Astronomy and Computing 16 (2016) 1–16

Contents lists available at ScienceDirect

Astronomy and Computing

journal homepage: www.elsevier.com/locate/ascom

Full length article

# A polyphase filter for many-core architectures

K. Adámek [a], J. Novotný [a], W. Armour [b,*]

[a] Institute of Physics, Silesian University in Opava, Faculty of Philosophy and Science, Bezručovo nám. 13, 746 01 Opava, Czech Republic
[b] Oxford e-Research Centre, University of Oxford, 7 Keble Road, Oxford OX1 3QG, United Kingdom

ARTICLE INFO

*Article history:*
Received 4 January 2016
Accepted 22 March 2016
Available online 30 March 2016

*Keywords:*
Graphics processors
Parallel architectures
Parallel programming languages
Parallel computing models
Parallel algorithms

ABSTRACT

In this article we discuss our implementation of a polyphase filter for real-time data processing in radio astronomy. The polyphase filter is a standard tool in digital signal processing and as such a well established algorithm. We describe in detail our implementation of the polyphase filter algorithm and its behaviour on three generations of NVIDIA GPU cards (Fermi, Kepler, Maxwell), on the Intel Xeon CPU and Xeon Phi (Knights Corner) platforms. All of our implementations aim to exploit the potential for data reuse that the algorithm offers. Our GPU implementations explore two different methods for achieving this, the first makes use of L1/Texture cache, the second uses shared memory. We discuss the usability of each of our implementations along with their behaviours. We measure performance in execution time, which is a critical factor for real-time systems, we also present results in terms of bandwidth (GB/s), compute (GFLOP/s/s) and type conversions (GTc/s). We include a presentation of our results in terms of the sample rate which can be processed in real-time by a chosen platform, which more intuitively describes the expected performance in a signal processing setting. Our findings show that, for the GPUs considered, the performance of our polyphase filter when using lower precision input data is limited by type conversions rather than device bandwidth. We compare these results to an implementation on the Xeon Phi. We show that our Xeon Phi implementation has a performance that is 1.5× to 1.92× greater than our CPU implementation, however is not insufficient to compete with the performance of GPUs. We conclude with a comparison of our best performing code to two other implementations of the polyphase filter, showing that our implementation is faster in nearly all cases. This work forms part of the AstroAccelerate project, a many-core accelerated real-time data processing library for digital signal processing of time-domain radio astronomy data.



## 1. Introduction

The technique of time-domain filtering is a rich and far reaching area in the field of signal processing. One of the cornerstones of time-domain data processing is the use of linear filters. Linear filtering of time-domain signals is a technique employed in many different scientific and industrial settings, from everyday tasks such as audio and video processing to the filtering of radio signals in the field of radio astronomy, it is this latter use of such filters that motivates our work.

This article focuses on the implementation of a polyphase filter on many-core technologies.[1] We use the problem posed by real-time signal processing of time-domain radio astronomy data as our application domain, however this work is in no way limited to this field alone. Typical signal processing pipelines in radio astronomy, Sclocco et al. (2014) and Chennamangalam et al. (2015) use several processing steps to extract a meaningful signal from input data. This is significant when employing many-core accelerators to achieve real-time processing because (in many cases) it allows data to reside on the accelerator card, circumventing the need for multiple host to device data transfers via the relatively slow PCIe bus. As such our codes can be used in two different ways. The first is as a stand-alone implementation of the polyphase filter, the second is a module that can be incorporated into an existing signal processing pipeline. Whilst our focus has been to produce implementations that run in real-time, the codes can also be used to process archived data.

Processing time-domain radio astronomy data in real-time enables events, such as Fast Radio Bursts, to be detected and observed as they occur. This allows scientists to create data rich observations by carrying out follow-up measurements whilst

* Corresponding author.
*E-mail addresses:* karel.adamek@fpf.slu.cz (K. Adámek), jan.novotny@fpf.slu.cz (J. Novotný), wes.armour@oerc.ox.ac.uk (W. Armour).

[1] https://github.com/wesarmour/astro-accelerate/tree/master/lib/AstroAccelerate/PPF.

http://dx.doi.org/10.1016/j.ascom.2016.03.003
2213-1337/

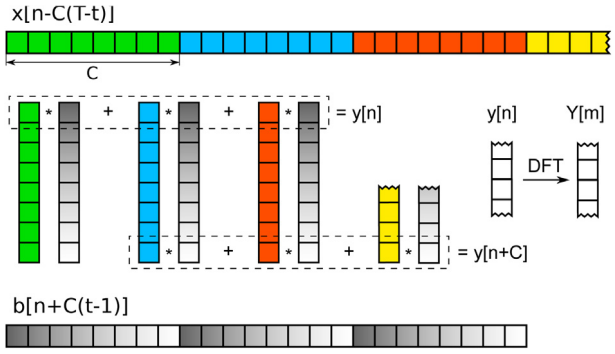


**Fig. 1.** Depicts the structure of an input data stream, the FIR filter operation and DFT. The input data stream (top), is divided into spectra each containing $C$ samples (spectra are differentiated by colour). Each sample (a single square) $x[n] \in \mathbb{C}$ belongs to a specific channel. In this example the spectra have 8 channels. The response function $b$ of the FIR filter must be of size $CT$ (bottom shaded by a gradient). The operation of the FIR filter is shown (middle). The FIR filter takes $T$ number of raw spectra (coloured groups of squares), in this case $T = 3$, and produces one filtered spectra $y[n]$. The next filtered spectra $y[n + C]$ reuses $T - 1$ raw spectra from the previous filtered spectra. The DFT acts on filtered spectra $y[n]$ to produce frequency spectra $Y[n]$ (middle, right). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

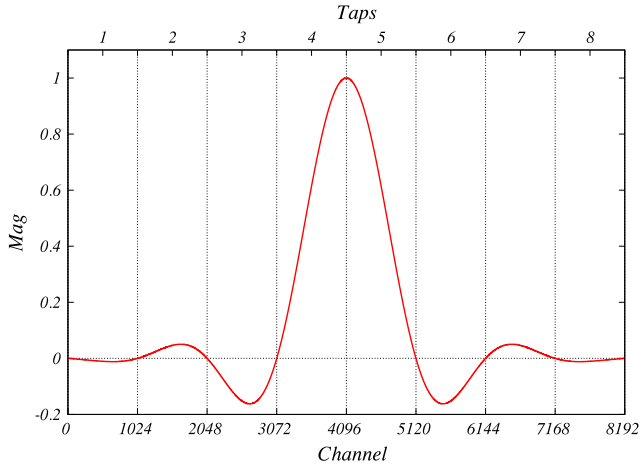


**Fig. 2.** Values of the coefficients of the response function $b[n]$ used in examples with 1024 channels per spectra. Coefficients are generated by a sinc($x$) function and multiplied by a Hanning window. On the $x$-axis we have the position within coefficients $0 \leq n < CT$, where $CT$ is the total number of coefficients. The division of the coefficients into taps (in this case $T = 8$) is depicted by vertical dashed lines.

an event is occurring. This is a vital part of our effort to understand very rare events (Karastergiou et al., 2015). However processing such vast data streams produced by modern radio telescopes can be an extremely demanding computational task. Data undergoes many different processes and transformations, such as de-dispersion (Armour et al., 2011; Clarke et al., 2014), before a signal from a distant celestial object can be discerned. This is why it is vitally important to ensure that all processes in a data processing pipeline operate as quickly and efficiently as possible, hence the use of many core acceleration Serylak et al. (2012), Magro et al. (2013) and Sclocco et al. (2014).

In this article we present various implementations of the polyphase filter (PPF). We discuss their application and limitations. We compare the performance of these on several different hardware architectures, three generations of NVIDIA GPUs (scientific and gaming), on CPUs and also the Intel Xeon Phi. We compare our results to two previous works on many-core platforms, the polyphase filter created for the VEGAS spectrometer (Chennamangalam et al., 2014), and the polyphase filter for the LOFAR radio telescope (van der Veldt et al., 2012).

Our work is structured as such: In Section 2 we discuss the polyphase filter and its features, in Section 3 we describe how we have implemented the polyphase filter on our chosen platforms, in Section 4 we discuss the behaviour of GPU implementations in detail and present our Xeon Phi implementation. Section 5 deals with performance comparison with other published work and sample rates per second are presented, we also briefly summarise our experience with different GPU generations. Lastly Section 6 summarises our work.

## 2. Polyphase filter bank

Before we describe the polyphase filter (PPF) algorithm, we shall address the structure of data on which we apply the polyphase filter. We assume input data to be a stream of complex samples $x[n] \in \mathbb{C}$ in the time domain, these are divided into $S$ groups, each containing $C$ samples, we call these groups *spectra*. We refer to the position of samples within a spectra as *channels*. The structure of the input data together with FIR filter is depicted in Fig. 1.

The polyphase filter consists of two steps. The first step is to apply a linear filter, which combines $T$ previous time domain spectra, we refer to them as *raw* spectra, into one *filtered* spectra. In the case of the polyphase filter the linear filter combines samples within a single channel of raw spectra and it is described by a finite impulse response (FIR) filter. The second step is to apply a discrete Fourier transformation (DFT) applied on a single filtered spectra which produces a single *frequency* spectra. These two steps are outlined as follows:

$$\text{raw spectra} \xrightarrow{\text{FIR}} \text{filtered spectra},$$

$$\text{filtered spectra} \xrightarrow{\text{DFT}} \text{frequency spectra}.$$

The FIR filter is mathematically given (Lyons, 2011) by

$$y[n] = \sum_{t=1}^{T} x[n - C(T - t)]b[CT - C(t - 1)], \tag{1}$$

where $0 \leq n < C$, square brackets [ ] indicate that a physical quantity is discrete (sampled), $x[n]$ represents samples from the input data belonging to the raw spectra and the quantity $y[n]$ represents samples in the filtered spectra. Quantities $y[n]$ and $x[n]$ are assumed to be complex. The FIR filter is a convolution of samples $x[n]$ within a single channel with coefficients of a response function $b$. The number of past samples which the FIR filter operates on is called *taps*, denoted by $T$. The choice of a response function $b$ depends on the desired features of the polyphase filter. The data access pattern for the FIR filter is depicted in Fig. 1.

We have used a sinc($x$) function to generate the coefficients used in this article, however these are easily replaceable in our code. The sinc($x$) function in the time-domain transforms into a pair of a rectangular windows in the frequency domain. To obtain more accurate results we have multiplied the sinc($x$) function by a Hanning window (Lyons, 2011). The resulting coefficients can be seen in Fig. 2.

The discrete Fourier transformation (DFT), which forms the second step of the polyphase filter, is given (Lyons, 2011) by

$$Y[m] = \sum_{n=0}^{C-1} y[n] \exp \frac{-i2\pi nm}{C}, \tag{2}$$

where $Y[m]$ represents data in frequency domain and $C$ represents the number of channels.

The polyphase filter reduces errors introduced by a discrete Fourier transformation, these are DFT leakage and DFT scalloping loss, depicted in Figs. 3 and 4. The polyphase filter can also serve for sample rate conversions and as a bandpass filter. Figs. 3 and 4