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
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

```c
// 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, ...)

« Wideband » continuous stream
Ex: 50 MHz bandwidth centered at 25 MHz

- 200 kHz centered at 7.100 MHz
- 350 kHz centered at 14.175 MHz
- 5 kHz centered at 5.505 MHz
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 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

\[ B_T = \frac{6000 - 3300}{50 \text{ MHz}} = 0.000054 \]

\[ N_{taps} = \frac{60}{22 \times 0.000054} = \frac{60}{0.001188} = 50 \, 500 \, \text{taps} \]
So what ????

Input: 50 500 values at 50 MSPS

\[ S_1 = E_1 F_1 + \ldots + E_{50500} F_{50500} \]

50500 coefficients

Filter output: 1 value

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....

N samples at BHZ
Low Pass filter N1 taps
N/2 samples at B/2
Low Pass filter N2 Taps
N/4 samples at B/4
Low Pass filter N3 Taps

Designed with a cut at B/2

• 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 ...
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
Convolution... and FFT

This works for 1 single block of N samples long

![Diagram of convolution and FFT process]

- Input signal
- FFT
- N complex values
- Filter
- FFT
- N complex values
- N multiplications
- ~ N.log₂(N) multiplications
- N = FFT Size
- ~ N.log₂(N) multiplications
- Filtered signal
- FFT
- ~ N.log₂(N) multiplications
The Overlap-Save method

Adding output channels

Samples from SDR → FFT → multiply → FFT⁻¹ → Decimate

Samples from SDR → multiply → FFT⁻¹ → Decimate
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:

```bash
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

<table>
<thead>
<tr>
<th>CPU</th>
<th>GPU</th>
<th>FFT Size</th>
<th>1 channel</th>
<th>2 channels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® Core™ i7-9700K CPU @ 3.60GHz x 8</td>
<td>GeForce RTX2060</td>
<td>512*1024</td>
<td>608 Mega samples/sec</td>
<td>530 Mega samples/sec</td>
</tr>
<tr>
<td>Jetson Xavier NX</td>
<td>Jetson</td>
<td>256*1024</td>
<td>130 Mega samples/sec</td>
<td>70 Mega samples/sec</td>
</tr>
<tr>
<td>Jetson Xavier NX</td>
<td>Jetson</td>
<td>512*1024</td>
<td>156 Mega samples/sec</td>
<td>117 Mega samples/sec</td>
</tr>
<tr>
<td>Jetson Xavier NX</td>
<td>Jetson</td>
<td>1024*1024</td>
<td>143 Mega samples/sec</td>
<td>103 Mega samples/sec</td>
</tr>
</tbody>
</table>
libGKR4GPU: A GPU optimized multichannel DDC

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