cuFFT in the 1D case has resulted in a fundamental understanding of the frequency spectrum produced by cuFFT
- Understanding of this frequency spectrum allowed that implementation of a high frequency filter that resulted in the expected top hat signal with ringing
- The demonstration of the batch FFT will enable many cases to executed with one call.

THANK YOU

Peter Brooker