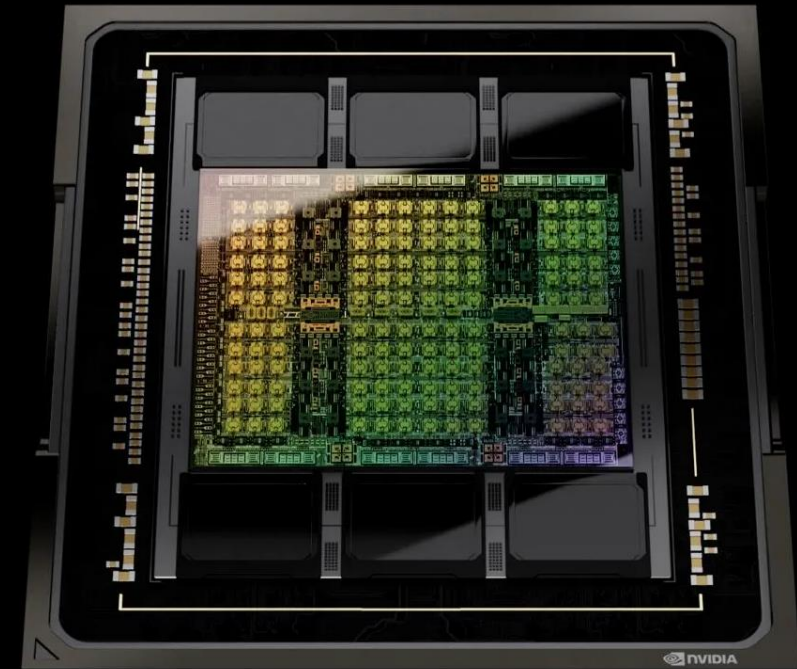


Using GPU for real-time SDR Signal processing

libGKR4GPU

Sylvain - F4GKR



Intro & Outline

- Author : Sylvain Azarian – F4GKR
 - Founder of « SDR-Technologies » , small French company around Paris
 - Former staff of ONERA (Radar Dept) and Director of SONDRA Lab in Paris-Saclay Univ.
 - Involved in Amateur Radio organizations (President of IARU R1)
- Outline of the talk
 - Motivation
 - DDC in SDR: why it does need “some” CPU cycles
 - Using GPU: does it bring anything ?
 - The “libgkr4gpu” : what is it like ?
 - Q&A

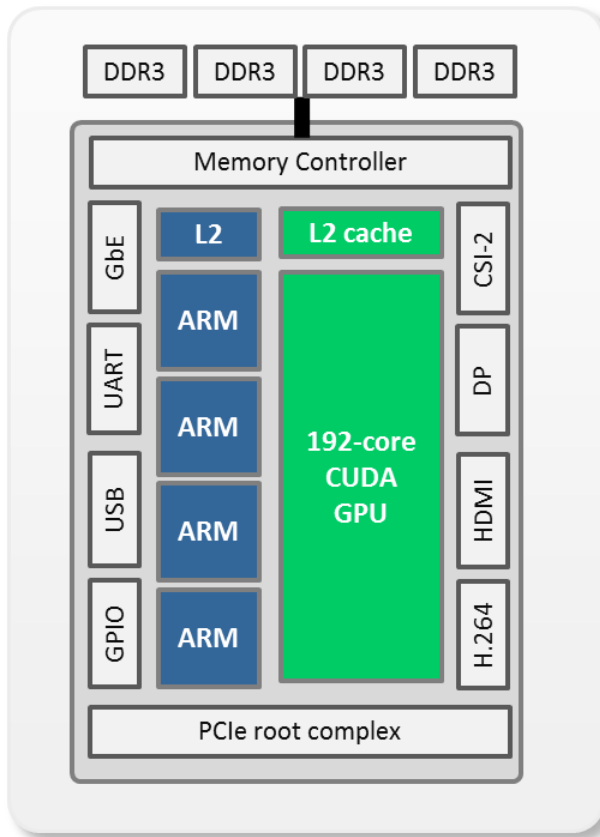
Background

- The story started while working in Radar & Signal Processing (at ONERA), when the Tegra K1 Soc was released
 - Radar processing, digital beamforming generate heavy processing needs and a « more compact » solution were required
- I was tasked to explore GPU-based solutions
- GPU for SDR is now the « core business » of the company funded in 2017



ONERA
THE FRENCH AEROSPACE LAB

What looked promising ?



326 GIGA FLOPS for 5 WATTS !!!!!!!!!!!!!!!

- 4 Core ARM Cortex-A15
- 192 CUDA cores
- Linux 😊

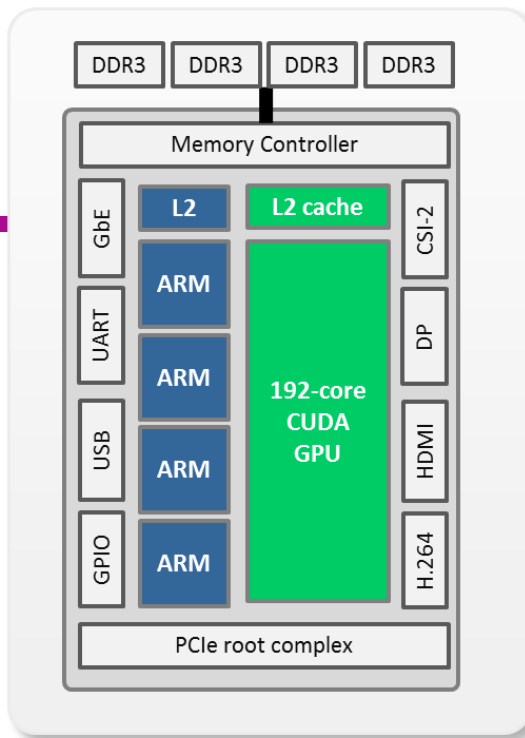
The 99€ question :

Can this bring **anything** to real-time continuous signal processing ?

The programming model

```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}

int main()
{
    ...
    // Kernel invocation with N threads
    VecAdd<<<1, N>>>(A, B, C);
    ...
}
```

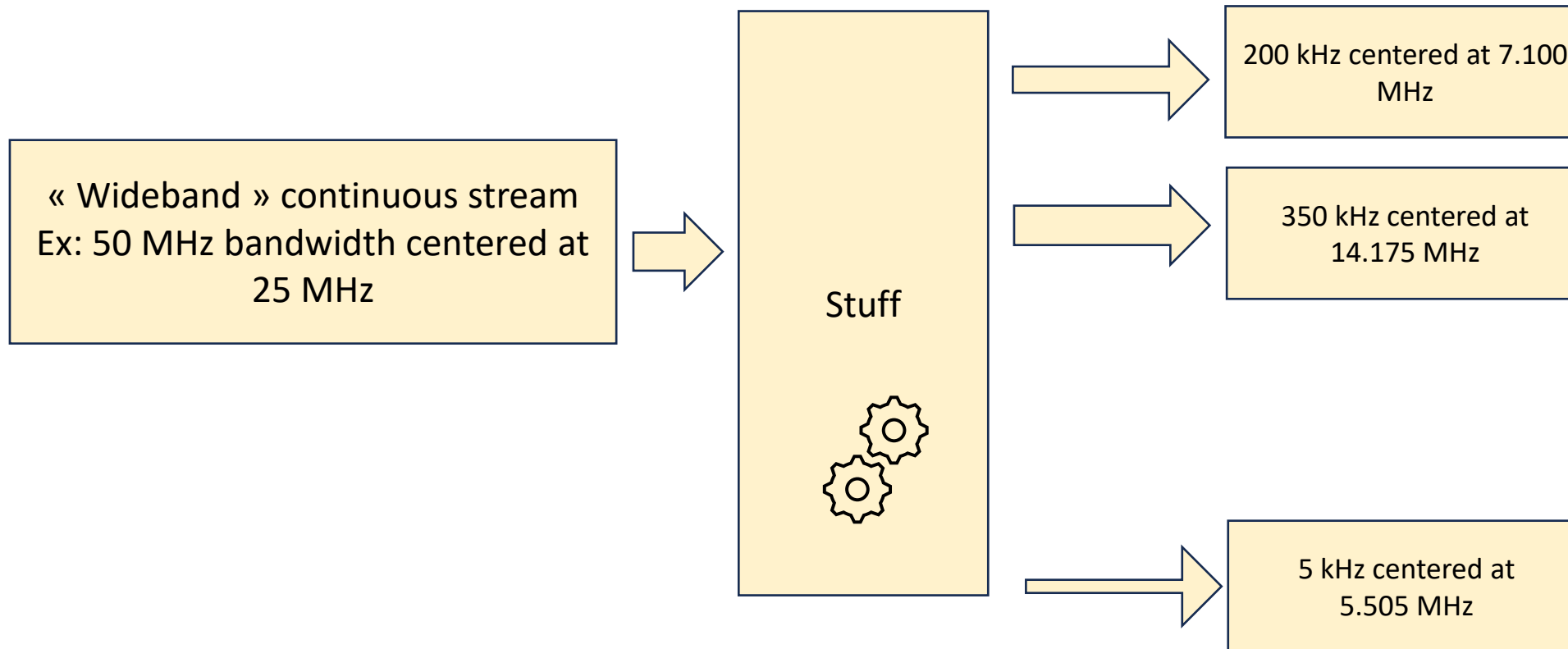


Examples of CPU consuming DSP blocks

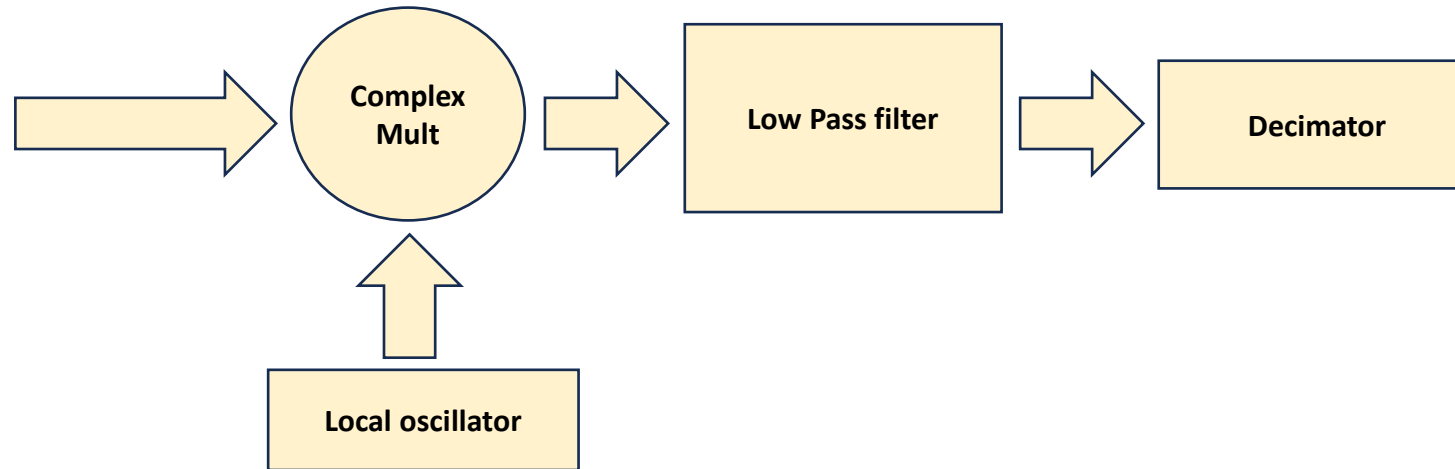
- Extracting narrow band signal from stream: DDC (Digital Down-Converter)
- Interpolation / Decimation
- Clock recovery
- Synchronization & pattern detection

What do we want to achieve

Have multiple sub bands from one single input, with different specifications (bandwidth, oversampling, ...)



How do we do this ? [for one channel]

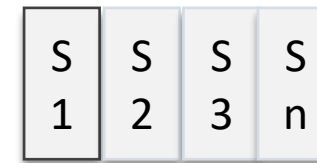


Low-Pass Filter : the convolution

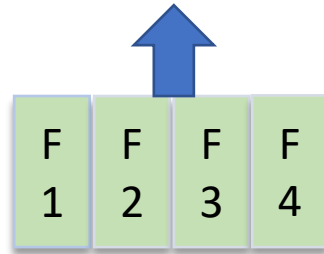
Input



$$S_1 = E_1 * F_1 + E_1 * F_2 + E_3 * F_3 + E_4 * F_4$$



Filter output



Filter coefficients – the « taps »

Where is the issue ?

Low-pass filter might need a lot of taps

For example, we want a SSB output IQ stream from a 50 MHz continuous stream

- Our signal is 3300 Hz wide, stop-band for example 6kHz
- We need at least 60 dB of attenuation for unwanted signals

$$B_T = \frac{6000 - 3300}{50 \text{ MHz}} = 0,000054$$

$$N_{taps} = \frac{60}{22 * 0,000054} = \frac{60}{0,001188} = 50\,500 \text{ taps}$$

« Harris approximation »

$$N_{taps} = \frac{Atten}{22 * B_T}$$

Atten is the desired attenuation in dB,

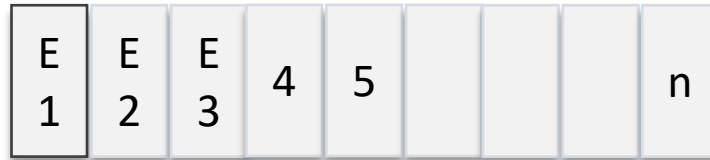
B_T is the normalized transition band $B_T = \frac{F_{stop} - F_{pass}}{F_s}$,

F_{stop} and F_{pass} are the stop band and pass band frequencies in Hz and

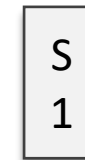
F_s is the sampling frequency in Hz.

So what ????

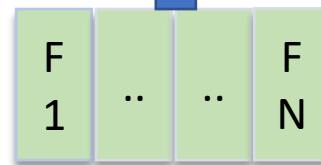
Input : 50 500 values at 50 MSPS



$$S1 = E_1 * F_1 + \dots + E_{50500} * F_{50500}$$



Filter output : 1 value

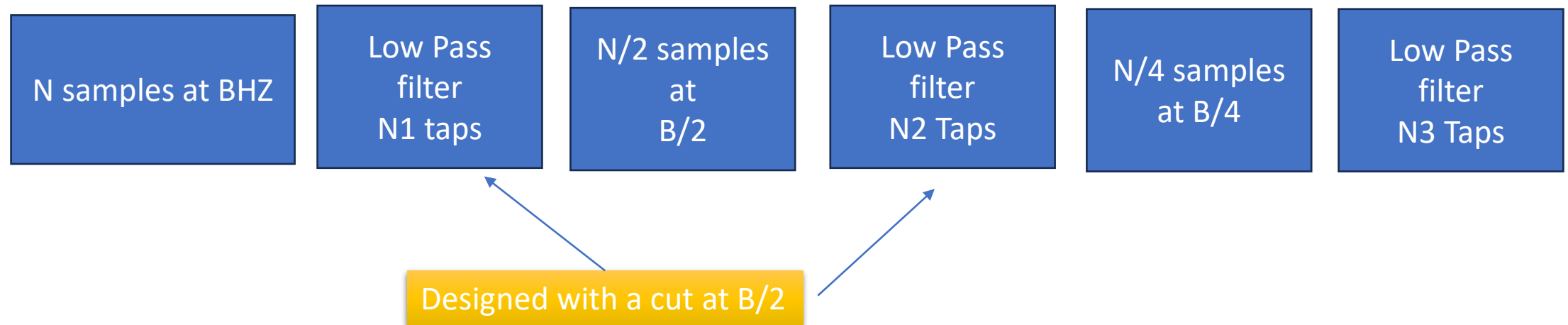


50500 coefficients

We must do this for **every sample**... that is 50 000 000 times per second

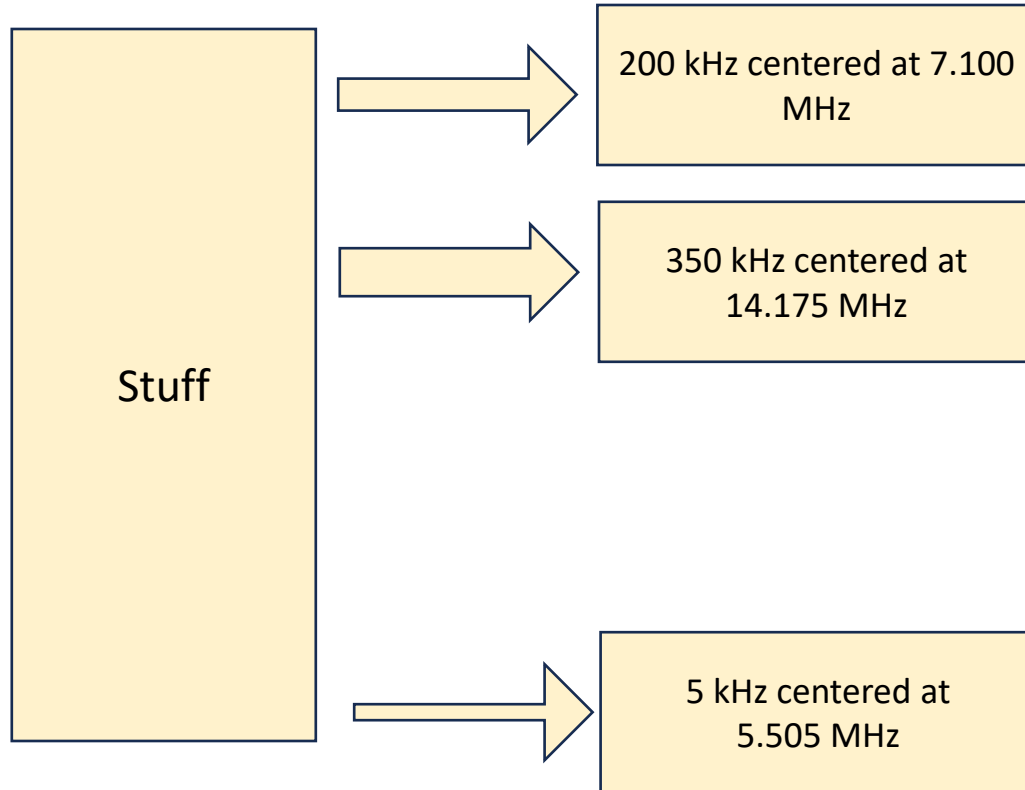
What are the solutions ?

- Divide by two, decimate, divide by two, decimate, divide by two, decimate....



- Half-band LPF = 50% of the coefficients are ... 0
- Each block deletes 50% of samples
- The number of taps is increased as the throughput is reduced : $N1 < N2 < N3 \dots$

But... ?



We can hardly reuse the “divide by 2 cascade”, because the center frequency of the different channel is different

Can GPU help ?

- NVIDIA Jetson Xavier NX

- GPU with 384 cores – 16 GB

- FFT Size : 524 288 (2^{19}) : 0.31 milli secs
- FFT Size : 8 388 608 (2^{23}) : 7.15 milli secs

- NVIDIA A100 :

- GPU with 6912 cores – 80 GB

- FFT size = 2^{23} : **0.17** milli secs (!)
- FFT size = 2^{30} : 23.3 milli secs

500 €

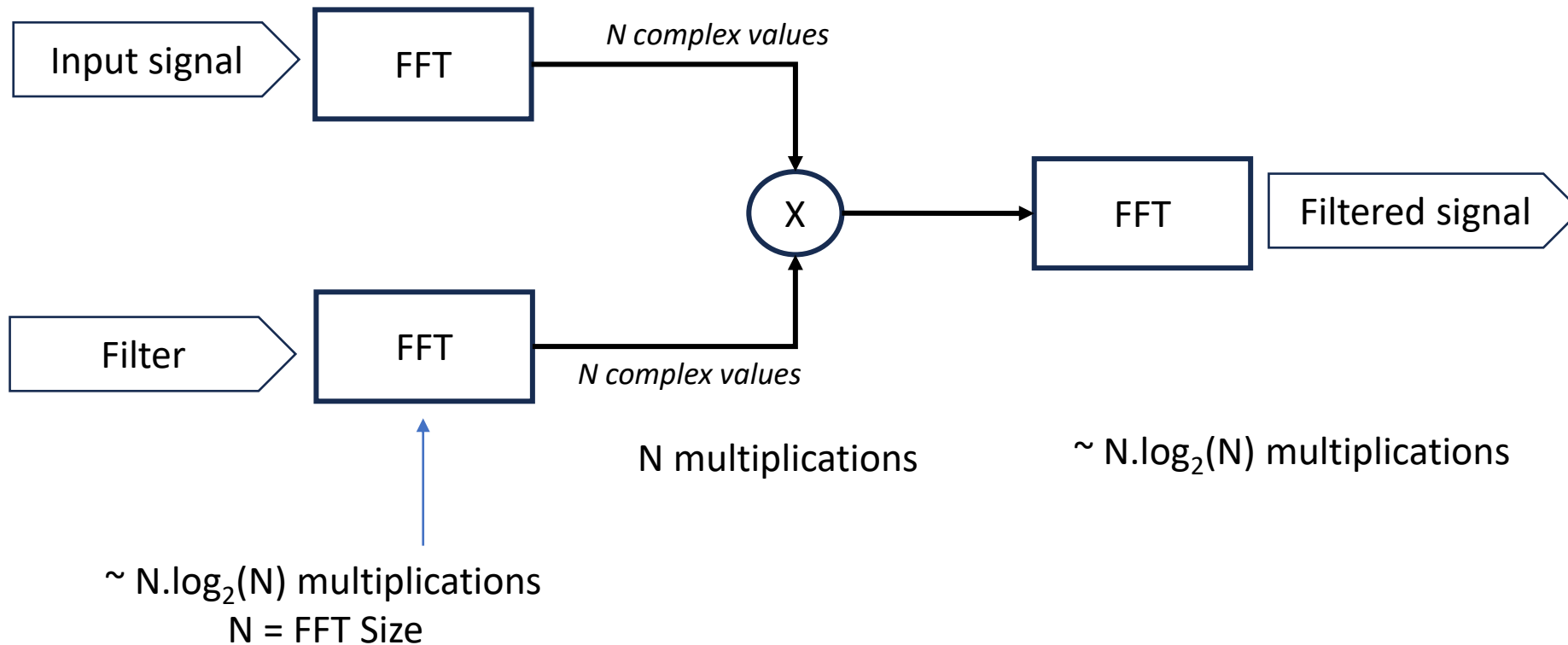


20 000 €

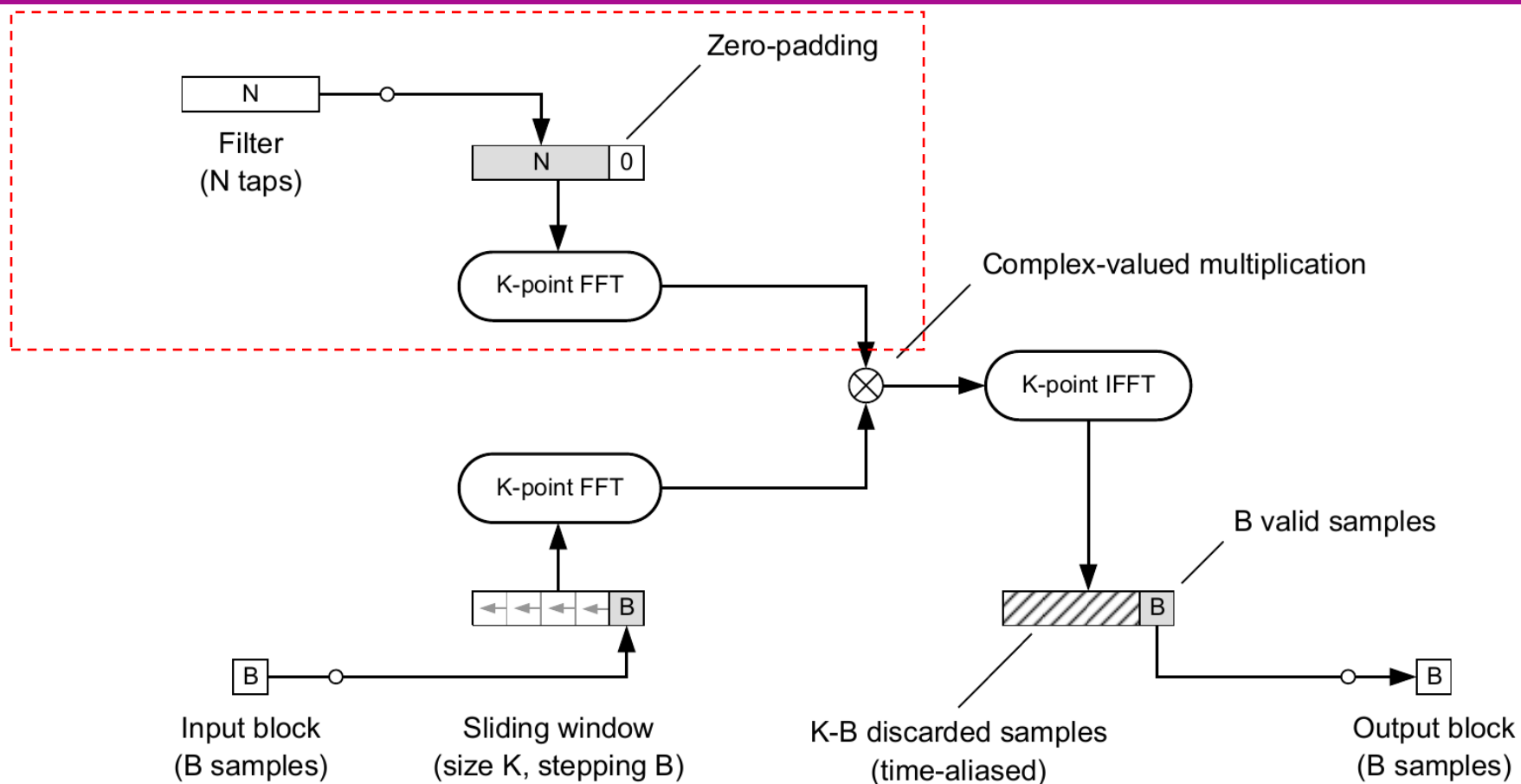


Convolution... and FFT

This works for 1 single block of N samples long

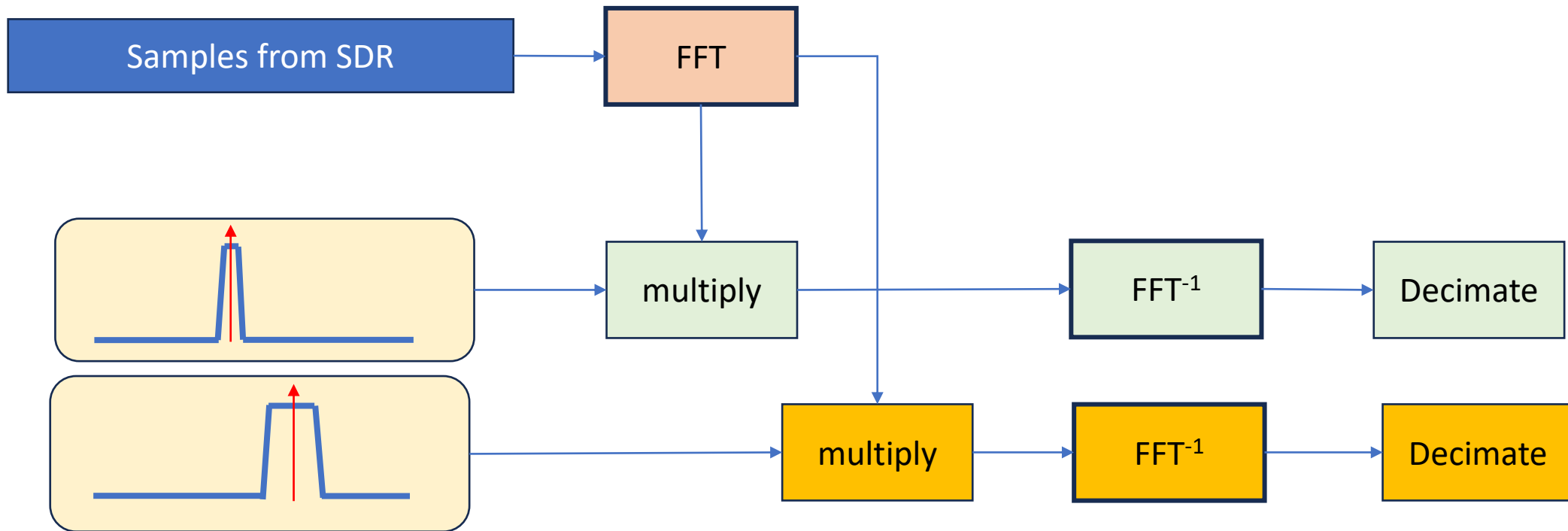


The Overlap-Save method



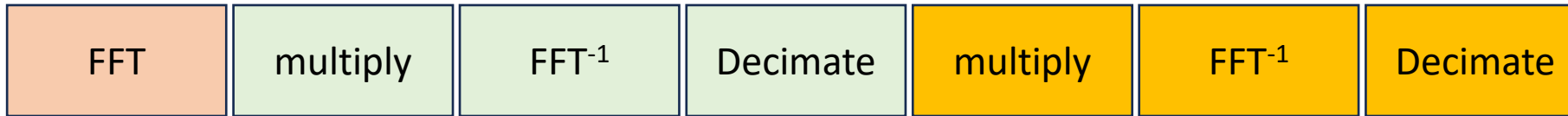
Source: <https://thewolfound.com/fast-convolution-fft-based-overlap-add-overlap-save-partitioned/>

Adding output channels



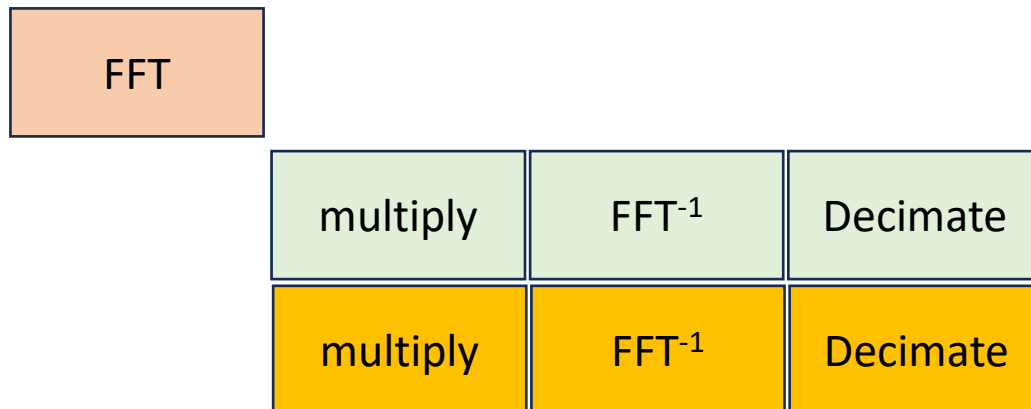
A nice feature from NVCC and NVIDIA devices

By default, kernels (CUDA code) are run sequentially...



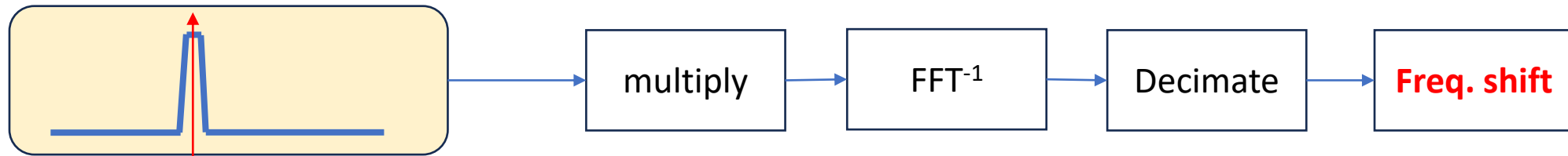
This enables the different GPU processing streams to run concurrently :

```
nvcc --default-stream per-thread
```



Small « issue » we need to fix

➡ We want our output band « centered »



➡ We need to frequency shift the signal...

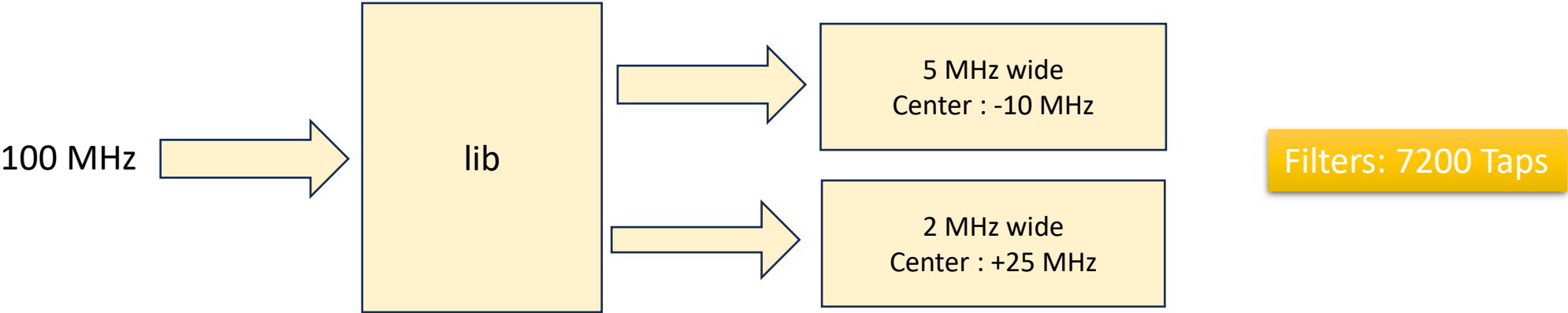
The easiest is to do this after the decimation step : we will use less multiplications
BUT we must compensate for the aliasing (look in the code 😊)

The « libGKR4GPU »

<https://github.com/f4gkr/libgkr4gpu/>


- Accepts « any » number of output channels (limit: GPU ram)
- Accepts « on the fly » addition, deletion of channels
- Thread safe
- No external dependency (except CUDA)
- Any channel can be retuned
- C “++” and CUDA, works **ONLY** with NVIDIA GPU, Desktop or Jetson family

A quick look at the performances



CPU	GPU	FFT Size	1 channel	2 channels
Intel® Core™ i7-9700K CPU @ 3.60GHz × 8	GeForce RTX2060	512*1024	608 Mega samples/sec	530 Mega samples/sec
Jetson Xavier NX	Jetson	256*1024	130 Mega samples/sec	70 Mega samples/sec
Jetson Xavier NX	Jetson	512*1024	156 Mega samples/sec	117 Mega samples/sec
Jetson Xavier NX	Jetson	1024*1024	143 Mega samples/sec	103 Mega samples/sec

NVIDIA Settings



Graphics Card Information

Graphics Processor:

NVIDIA GeForce RTX 2060

GPU UUID:

GPU-625fe369-d250-8c6a-f9d2-7f99931e268e

CUDA Cores:

1920

VBIOS Version:

90.06.3F.40.40

Total Memory:

6144 MB

Total Dedicated Memory:

6144 MB

Used Dedicated Memory:

803 MB (13%)

Memory Interface:

192-bit

GPU Utilization:

79 %

Video Engine Utilization:

0 %

Bus Type:

PCI Express x16 Gen3

Bus ID:

PCI:1:0:0

PCI Device ID:

0x1f08

PCI Vendor ID:

0x10de

IRQ:

148

PCIe Generation:

Gen3

Maximum PCIe Link Width:

x16

Maximum PCIe Link Speed:

8.0 GT/s

PCIe Bandwidth Utilization:

36 %

X Screens:

Screen 0

Display Devices:

Idek Iiyama PL3270Q (HDMI-0)

Help

Quit

azarian@azarian-MS-7B17: ~

0[| 1.3%] 4[| 100.0%]

1[| 2.0%] 5[| 0.0%]

2[| 3.3%] 6[| 0.0%]

3[| 1.3%] 7[| 0.7%]

Mem[| 3.65G/31.3G] Tasks: 166, 764 thr; 2 running

Swp[| 0K/2.00G] Load average: 1.49 0.96 0.70

Uptime: 03:06:43

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
17545	azarian	20	0	5147M	145M	109M	R	100.	0.5	2:03.69	./ddcexample
17546	azarian	20	0	5147M	145M	109M	S	0.0	0.5	0:00.00	./ddcexample
17550	azarian	20	0	5147M	145M	109M	S	0.0	0.5	0:00.10	./ddcexample
17551	azarian	20	0	5147M	145M	109M	S	0.0	0.5	0:00.00	./ddcexample

EnterDone EscClear

Filter: ddc

PROBLEMS 3

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

perfs : 545.315594 Millions samples / sec

perfs : 544.431402 Millions samples / sec

perfs : 543.571514 Millions samples / sec

perfs : 544.732991 Millions samples / sec

perfs : 544.732991 Millions samples / sec

perfs : 544.732991 Millions samples / sec

perfs : 544.732991 Millions samples / sec

perfs : 543.571514 Millions samples / sec

perfs : 542.992630 Millions samples / sec

perfs : 544.151633 Millions samples / sec

perfs : 544.732991 Millions samples / sec

perfs : 544.732991 Millions samples / sec

perfs : 543.571514 Millions samples / sec

perfs : 544.732991 Millions samples / sec

perfs : 544.732991 Millions samples / sec

perfs : 544.151633 Millions samples / sec

perfs : 543.850985 Millions samples / sec

FFT Size : 512x1024
Input: blocks of 256x1024 samples

Looking for speed

- Size of FFT and Filter length : depends on # of Cuda Cores
- Moving data from Host to GPU is expensive
- Gathering samples from SDR via USB through LibUSB is expensive
- The most important: the CPU is available for other tasks !

That's all folks

- Contact: f4gkr[at]iaru-r1.org