GPU FFT Filtering of Seismic Data

Capstone Project

Challenges

- Writing code from scratch to read and write SEG-Y format seismic data
- Learning how to use cuFFT to do a 1-D R2C and then C2R transform, although it appeared to work, it was not until adding in cuFFT error checking to find that one of the cuFFT calls was silently failing
- Learning about Nyquist and Frequencies so I could properly write the kernel for filter the data samples based on a range of minimum and maximum frequency to keep
- All class example code generally centered around filtering pictures, so doing a project like this meant you had to figure things out for yourself

1. Intro

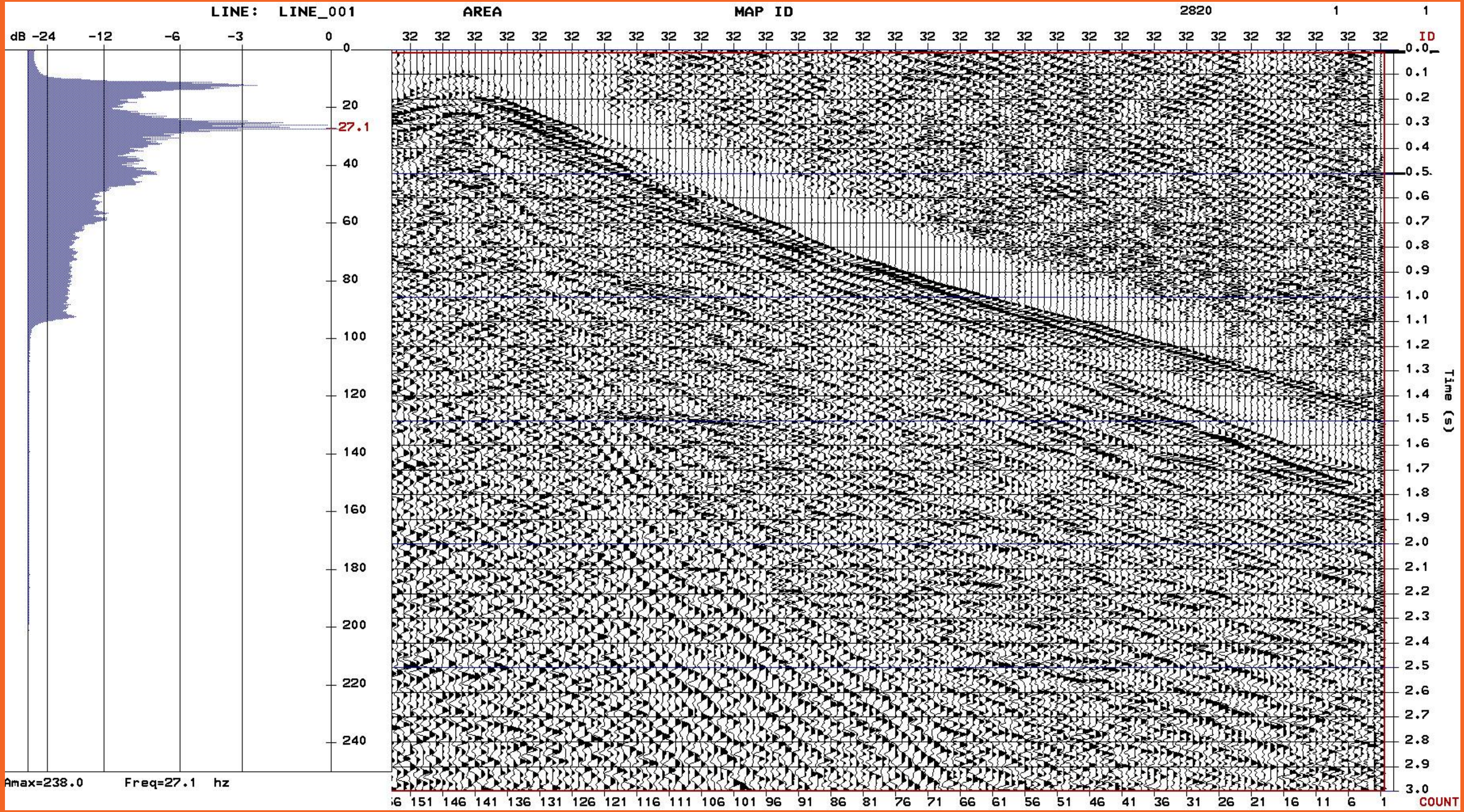
Seismic Data is the recording of sound/vibration by geophones when a source like dynamite or vibrators is set off. Filtering the data removing high and low frequencies can help to remove unwanted noise from the data.

Fast Fourier Transform

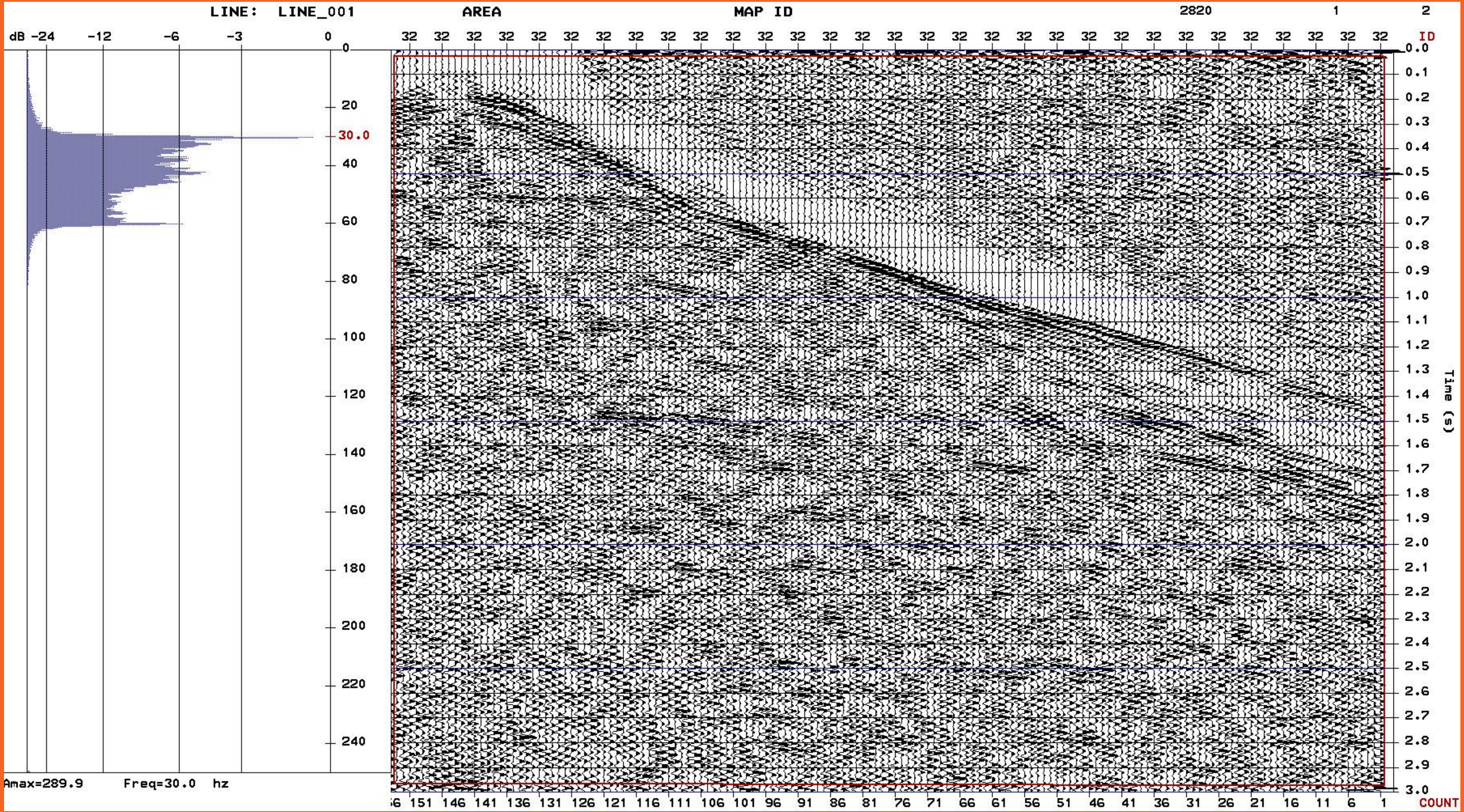
Converts from time domain to frequency domain, allowing you to work on specific frequencies in hertz

Nyquist

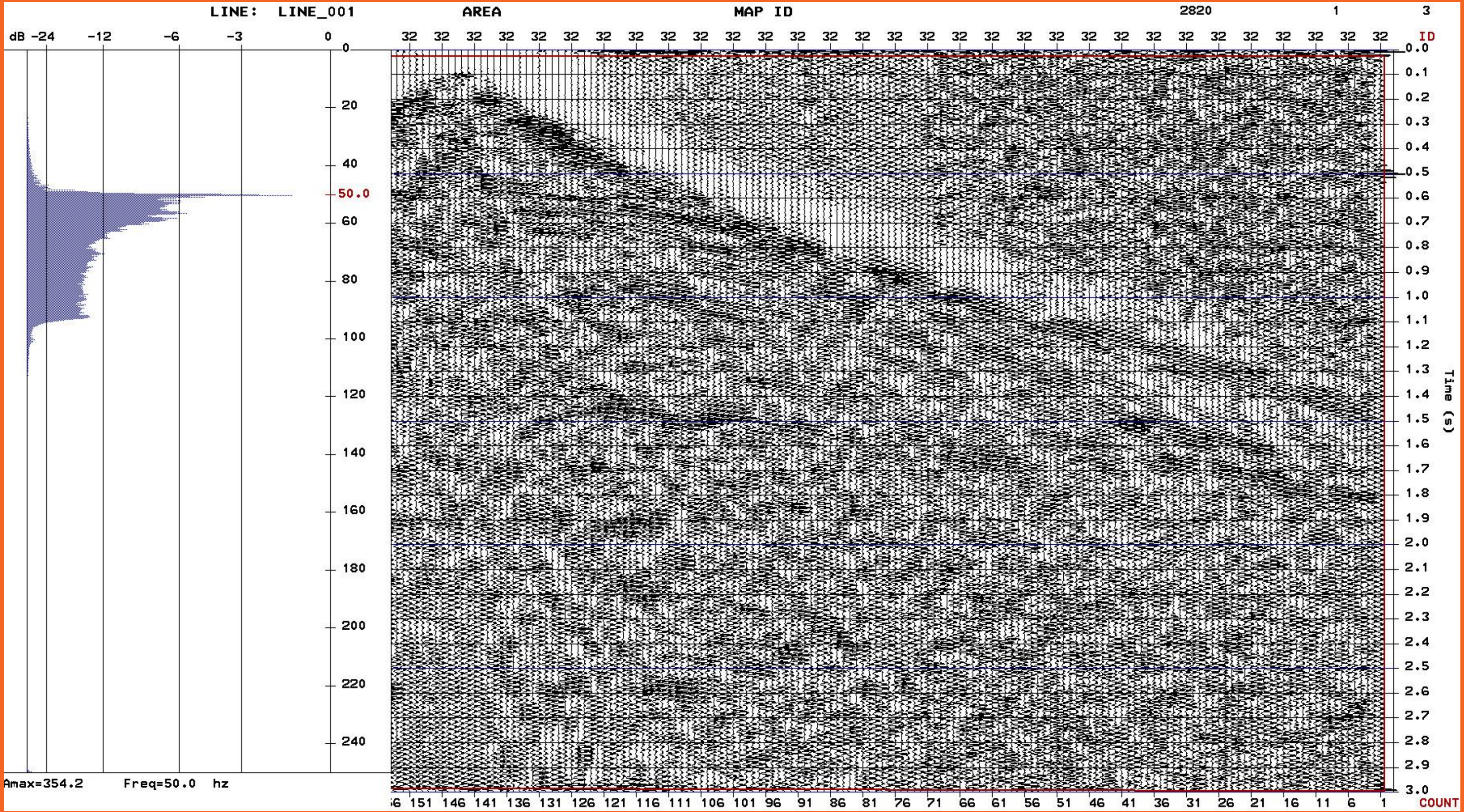
Nyquist frequency (*cycles per second*) is the frequency whose cycle-length (or period) is twice the interval between samples, thus *0.5 cycle/sample*. This information along with sample rate and number of samples is required to figure out what frequencies are contained within the converted FFT buffers the GPU memory.



Input Data before filtering, left hand graph shows frequency content through most of the range from 0 to 250 HZ



Output data after running with 30 to 60 min max parameters showing data concentrated in the range from 30 to 60 HZ



Output data after running with 50 to 100 min max parameters showing data concentrated in the range from 50 to 100 HZ

30 to 60 Run log:

```
CUDA    Device name: NVIDIA GeForce GTX 1650 Ti with Max-Q
Design
GPU Count: (1)
Core Count: (16)
ARCHITECTURE -> little endian (0)
Input File: ../data/Line_1_IEEE_Float_LE.sgy
Output File: output_30_60.sgy
Minimum Frequency: 30
Maximum Frequency: 60
Simultaneous Traces: 10000

Open for read: ../data/Line_1_IEEE_Float_LE.sgy
EBCDIC Header Read 3200 Bytes
Binary Header Read 400 Bytes
Ebcdic Header: EBCDIC
Revision < 2.0 Detected
Binary Header: IBM ORDER
Num Traces:     -31073 (FFFF869F)  bytes 3213-3214
Num Aux Tr:       0 (0000)  bytes 3215-3216
Sample Interval: 2000 (07D0)  bytes 3217-3218
Num Samples:    1501 (05DD)  bytes 3221-3222
Revision:  ( 0)    0 (0000)  bytes 3297-3300
Format:         11 (000B)  bytes 3225-3226

Calculated Number of Traces:        2820
Open for write: output_30_60.sgy

First/Last 8 data samples for QC:
Input 0: 4.49441051
Input 1: 6.59748077
Input 2: 1.05115414
Input 3: -5.72130013
Input 4: -5.60891533
Input 5: 1.45248413
Input 6: 7.15355396
Input 7: 4.57948112
Input 15009992: 0.00000000
Input 15009993: 0.00000000
Input 15009994: 0.00000000
Input 15009995: 0.00000000
Input 15009996: 0.00000000
Input 15009997: 0.00000000
Input 15009998: 0.00000000
Input 15009999: 0.00000000
#### gufftfilt_32f_i  (60040000)  17611680/17611680
Output 0: 20292.77539062
Output 1: 7613.83007812
Output 2: -3015.89257812
Output 3: -4689.10351562
Output 4: 3657.17187500
Output 5: 16573.44921875
Output 6: 25208.56835938
Output 7: 22326.87304688
Output 15009992: 0.00000000
Output 15009993: 0.00000000
Output 15009994: 0.00000000
Output 15009995: 0.00000000
Output 15009996: 0.00000000
Output 15009997: 0.00000000
Output 15009998: 0.00000000
Output 15009999: 0.00000000

Total Traces Processed: 2820
Total Samples Processed (float numbers): 4232820
Elapsed Time: 46590706 microseconds
```

50 to 100 Run log:

```
CUDA    Device name: NVIDIA GeForce GTX 1650 Ti with Max-Q
Design
GPU Count: (1)
Core Count: (16)
ARCHITECTURE -> little endian (0)
Input File: ../data/Line_1_IEEE_Float_LE.sgy
Output File: output_50_100.sgy
Minimum Frequency: 50
Maximum Frequency: 100
Simultaneous Traces: 10000

Open for read: ../data/Line_1_IEEE_Float_LE.sgy
EBCDIC Header Read 3200 Bytes
Binary Header Read 400 Bytes
Ebcdic Header: EBCDIC
Revision < 2.0 Detected
Binary Header: IBM ORDER
Num Traces:     -31073 (FFFF869F)  bytes 3213-3214
Num Aux Tr:       0 (0000)  bytes 3215-3216
Sample Interval: 2000 (07D0)  bytes 3217-3218
Num Samples:    1501 (05DD)  bytes 3221-3222
Revision:  ( 0)    0 (0000)  bytes 3297-3300
Format:          11 (000B)  bytes 3225-3226

Calculated Number of Traces:        2820
Open for write: output_50_100.sgy

First/Last 8 data samples for QC:
Input 0: 4.49441051
Input 1: 6.59748077
Input 2: 1.05115414
Input 3: -5.72130013
Input 4: -5.60891533
Input 5: 1.45248413
Input 6: 7.15355396
Input 7: 4.57948112
Input 15009992: 0.00000000
Input 15009993: 0.00000000
Input 15009994: 0.00000000
Input 15009995: 0.00000000
Input 15009996: 0.00000000
Input 15009997: 0.00000000
Input 15009998: 0.00000000
Input 15009999: 0.00000000
#### gufftfilt_32f_i  (60040000)  17611680/17611680
Output 0: 19214.82031250
Output 1: 27434.47851562
Output 2: 20398.21093750
Output 3: 5081.96093750
Output 4: -6893.46484375
Output 5: -10853.37500000
Output 6: -11840.56250000
Output 7: -15484.06640625
Output 15009992: 0.00000000
Output 15009993: 0.00000000
Output 15009994: 0.00000000
Output 15009995: 0.00000000
Output 15009996: 0.00000000
Output 15009997: 0.00000000
Output 15009998: 0.00000000
Output 15009999: 0.00000000

Total Traces Processed: 2820
Total Samples Processed (float numbers): 4232820
Elapsed Time: 47010962 microseconds
```

2. Results

Filtered Data

Output data was correctly filtered by user supplied minimum and maximum frequency

Repeatability

The data was run and compared multiple times to ensure there was luck involved and the transform and filter worked correctly

Robustness

The program was run on data sets over 500MB containing millions of data samples and functioned correctly. Unfortunately these data sets could not be uploaded to GItHub as they have size limit restrictions

3. Learning

Troubles I ran into and things I learned :

FFT Buffers

Learned why you need to divide by 2 and then add 1 for cuFFT buffers, Realized how your time samples are now half the amount but contain a real/imaginary part representing amplitude and phase.

NVtop

The importance of trying large data sets and using NVtop to troubleshoot. Upon a large run the program reported memory inconsistencies, NVtop showed a growing memory usage while running. I tracked it down to the cuFFT plan not being freed. A small run did not crash, so use larger data sets and profile your run with NVtop to ensure your run is sustainable. Free you buffers and plans!

4. More Learning

Even more things I learned :

Cuda Functions

Unless there is an actual fault, cuda functions will happily return quietly while doing nothing they were supposed to. You must write all error checking of functions from the very start, or you will chase problems needlessly for hours.

Parameters

Test many different parameters and data sets, they will let you find bugs you did not see and let you find performance improvements. Also test your code with parameters that do not change the data, this will ensure that you are able to reproduce your input correctly, before you start adding complex filtering.