Square Kilometer Array South Africa

Internship technical report

GPU-accelerated Inverse Polyphase Filterbank

Author:
Benjamin Hugo

Department of Computer Science
University of Cape Town
bennahugo@aol.com

Advisor:
Ludwig Schwardt

February 6, 2014

The results obtained in this report were made possible through the use of the ICTS High Performance HEX cluster of the University of Cape Town (UCT) `hex.uct.ac.za`. The author wishes to thank Andrew Lewis of UCT ICTS for the technical support he provided during this project.

Glossary

DFT	Discrete Fourier Transform. 1, 3–5, 7
FFT	Fast Fourier Tranform is a fast algorithm for computing the DFT using the Cooley–Tukey algorithm amongst others. 7
FIR	Finite Impulse Response. 4, 5, 7
FPGA	Field Programmable Gate Array. 6
lo	Local Oscillator. 4
PFB	Polyphase Filter Bank. 4
PR	Perfect Reconstruction. 5, 6
SNR	Signal to noise ratio. 3, 8
VLBI	Very Long Baseline Interferometry. 4

Contents

List of Figures

List of Tables

1 Introduction

At its core Very Long Baseline Interferometry (VLBI) is an interferometry process where the output from several radio antennae are combined, to form an equivalent output to a telescope of the size equal to the distance between the furthest two antennae in the VLBI array. For a comprehensive overview of the technique the reader is referred to the detailed introduction by Middelberg et al. [2]. Several such extended arrays exist including the European VLBI network and the Very Long Baseline Array. The hope is to include the KAT-7 in future VLBI observations.

Traditionally the process required raw data to be dumped to a storage medium, say tape and physically shipped to a correlator where the data from several telescopes could be combined. The new trend in VLBI is to perform real-time correlation between antennae using high-speed internet connections and is known as 'eVLBI'. Both the Australian Long Baseline Array and the European VLBI network have performed successful eVLBI observations.

Ultimately the problem being investigated (at least in part by this report) can be boiled down to converting data sent over the SPEAD protocol (employed internally by the KAT-7 array) to the VDIF format. This conversion process includes a necessary first step, where the current sampling rate of 800 MHz [1] is reduced to 128 MHz through *Digital Downconversion*. The basic operation involves the following three steps:

1. Mixing. Where the signal is shifted down to *baseband* (frequency 0 of the spectrum), by mixing the signal with a tone produced by a Local Oscillator (lo). This is simply a generated sine wave at the lower end of the sub-spectrum to be extracted. This is simply done by an element-wise multiplication of the original signal with the local oscillator tone. In essence mixing is not a *linear* (refer to [3, ch. 5]) process. If f was a single channel in the frequency domain then mixing produces replicated channels at $f - f_{lo}$ and $f + f_{lo}$.
2. Filtering. In order to eliminate aliasing in the frequency domain due to mixing and frequencies above the new sampling rate we use a Finite Impulse Response (FIR) filter with the cutoff set at the rate $\frac{1}{2} f_{decimated}$ to comply with the Nyquest sampling theorem (see [3, ch. 3]).
3. Interpolation and Decimation. The reader is referred to `http://www.dspguru.com/dsp/faqs/multirate/basics` for an overview of the process.

There is, however, a further complication to deal with, before this downconversion process can begin. The KAT-7 beamformer produces a series of frequency spectra. If no filtering was applied to these spectra, the undo operation would only have involved applying the inverse Discrete Fourier Transform (DFT). However, the original voltage data went through a filter-bank operation, in a process known as the Polyphase Filter Bank (PFB). This method is also known as the Weighted Window Overlap method and is necessary for computing Short Time Discrete Fourier Transforms subsections of very long signals. The reader is referred to `https://casper.berkeley.edu/wiki/The_Polyphase_Filter_Bank_Technique` for a detailed description on the forward PFB.

2 Analysis and design

2.1 Validation of the forward and inverse processes

The forward PFB process can be thought of as a very basic analysis filter bank, whereas the inverse process can be thought of as a synthesis filter bank. The analysis filter bank uses a Hamming-windowed

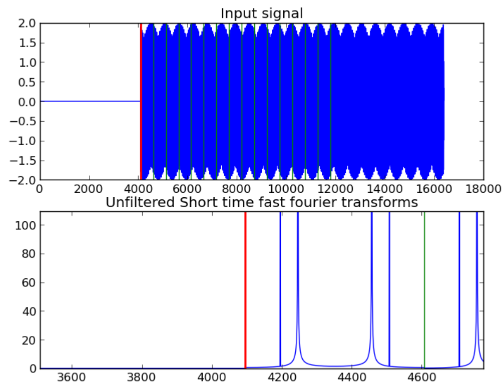
[1] According to the KAT-7 Data Interface document

FIR with $P = 8$ banks of $N = 1024$ elements, with no up or down-sampling stages [2]. Our implementation focusses on 8-bit sized samples produced by the KAT-7 beamformer [3], but could easily adapted to other sample-sizes, as well as prototype filter lengths.

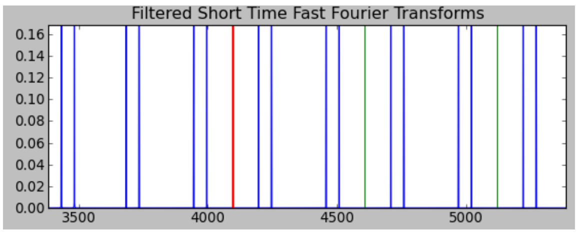
The key differences between the analysis and synthesis processes are:

1. If $H[n]$ is the prototype FIR filter and the analysis filter bank uses the subfilters $H_1[n], H_2[n] \ldots H_P[n]$ each of length N then the synthesis filter bank uses the subfilters $\bar{H}_1[N-n-1], \bar{H}_2[N-n-1] \ldots \bar{H}_P[N-n-1]$[4]
2. The commutator is flipped around (as shown on page 10 of Zhou's report) and therefore the sub filters should be processed in reverse order.

We've implemented both the basic analysis and synthesis filter banks, as well as the necessary tools to analyse the performance of this basic filter bank construction. As shown in figure 1 the analysis filter bank successfully limits leakage of any frequencies that lie between neighbouring bin centers of the DFT. This confirms that our basic analysis filter bank construction is correct.



(a) Unfiltered analysis filter bank



(b) Filtered analysis filter bank

Figure 1: Example of DFT leakage. It is clear that the unfiltered analysis filter bank suffers severely when containing frequencies even slightly off the bin centers of the DFT. Its filtered counterpart produces crisp binning of both centered and uncentered frequencies.

The synthesis filter bank construction was tested through cross correlation with the original input signal. A surefire way of determining if there is a correlation between the two intput and output of the sythesis filter bank, is to use Gaussian noise as input signal. Perfect Reconstruction (PR) requires that the signal only varies in the following two ways:

1. The output signal may be shifted by some Δ time steps.
2. The output signal may be scaled by a constant.

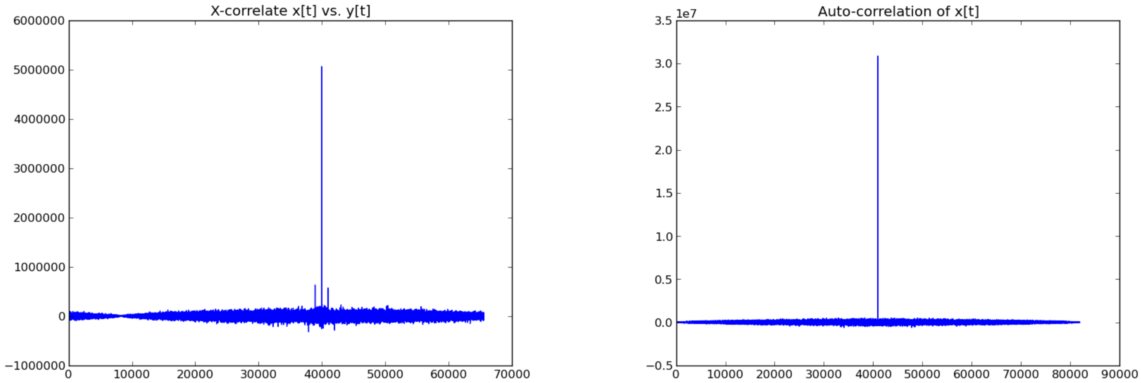
The signal shift may be found by the following method (where $x[t]$ is the input to the analysis filter and $y[t]$ is the output of the synthesis filter):

$$\Delta = \underset{t}{argmax}(x[t] \star y[t]) \tag{1}$$

[2]According to information provided by Jason Manley

[3]According to the KAT-7 Data Interface document

[4]Findings published in an in-house final technical report titled 'A review of polyphase filter banks and their application'. Daniel Zhou. Air Force Research Laboratory, Information Directorate, Rome Research Site, Rome, New York.



(a) Cross correlation between $x[t]$ and $y[t]$ (b) Auto correlation of $x[t]$

Figure 2: It is clear that even for Gaussian noise the synthesis filter bank achieves a limited reconstruction - the correlation between the reconstructed- and input signals is about 6 times weaker than the auto correlation of the input signal. The reader should also notice that it appears that $\Delta = -N(P-1)$. This observation was made for trials with input signals of different lengths, but we have not rigorously proven this and it should not be taken as a fact. Future expansion, testing and debugging should use the cross correlation to determine Δ.

According to Parishwad Vaidyanathan [4] this basic filter bank cannot achieve PR and have to be modified in order to do so. This involves modifying both the analysis and synthesis filter banks. In our situation this is not plausible, since the only the basic form of analysis filter bank is implemented on Field Programmable Gate Arrays (FPGA) due to their simple construction and *fast evaluation times* [4]. Vaidyanathan goes on to state, however, that the synthesis filter bank requires subfilters of much greater length to obtain reasonably good reconstruction and is therefore computationally prohibiting. That said, as the reader can see from figure 2, $x[t] \star y[t]$ shows that our basic synthesis filter bank achieves limited reconstruction.

2.2 Validation of the GPU version

We will for now only focus on the most basic construction to determine what throughput rates can be achieved, as well as the quality of the reconstruction. The full source code along with test and analysis suite should be included with this report. With this suite the C++ CUDA GPU code is validated against a serial python implementation. The reader should have GCC 4.6 installed, along with the NVIDIA NVCC compiler toolkit 5.0 redistributable. Although the code should work on any platform, we have only tested this under recent GNU/Linux kernels. Another requirement is that the user have a Compute Capability ≥ 2.0 installed on their system. To run the test suite the user should have Python 2.7 along with the SciLab suite of packages installed (including NumPy, SciPy and MatPlotLab).

In order to validate that the GPU version is in fact producing valid results we've tested it against the output of our Python synthesis filter bank. The Mean Squared Error is defined as:

$$MSE(input_1, input_2) := \frac{\sum_{k=0}^{L-1} (input_1[k] - input_2[k])^2}{L} \text{ where } ||input_1|| = ||input_2|| = L \quad (2)$$

By generating 0.976 GiB of Gaussian noise we determined these following characteristics for the two output files:

This subtle difference between the output files may be attributed to both floating-point rounding errors and the fact that we discarded the last non-redundant sample of FFT output. Different FFT implemen-

Metric	Python Synthesis Bank	GPU Synthesis Bank
$\bar{x}$	-0.000307	-0.000307
σ^2	2.969016	2.969015
Mean Squared Error	**0.000001**	

Table 2: Results from validating the GPU version against the serial python version of the synthesis filter bank

tations may behave differently under this situation and may produce slightly different results. It should be noted that here the GPU has processed the file in several batches as we will explain next.

2.3 Optimizations on the GPU-accelerated synthesis filter bank

The GPU version has been constructed with the requirements of real-time operation in mind. Although it has not been integrated into a SPEAD receiver, the device simply has to be initialized once. The GPU will allocate all the memory it requires during this initialization process. After this initialization step the user can send batches of any size off to the GPU for processing on a rolling bases, as long as one batch is fully processed before the next batch is started. The GPU inverts all the DFTs in a combined operation and keeps track of the last $P \times N$ N-sized inverse DFT, by keeping a persistent buffer between batches. The only hard constraint placed on this batching process is that the size of the batch is an integral multiple of N (the length of each analysis spectrum as mentioned earlier). At this point we wish to make the reader aware of subtlety in the output data received from the beamformers. Although a real-valued inverse DFT will require $\frac{N}{2} + 1$ non-redundant samples to perform a computation the last frequency bin is discarded before we receive it. The author has assumed $X_{filtered}[f = \frac{N}{2} + 1] = 0$ in his construction of the analysis and synthesis filterbanks.

Care has been taken in optimizing the GPU code according to the architectural requirements of GPUs:

1. Memory copies to and from the device is done in batches to improve latency
2. Memory accesses have been coalesced, including necessary padding to align memory to the 128 byte memory boundaries of Compute Capability 2.0. This constant can be adjusted for later architectures if necessary. At this point we should warn the reader that N must be a multiple of 128 for this coalescing scheme to function as intended.
3. Memory transfers to and from the GPU have been optimized by pinning host memory to primary memory so that they do not get paged out to disk by the memory controller of the host operating system.
4. DFTs are computed in batches using the NVIDIA cufft libraries using the Fast Fourier Transform.
5. CUDA streams are used to run kernels and memory copies asynchronously whenever possible. This adds an additional level of parallelism to the solution.
6. We investigated using the cached texture memory to store the prototype FIR filter coefficients. This did not provide greater throughputs compared to coalesced global memory accesses. We note for future reference that it is probably not worthwhile copying these filter taps to constant device memory as this memory only provides better throughputs when multiple threads (usually either a half-warp or warp of 32 threads) access the same element simultaneously. This is not true in our situation and using constant memory will result in serialization of the entire group of threads!
7. In compliance with the limitations of compute capability 2.0[5] devices we split the number of blocks executed in each grid between the Dim_x and Dim_y dimensions of the grid.

[5]Compute Capability 2.0 accepts a maximum of 65536 blocks per grid dimension

8. As it stands the computations of the basic synthesis filter bank did not require the use of shared memory. If kernels are added at a latter stage care should be taken to avoid bank-conflicts between warps of threads. The reader is referred to GPU Gems 3 [1, ch. 3] for a short introduction to this architectural constraint.

The user should manually tweak the memory allocation to fulfill their needs. All these constants are defined in the header file `inv_pfb.h`. It should be pointed out that the card should be completely filled with data in order to achieve good occupancy and offset relatively slow memory transfers through the PCI express bus.

3 Results and discussion

3.1 Quality of the reconstruction

In order to asses the quality of reconstruction we used Gaussian noise as input to the analysis filter bank and measured the signal to noise ratio SNR. Our results confirm our initial fears about the quality of reconstruction: the basic filter bank construction works well for analysis but does not achieve the desired results in the synthesis. Although there is a reasonable correlation, which is confirmed by visual inspection of the (time-shifted) output of the synthesis filter bank.

The SNR is calculated in the following manor ($x[t]$ is the input Gaussian noise and $y[t]$ is the output of the synthesis filter bank). Note that the initial padding, as outputted by the analysis and synthesis filter banks, are discarded and that an equal number of samples at the end of $y[t]$ are ignored. After this clamping operation $x[t]$ and $y[t]$ have equal length.

1. Obtain Δ through $x[t] \star y[t]$. It is assumed that $x[t], y[t] \in \mathbb{R}^n$.
2. The filter bank introduces a scaling factor to the output. This is canceled by scaling $y[t]$ by $c = \frac{\sigma^2_{x[t]}}{\sigma^2_{y[t]}}$. Additionally the shift introduced by analysis and synthesis is canceled: $y'[t] = c \times y[t - \Delta]$.
3. Compute the noise: $n[t] := y'[t] - x[t]$
4. Calculate $SNR(x[t], n[t])$ by the equation given below. The input signal $x[t]$ must have $\bar{x} \approx 0$. Standard Gaussian noise meets this requirement.

$$SNR(signal[t], noise[t])_{dB} := 10 \log_{10} \frac{\sigma^2_{signal[t]}}{\sigma^2_{noise[t]}} \tag{3}$$

We ensured that the correlation and shifting is functioning properly by passing in a tone that is only non-zero over a subinterval. The results are shown in figure 3.

On 24576 samples the SNR is equal to 0.063 dB, or equivalently a standard deviation ratio of 1.014. This is a poor reconstruction. Upon further investigation we noticed that $\sigma^2_{signal[t]} = 27.514$ and $\sigma^2_{signal[t]} = 27.117$. This shows that there is a significant amount of noise in the reconstruction. Refer to figure 4 for a visual inspection of this result.

3.2 GPU Computational performance

We have recorded performance measurements on a Tesla M2090 and a single core (Python implementation) of a AMD Opteron 6274 clocked at 2200 MHz with 128 GiB primary memory, clocked at 1333 MHz.

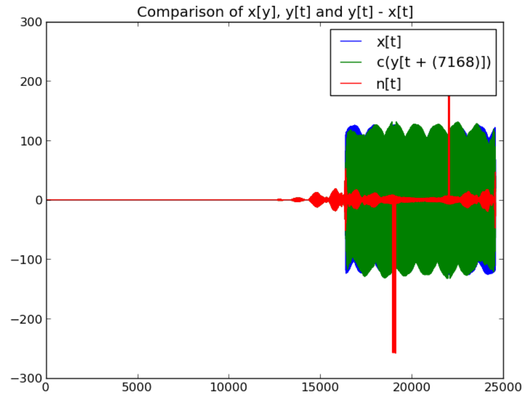
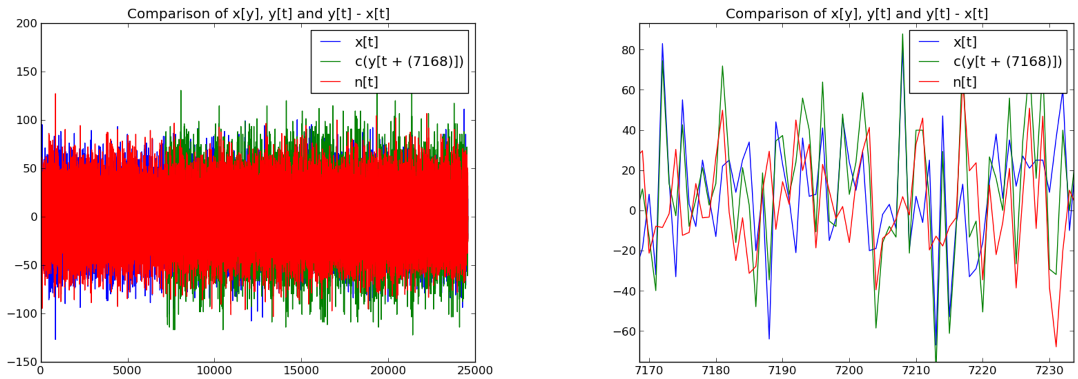


Figure 3: The output tone is shifted to the correct position



(a) Overview

(b) Closeup inspection of the input and output

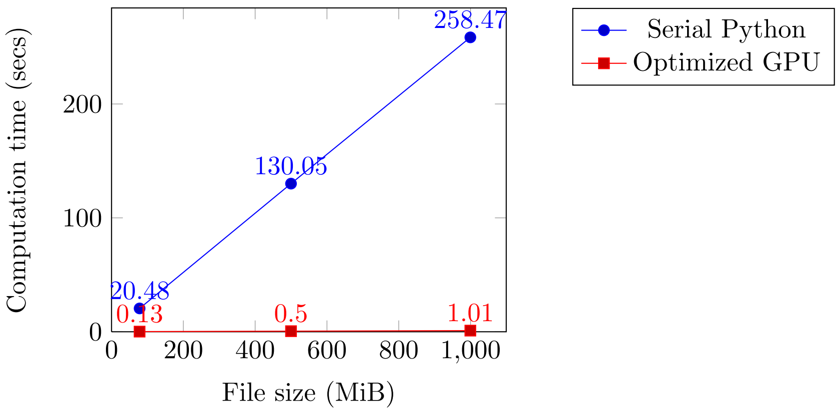
Figure 4: As is clearly visible there is a significant amount of noise in the synthesized signal. Although there is clearly a a limited correlation ($x[t]$ and $y[t]$ lines up almost perfectly on some of the peaks), at other positions there is almost no correlation. This explains why the signal to noise ratio is near 0 dB, and is indeed a very disheartening result.

These timings include memory transfer costs between the host and the GPU. It would be considered an unfair comparison if they are neglected.

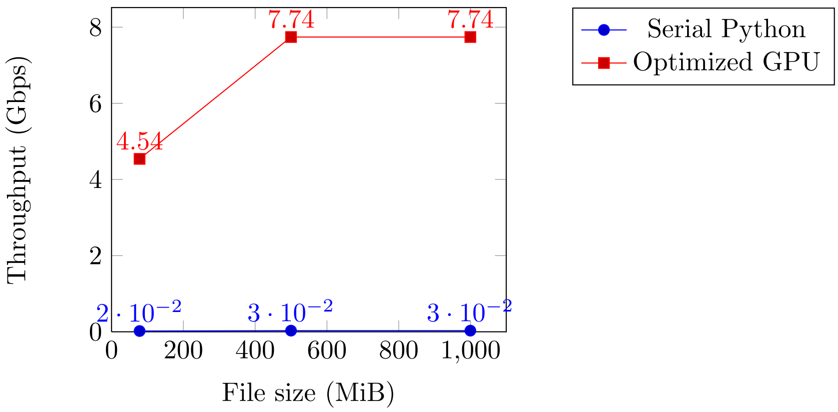
Although this speedup seems remarkably good the author wants to caution the reader that, ideally, the optimized GPU version should be compared to an parallel optimized CPU version. Not only can we easily parallelize this algorithm using OpenMP, but there is also opportunity here to exploit the full parallel capabilities of modern processors, by making use of the special vectorized instruction sets like Intel/AMD Streaming SIMD Extensions or Intel AVX. The latter can perform up to 8 32-bit floating-point operations per core simultaneously, by loading 8 elements into a special 256-bit register. This approach, coupled with higher CPU clock speeds may reduce this speedup significantly if executed on an 8 or 16 core server CPU, and by extension the latest Intel Xeon Phi co-processors.

4 Future works

There is a clear need for an investigation into extending the lengths of the subfilters in order to obtain better reconstructions. The filterbank operation has complexity of $O(n)$. Therefore it is reasonable to



(a) Computation time



(b) Throughput

Figure 5: These results indicates that the GPU significantly outperforms its serial counterpart. Additionally, when the GPU version is executed on a much larger file of almost 4 GiB the throughput increases to 8.20 Gbps - more than a 273x speedup! These rates well exceeds the requirements for real-time synthesis. It does, however, show that the GPU memory should be saturated with data in order to offset the memory transfer between the host and device.

assume that if the subfilters are expanded by a constant factor the computation time will also increase in a linear fashion. The author notes that this will require a slight adjustment in both the Python and GPU implementations where the number of FFT bins and subfilter lengths are not assumed to be equal, as per the basic construction.

References

[1] Mark Harris, Shubhabrata Sengupta, and John D Owens. Parallel prefix sum (scan) with cuda. *GPU gems*, 3(39):851–876, 2007.

[2] Enno Middelberg and Uwe Bach. High resolution radio astronomy using very long baseline interferometry. *Reports on Progress in Physics*, 71(6):066901, 2008.

[3] Steven W Smith et al. The scientist and engineer's guide to digital signal processing. 1997.

[4] Parishwad P Vaidyanathan. Multirate digital filters, filter banks, polyphase networks, and applications: A tutorial. *Proceedings of the IEEE*, 78(1):56–93, 1990.