EURASIP Journal on Advances in Signal Processing
a SpringerOpen Journal

RESEARCH Open Access

Implementation of a high-throughput low-latency polyphase channelizer on GPUs

Scott C Kim* and Shuvra S Bhattacharyya

Abstract
A channelizer is used to separate users or channels in communication systems. A polyphase channelizer is a type of channelizer that uses polyphase filtering to filter, downsample, and downconvert simultaneously. With graphics processing unit (GPU) technology, we propose a novel GPU-based polyphase channelizer architecture that delivers high throughput. This architecture has advantages of providing reduced complexity and optimized parallel processing of many channels, while being configurable via software. This makes our approach and implementation particularly attractive for using GPUs as DSP accelerators for communication systems.

Keywords: DSP accelerator; GPU front-end receiver; GPU-based radio; Polyphase filter; Polyphase channelizer

1 Introduction

A modern communication transceiver contains two major components: a radio frequency integrated circuit (RFIC) and a baseband processor. An RFIC is responsible for conversion between analog and digital domain signals, and mixing signals up and down from baseband (BB) to some RF. A BB processor or a modem is responsible for handling all of the signal processing tasks and communication protocols. The notion of software radio (SWR) is defined in [1]. An SWR is responsible for the entire processing chain between RF and BB via software.

In SWR systems, signal processing (SP) tasks, such as signal conversion, mixing, resampling, and filtering, are all done in BB using software in the discrete-time sample domain. An RFIC is then responsible for direct conversion to and from RF.

A majority of the transceiver functionality is contained in the software modem. The goal of the software-based transceiver is to bring the SP functionality closer to the antenna as much as possible, reducing the burden on the RF front-end and utilizing the full flexibility of software. A software-based modem is particularly attractive over dedicated hardware solutions, such as ASIC- and FPGA-based solutions, due to significantly reduced design time from modeling to implementation to production. One of the major advantages of dedicated hardware is low-power design, but with the advancements made in system-on-chip (SoC) architectures over the years with particular emphasis on low-power design, solutions based on programmable SoC architectures can deliver levels of energy efficiency that are sufficient for many applications (e.g., see [2]). An SoC can be delivered as a complete solution that integrates not only the radio unit but other key units, such as central processing unit (CPU), graphics processing unit (GPU), and peripheral controller subsystems, as well.

A modern communication system requires multiple users and data streams to be processed simultaneously. A front-end transceiver must be able to transmit and receive multiple channels simultaneously, and a technique known as channelization is used to separate multiple users or channels from a single communication stream. A channelization process is responsible for three basic tasks: (1) signal up/down conversion (mixing), (2) sample rate change, and (3) filtering. In this paper, we refer to channelization in the receiver architecture only, which means that the channelization process is responsible for down conversion, reducing sample rates, and filtering to reject images at the receiver.

A straightforward approach to designing a channelizer is to design a bank of dedicated sub-receivers. Each sub-receiver is allocated to a single channel. Such an approach involves large costs in terms of area, power, and complexity. At the same time, the channels are in general not all

*Correspondence: sckim@umd.edu
Department of Electrical & Computer Engineering, University of Maryland, College Park, MD 20742, USA

used simultaneously, and many of the sub-receivers may be idle at any given time during operation. Such idle sub-receivers are wasteful in terms of power consumption and area.

A more integrated approach is needed to replace such a 'room full of receivers' to reduce redundancies and improve resource utilization. For this purpose, a polyphase channelizer was introduced in [3,4]. This architecture employed polyphase filter banks (PFBs) and discrete Fourier transform (DFT) operations to accomplish multiple channelization tasks at the same time. In particular, the PFB was used to perform inner-product (IP) computation for filtering and resampling at the same time, and DFTs were used for mixing signals up or down. We refer to a polyphase channelizer in the receiver chain as a polyphase down channelizer and in the transmitter as a polyphase up channelizer. In this paper, we focus on polyphase down channelizer implementation, which we will simply refer to as a polyphase channelizer (PCZ). The input to a PCZ is a frequency domain multiplexed (FDM) signal, and the output is a time domain multiplexed (TDM) signal. The input FDM signal can represent, for example, a dedicated channel for one user or a channel that is shared across multiple users using spreading codes within the channel.

In this paper, we demonstrate an important application of GPU technology to SWR systems. In particular, we develop a novel GPU-based polyphase channelizer architecture that delivers high throughput and provides reduced complexity and optimized parallel processing of many channels, while being configurable via software. Since BB modems require SP accelerators that are performing the same SP tasks on incoming streams of data, there are significant data and task parallelism available, which we exploit in our proposed architecture using the intensive parallel processing capability of a GPU. In our proposed design, the GPU can be used as a stand-alone unit or in conjunction with an existing hardware modem. Our goal is to use the GPU as a radio and bring it as close to the antenna as possible. Such a GPU-based system can reduce the burden on a power-hungry BB modem and ideally replace the existing modem altogether. Thus, our proposed channelizer architecture simplifies the design and enhances flexibility while providing significantly accelerated performances.

The remainder of this paper is organized as follows. First, we discuss the theory and operation of PCZ and the general-purpose GPU parallel programming language called Compute Unified Device Architecture (CUDA). We then introduce a novel GPU-based approach for high-throughput PCZ implementation.

Finally, we integrate all of the novel methods developed in this paper and demonstrate their utility using an important wireless communication standard.

2 Background information

In this section, we review the theory and operation of polyphase channelizers. In our system model, we assume baseband operation and we are given a wide system bandwidth (BW) that contains multiple channels. These channels do not overlap and are equally spaced apart. It is not our goal here to re-derive the system equations, but rather to focus on relevant design and architecture aspects in signal processing systems for parallel processing.

2.1 Polyphase channelizer

A PCZ combines multiple operations into an all-in-one design. A basic operation of down channelization is as follows. A downconversion of a channel-of-interest is performed, followed by a low-pass filter to reject adjacent channels. Finally, a downsampling is necessary to reduce the sampling rate to a Nyquist rate so that unnecessary computation is avoided. A PCZ has two major components: PFB and DFT. A PFB is a multirate filter that performs downsampling and low-pass filtering at the same time. A DFT is used to downconvert the output of PFB to baseband and allow the user to select desired channel indices. We will not derive or discuss the transformation of band-pass filters and mixers into a PCZ (for details on this, see [3]), but rather focus on design and implementation of an efficient PCZ architecture.

A prototype filter is designed that has a filter order of N. This 1D filter is reshaped, using polyphase decomposition [4,5], into a 2D polyphase filter. A polyphase filter has Q rows or filter banks, where each row has M columns or sub-filter coefficient taps. Therefore, the overall filter length N can be viewed as having a dimension of $Q \times M$. In addition, the number of DFT points equals the number of PFB rows, Q. In order to filter or perform convolution operations, a buffer is created to match the dimension of the PFB. This is an IP operation across each row of the PFB. A commutator is used to present the input sample in a bottom-up fashion as shown in [3,4] since this is a downsampling operation. Once all Q rows of samples have been inserted into the input buffer, the IP operation is performed.

The output of the IP produces a $Q \times 1$ vector. This vector is presented to the DFT for a downconversion operation. It is important to note that each bank or row is independent from the other rows, and as the input samples are presented one at a time, the IP operation can be performed on a per-row basis. Therefore, once the commutator reaches the top, the matrix multiplication between the input buffer and the filter coefficient matrix is complete. Since the row operations are independent of the others, they can be fully parallelized. In addition, if the input samples can be presented Q at a time instead of one sample at a time, the need for the commutator is eliminated. However, there is a trade-off in that more resources are required. For example, more multipliers are required in the commutator-free version to support parallel operations, whereas when the commutator is employed, a single multiplier can be shared across the entire matrix multiplication provided that all of the required operations can be completed prior to arrival of the next input sample.

Prior to applying the PCZ, we have an input sampling rate F_s^{in}, a given system BW, and a data rate R_d. After application of the PCZ, the input sampling rate is divided by Q and, similarly, the system BW is divided by Q, yielding equally spaced channels and a reduced sampling rate at the output of the PCZ. This is called a standard or maximally decimated polyphase channelizer [3]. In [3], an interpolation operation was combined with a maximally decimated PCZ to perform a rational resampling at the same time. This is a partially decimated PCZ. Since the interpolation in a partially decimated PCZ is being performed at some rate P while Q samples are being presented to the input, the output of the IP is presented $R = Q/P$ (R is referred to here as the rational resampling ratio) times faster than in the standard PCZ configuration. This in turn causes the need to shift the data at the input and output of the IP operation. A serpentine shift and circular shift are needed to provide such translations, which prevent phase shifts at the output of the DFT [3].

In summary, there are five factors involved in designing a PCZ: the (1) input sample rate F_s^{in}, (2) system bandwidth BW, (3) data rate R_d, (4) channel spacing $\Delta f = \text{BW} \div Q$, and (5) output sample rate $F_s^{\text{out}} = F_s^{\text{in}} \div Q$.

2.2 CUDA

NVIDIA introduced CUDA [6] as a parallel programming language for programming GPUs for use in graphics as well as in other computationally intensive application areas. CUDA is based on a single-instruction multiple-thread (SIMT) programming model, where multiple threads execute the same instructions over different data sets. The SIMT model provides an attractive model for implementation of SP algorithms.

A CUDA kernel is a grid set of blocks, and a block is a set of threads. A processing core is referred to as a streaming multiprocessor (SMX). A group of 32 threads is called a *warp* and is executed as a group inside an SMX. The total number of threads inside a block should be multiples of a warp. To achieve maximum performance, it is important to keep the GPU as busy as possible and utilize as many threads and blocks as possible.

The GPU memory hierarchy also needs to be utilized carefully to maximize performance. An external memory or global memory (GM) is the largest memory in a GPU system and is also the slowest memory. GM is commonly used to transfer data back and forth between the GPU and a corresponding host CPU. Shared memory (SM) is contained within an SMX and is visible only to a specific block of threads. It is a fast read/write local memory that can be viewed as a fast, user-enabled cache. A constant memory (CM) is a fast, read-only memory for storing constants. In addition, there are registers for local variables. Any register spills or arrays that are not in SM are stored in local memory (LM). It is important to utilize SM as much as possible since registers are limited and GM and LM are relatively slow.

There are several important features of CUDA that require careful attention when programming GPUs. First, memory transfers between CPU and GPU must be minimized due to the high latency of transferring data over the bus. Accesses to GM should be coalesced whenever possible, and SM should be used to avoid unnecessary access to GM. Grouping threads in multiples of a warp facilitates coalescing of data with GM and helps to enhance GPU utilization. The programmer typically profiles the application extensively to fine tune performance and identify bottlenecks. To summarize, a GPU programmer should design kernels to spread the workload as much as possible throughout the GPU, read from GM in a coalesced manner to an SM, instantiate sufficient numbers of threads per block (TPB), and write back results to GM in a coalesced manner.

3 Related work

We introduced the notion of using the GPU as a radio, particularly as a front-end transceiver, in [7], and in this paper, we continue to explore the concept of a GPU front-end (GFE) receiver. GPU back-end receivers, which are responsible for channel decoding (e.g., using Turbo and LDPC decoders), are captured in [8,9]. A modern GPU-powered communication system that uses multiple antenna configurations and a MIMO detector has been presented in [10].

Other related works on application of GPUs to communication system design include GPU acceleration of fast Fourier transform (FFT) computation for channelization [11], integrating GPU technology into a software radio framework [12], accelerating polyphase filters using GPUs [13], and channelization via mobile GPUs [14]. In [11,14], optimizing FFT and PFB for wideband channelization was introduced using OpenCL. This work investigated implementations that were targeted to different classes of AMD GPUs. It targeted GNU Radio's polyphase filter channelizer and compared the speedups and computation time between CPUs and different GPUs. In this work, we design and optimize our polyphase channelizer on NVIDIA GPUs using CUDA. Our primary goal is to target our implementation toward wireless communication systems and toward meeting critical performance constraints of such systems - in particular, constraints on throughput and latency.

Additionally, there are various FPGA/VLSI implementations of PCZs in the literature (e.g., see [15-19]). These works largely focus on optimizing resource usage, such as use of multipliers, and memory.

A preliminary version of this work was presented in [7]. This new paper goes beyond the developments of the preliminary version by incorporating significant new enhancements to our proposed polyphase channelizer architecture for high-throughput and real-time communication systems. Specifically, we further enforced coalesced loads and stores from GM to SM, spread work across the GPU more efficiently by enabling increased workloads and scheduling more blocks of threads for the GPU to process, and eliminated some sequential aspects of the underlying algorithms that were present in our preliminary version. Due to these enhancements, the execution time of our new architecture is significantly reduced, well below the target latency. Furthermore, the throughput has been increased significantly while providing for simultaneous processing of multiple channels.

4 GPU-based high-throughput polyphase channelizer

We map our PCZ algorithm onto a GPU and exploit parallelism found in PCZ operations. First, we implement a fully parallel PFB on a GPU, with the GPU used here to accelerate IP operations. We then integrate into our GPU implementation a CUDA fast Fourier transform (CUFFT) kernel. CUFFT, a part of NVIDIA's library of signal processing blocks, is a parallel version of the DFT that is highly optimized for use in CUDA. We process real I-Q values instead of complex values in our GPU implementation.

We demonstrated an approach to high-throughput IP computation using GPUs in [7,20]. In this approach, we are given an input array from the host CPU that is stored initially in GM. Instead of using an input buffer with dimensions $Q \times M$ to match the PFB, we simply index the necessary input samples. In order to minimize the usage of GM, we first load the data from GM into SM. Each SM contains M groups of samples so that the IP for multiple samples can be computed simultaneously. Since we can access Q samples at the same time, we no longer need a commutator. However, one must be careful to access the input samples in a bottom-up manner to ensure that processing is carried out in the correct order.

The algorithm presented in [7] uses a sequential for-loop to index through the input data. This is a simple approach to access Q samples at a time, but this serialization inside the GPU causes increased latency. If there is a large number of input samples to process, then this loop will dominate over the fast parallel processing inside the loop. We propose a new algorithm that eliminates this for-loop based processing. Specifically, instead of using a for-loop to step through the input, we unroll the loop completely and map each sample to a separate thread. This enforces the notion of SIMT processing in the GPU since we are performing the same operation over and over across the input data stream - i.e., polyphase filtering over the input data. This design optimization eliminates

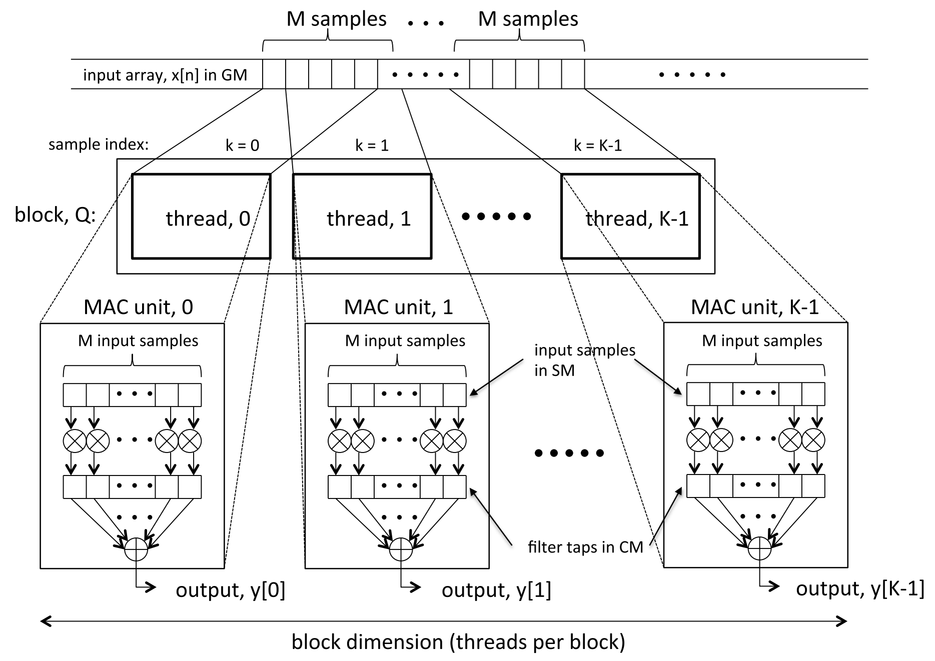

Figure 1 An example of how data is split and loaded onto the GPU.

sequential operation. Furthermore, because only one output sample is generated by each thread, the optimization enables the spawning of larger numbers of blocks and threads across the GPU, which helps to improve overall device utilization. Figure 1 shows how the data is split and loaded onto the targeted GPU.

Our objectives in further optimizing the design beyond the developments in [7] include exploiting the SM as much as possible rather than reading and writing excessively from and to GM. The optimized kernel design provides a larger workload that is spread across the threads, with each thread encapsulating a lightweight operation. Since the volume of input data exceeds the filter dimension, GPU utilization is increased. Each block or SM is loaded with TPB$+M-1$ input samples, and each thread is now responsible for filtering M samples. Therefore, in the optimized design, a thread encompasses a multiply-and-accumulate (MAC) operation, which also takes advantage of GPU's fused multiply-add (FMA) operation. This is compared to our previous design, where each thread performed multiplication only. The results of these multiplications were then summed separately using a single thread, which paused the other threads, leading to less efficient GPU utilization and loading of the GPU.

For polyphase filtering, we load the values column-wise, but we operate row-wise. Thus, when loading the data from GM to SM, the data is not coalesced properly, and an additional step is necessary to enforce further coalescing in the IP operation. A kernel is applied to shuffle the data to pre-position the data prior to polyphase filtering. A pseudocode specification of this data shuffling process is shown in Algorithm 1. It is important to note that we read the data in a linear fashion initially, to enforce caching on the read operation, then we write back to GM in polyphase decomposed fashion. This reduces the latency slightly compared to reading the data in polyphase decomposition manner first, and then writing it back linearly. Following this operation, the polyphase filter kernel is called. This kernel now reads the data linearly in a coalesced manner to SM. Since we instantiate threads that are multiples of a warp, the access pattern is byte aligned and linearly read, which further enhances the efficiency of the GPU implementation.

Algorithm 1 Pseudocode for data shuffling

```
$idx = blockIdx.x \times blockDim.x + threadIdx.x$
**if** $idx < INPUT\_LENGTH$ **then**
    $out[col + row \times SAMPLES\_PER\_ROW] = in[idx]$
**end if**
```

To demonstrate the overall operation of our proposed fully parallel PCZ design, we provide the pseudocode specification shown in Algorithm 2. Even with an extra kernel for data shuffling, the entire operation is now simplified and further streamlined by eliminating a dominant sequential loop from our previous design. We reshuffle the data back to its original format prior to applying CUFFT for the DFT. After application of CUFFT, the channelization process is complete. Each of the CUFFT output index corresponds to a TDM output stream. All of the the channel outputs are produced simultaneously due to the parallel structure of our proposed architecture. Figure 2 shows the overall block diagram of our new, optimized PCZ implementation.

Algorithm 2 Pseudocode for polyphase channelizer

```
$ix = threadIdx.x$
$iy = blockIdx.y$
$pdx = blockIdx.x \times blockDim.x + threadIdx.x$
$idx = iy \times SAMPLES\_PER\_ROW + pdx$
$odx = pdx \times Q + iy$
**if** $pdx < SAMPLES\_PER\_ROW$ **then**
    $SM\_REG[ix + M - 1] = in[idx]$
    **if** $ix < M - 1$ **then**
        $SM\_REG[ix] = in[idx - M + 1]$
    **end if**
    $SYNC\_THREADS$
    **for** $ii = 0$ **to** $M - 1$ **do**
        $SM\_MAC[ix] += CM\_COEF\big[(M-1-ii) \times Q + iy\big] \times$
        $SM\_REG[ix + M - 1 - ii]$
    **end for**
    $SYNC\_THREADS$
    $out[odx] = SM\_MAC[ix]$
**end if**
```

In summary, in this section, we have built on our recent developments on PCZ implementation [7] and incorporated additional design optimizations to further improve performance. The new design optimizations discussed here include minimizing the rate of data transfers, enhancing coalesced access of GM, optimized utilization of SM, and enhanced GPU utilization by reducing thread granularity (operation complexity). The result is a simpler architecture with reduced bottlenecks and elimination of a dominant sequential loop. Collectively, these optimizations result in significant further improvement in throughput and latency.

In the remainder of this paper, we demonstrate and experiment with our proposed design methods using a wireless communication standard. The results provide concrete insight into the the performance of our GPU-based, multichannel, parallel transceiver in the context of a practical wireless communication system.

5 Design and implementation

To experiment with our proposed new PCZ design, we target an important 3G wireless standard, the Universal

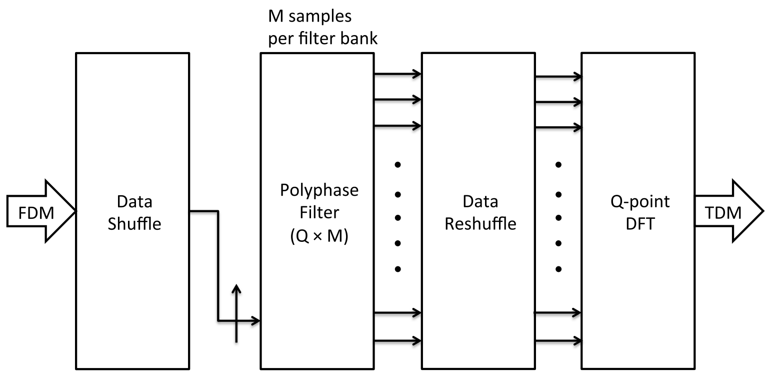

Figure 2 Block diagram of GPU optimized polyphase channelizer.

Mobile Telecommunications System (UMTS) air interface, wideband code division multiple access (WCDMA) [21].

The radio frame duration of WCDMA is 10 ms, which is further divided into 15 time slots per frame. In our experiments, we consider the front-end of a receiver at the base station or at the associated user equipment to evaluate our optimized PCZ design.

WCDMA/UMTS has a set of allocated frequency bands or operating band numbers. Each operating band has a center frequency and a BW associated with it. Each band can occupy several tens of megahertz, as much as 80 MHz. WCDMA is a spread spectrum system that has a data (chip) rate of 3.84 MHz and occupies approximately 5 MHz of BW. One of the common BW levels of UMTS is 60 MHz. Given that a modern RFIC can handle an instantaneous BW of more than 60 MHz, we assume that at the input to our GFE, a 60-MHz wide BW is presented. Within this wide BW, there can exist multiple WCDMA signals with 5-MHz channel spacing. Therefore, we have at most 12 WCDMA channels present, as shown in Figure 3. We process all of the available WCDMA channels (up to 12) simultaneously using our PCZ implementation.

Our approach to using PCZ here works well given a wide input BW, equal channel spacing, downconversion, and the ability to reduce the sampling rate at the output of the PCZ simultaneously using a prototype filter and DFT. In addition, this particular type of channelizer converts FDM channels into TDM channels. The GPU-based PCZ provides a highly parallelized and efficient channelization option, which we demonstrate in this paper for a realistic system scenario. More details on this demonstration are discussed in the following section, which covers experimental results and analysis.

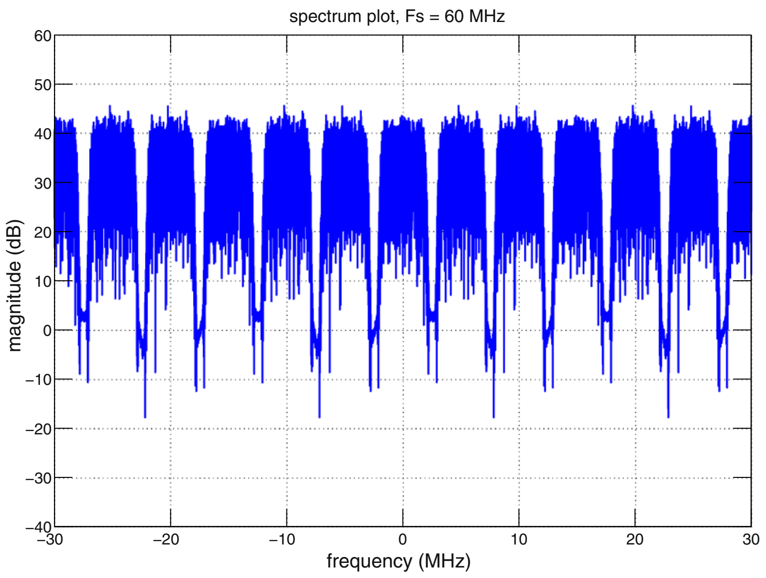

Figure 3 Plot of WCDMA channels. Twelve WCDMA channels are present in 60-MHz wide bandwidth.

We design our prototype filter using an equiripple FIR filter with order $N = 192$. We design the filter such that the passband covers the WCDMA band up to 3.84 MHz and the stopband is near 5 MHz. The passband ripples and stopband attenuation are 0.05 and 70 dB, respectively. This gives us non-overlapping polyphase filters, unlike the overlapping PCZ filter design that we presented in [7] for GSM. Since WCDMA uses QPSK modulation, it is important that we preserve the passband and that we do not overlap in the filter transition region, unlike GSM's GMSK modulation in [7]. A sample plot of our filter design is shown in Figure 4. Across our 60-MHz system BW, there is a total of 12 polyphase filters and channels side by side. This can be viewed as an example where a maximum number of channels is present in a band to maximize the network capacity.

Since there are 12 channels to process, we decompose our PCZ into 12 rows or PFBs, resulting in 16 sub-filter coefficients per row for our prototype filter length of 192. Therefore, we have a $Q \times M$ IP matrix, where $Q = 12$ and $M = 16$. The decimation rate or number of rows, Q, is also the number of DFT points. Here, $M = 16$, which is half of the GPU warp size. After application of the PCZ, the output sampling rate should be $60/Q$ or 5 MHz, which matches the desired channel spacing of WCDMA. Since we have Q input samples being presented and Q IP values for a Q-point DFT, which produces Q channel outputs, we have a maximally decimated channelizer. In addition, we process each channel independently, thereby realizing a fully parallel transceiver.

Now that we have all of the parameters in place, we map our WCDMA-targeted PCZ using the algorithm presented in the previous section. First, we prepare the input data for coalesced access with polyphase filtering. Given 60 MHz of BW and a 10-ms radio frame duration, we pre-process 600,000 samples into the desired 2D matrix for polyphase filter operation in the PCZ. We shuffle or reshape the 1D input array into a 2D matrix, as discussed earlier. This can be viewed as a row-major order that is transposed or simply a matrix that is loaded column first, but operated on across the rows, as given by the polyphase decomposition. This operation takes advantage of cached read accesses each time and block processing of input data. In addition, it enhances coalesced reads from GM to SM in the polyphase filtering process. This is implemented as a separate CUDA kernel prior to a filter kernel.

Earlier, we described our previous algorithm as being for-loop-dominant and indexing through the input array sequentially. Performance can be improved significantly when it is possible to parallelize this sequential process. For this purpose, we unrolled the loop completely and divided up the workload across more threads and blocks to utilize large numbers of blocks and cores in the targeted GPU.

While our original method had worked in the context of standards such as GSM which have longer radio frame durations of 120 ms, such a method is not well suited to 3GPP radio frames, which have durations that are 12 times shorter at 10 ms. Operation within 3GPP poses further challenges since faster processing is required. The new design presented in this paper maps the PCZ operation more efficiently for practical operation within the context of 3GPP. The workload is spread across the GPU

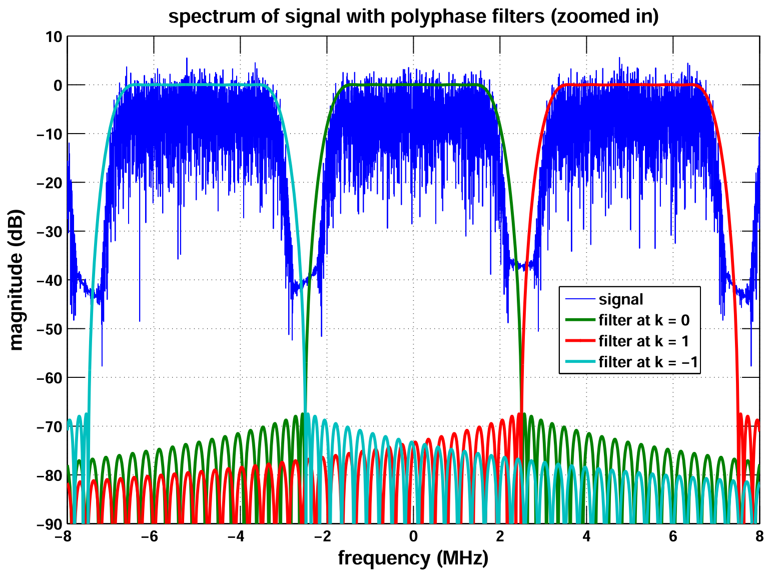

Figure 4 Plot of WCDMA channels with polyphase filters. A zoomed-in plot of three WCDMA channels with non-overlapping polyphase channelizer filters spaced apart at 5 MHz.

more evenly such that the GPU has a large number of lightweight threads operating, which helps to improve GPU utilization, as discussed earlier.

Next, we discuss the integration of the different components discussed above to construct our overall PCZ design. We exploit SIMT in our kernel dimension designs. We can reshape our kernel dimension each time, dividing up the workload evenly across different kernels. Multiple kernel calls are used to split the data for different purposes. First, our reshape or data shuffling kernel divides up one frame worth of data, composed of 600,000 input samples, by 512 TPB, yielding 1,172 blocks. This is a 1D split that assigns one thread per sample. For the PFB kernel, this same data set is divided into 12 channels, with 50,000 samples per channel. Each channel is then further divided by 512 TPB, which yields 98 blocks. Since there are 12 channels, we use a 2D data split, where the x-direction corresponds to individual samples within a channel, and the y-direction corresponds to specific channels. Thus, the total lengths of the x- and y-directions are N_x and N_y, respectively, where N_x is the number of samples per channel and N_y is the number of channels.

This decomposition maps 600,000 input samples into one sample per thread across the 2D grid dimension. Here, each thread is responsible for one sub-filter operation; therefore, it will compute $M = 16$ MAC operations per thread. Each input sample is read into SM along with $(M - 1)$ previous sample points to perform filter operation. The independent MAC operations are performed by each thread (i.e., 512 threads perform 512 MAC operations with each thread accessing 16 samples). This further improves the performance by increasing the utilization of SM instead of using registers or LM. The filter coefficients are stored in CM so that they are cached and broadcast throughout the entire kernel for fast, read-only operation. The output of the IP is then reshuffled when written back to GM for the subsequent CUFFT operation. The last kernel call in the sequence is the CUFFT, which outputs the TDM data for each of 12 individual channels at the same time.

Thus, to handle the more demanding processing required when applying our GPU-based PCZ design in 3GPP, we see that our new design employs much larger numbers of blocks and threads per block. In particular, under these design constraints, our previous design instantiates 12 blocks and 16 threads per block, which are sufficient for the GSM context in which the design was originally applied, but leaves large numbers of unused blocks and threads in the GPU. This low utilization of GPU resources is problematic under the more severe performance constraints of the 3GPP communication system targeted in this paper. Additionally, using multiple kernel calls provide significant flexibility to reshape the kernel dimension each time, which is a form of flexibility not found in dedicated hardware solutions.

6 Results and analysis

For our experiments, we employed NVIDIA's GeForce GTX 680 as the target GPU to implement our design. This GPU is based on the Kepler architecture, which has 1,536 CUDA cores (8 SMXs with 192 CUDA cores per SMX), 2 GB of GDDR5 memory, 64 kB of CM, and 48 kB of SM. It has 256-bit memory bus width, 192 GB/s bandwidth, and over 3,000 GFLOPS speed. We used the latest CUDA driver version 6.0 and a compute capability of 3.0. We summarize our results in Table 1 for both throughput and run-time. We also present our results with and without memory transfer between the CPU and GPU. Our target latency for real-time operation is 3GPP's radio frame length of 10 ms, and our calculated speedups are based on a sampling rate of 5 MSps (mega samples per second).

Our emphasis here is on high-throughput and low-latency PCZ implementation. We use 32-bit floating-point precision throughout the experiments. We test our implementation on a collected WCDMA band of 60 MHz for 10 ms, where each WCDMA channel occupies 5 MHz of BW. Thus, the overall 60-MHz bandwidth is channelized into 12 channels. We achieve a high throughput on all of the kernel calls. The PCZ kernel call includes CUFFT since CUFFT is highly optimized and available as a part of the CUDA software development kit (SDK). As one can see from the results, when memory transfer is involved, it dominates the overall run-time. Therefore, unnecessary transfer of data between the CPU and GPU is highly undesirable.

Our implementation results in no register spills, SM usage of 8,312 bytes, and CM usage of 360 bytes. As one can see from Table 1, all of our kernel calls are executed with performance that falls within our target latency of 10 ms, even with the overhead of data transfers taken into account. Without data transfers, the kernels ran under 1 ms. The overall kernel call (which is the slowest of all) achieves over 280× speedup compared to the 5-MSps sampling rate. Since this is a front-end and first stage of baseband processing, one would typically leave the data in the GPU for further processing and only transfer data back to the CPU when needed after such further processing is complete.

We note that the data shuffle kernel achieves an overall occupancy of 84.5%; a global memory load efficiency of

Table 1 Experimental results (run-time/throughput)

	With transfer	Without transfer
Shuffle	2.085 ms / 575.5 MSps	0.116 ms / 10,357.5 MSps
New PCZ	2.868 ms / 418.5 MSps	0.734 ms / 1,633.6 MSps
Overall	2.901 ms / 413.5 MSps	0.856 ms / 1,402.0 MSps

100%, which is as expected due to our use of linear coalesced reads; and a global memory store efficiency of 33%, which is due to shuffling of data for polyphase decomposition, as we discussed earlier. Nearly all of the kernel execution time is spent on load and store operations since there are no arithmetic operations involved.

The PCZ kernel, which encapsulates the PFB operation, achieves an overall occupancy of 95.6% and GM load efficiency of 99.8%. Due to our use of the data shuffle kernel, we are able to enforce coalescing for nearly 100% of the global read operations. Without our use of the data shuffle kernel here, we expect that the GM load efficiency would be much lower due to irregular read patterns for polyphase filtering. For the PCZ kernel, SM efficiency is 99.8%, but GM store efficiency is nearly 0%. This low level of GM store efficiency is expected; it is due to non-coalesced writes from reshuffling data after MAC operations across different blocks and threads.

The PFB operation utilizes full FMA floating-point operations in the kernel. The GPU excels in such computations [22]. Because our kernels under-utilize available computational resources to some extent, the kernels are memory bounded (at the L1 cache). However, this is not a major bottleneck in our implementation since our main goal is to process data channels in real-time, and the implementation meets these objectives in terms of throughput and latency. We note here that our experiments apply to a single instance of a data set rather than a continuous stream of data. Applying a continuous stream of data could lead to higher levels of utilization for the available computational resources. A useful direction for future work is the further exploration of the potential of our implementation in the context of continuously streaming data.

We compare the performance of our previous PCZ design [7] with the new design we that we propose in this paper. We emphasize that although our previous design (from [7]) exhibits lower performance compared to our new design, the previous design successfully met the performance constraints of GSM, which is the primary standard to which it was targeted. The new design introduced in this paper has been developed by building on the experience and insights gained from the previous design and targeting the more stringent constraints of 3GPP communication.

Table 2 demonstrates that the previous design cannot achieve the 3GPP real-time latency constraint of 10 ms, and in fact, its performance is at least 5 times slower than what is required for the communication system performance targeted in this work. In contrast, our new method achieves the real-time latency constraint, even with memory transfer, as discussed above. Furthermore, without memory transfer, the new design runs under 1 ms, which is near the slot time of 0.667 ms. In other words, the time to process a complete frame is less than 2 times the time required by a single slot in the frame. Overall, our new PCZ design achieves significantly improved throughput, and a speedup of 326× compared to the sampling rate of 5 MSps. The new design is also nearly 70× faster than the previous design, as shown in Table 2.

Table 2 Comparison of polyphase channelizer designs

	With transfer	Without transfer
Old PCZ	52.591 ms / 22.8 MSps	50.808 ms / 23.6 MSps
New PCZ	2.868 ms / 418.5 MSps	0.734 ms / 1,633.6 MSps
Speedup	18.3×	69.2×

As expected, the data shuffling kernel exhibits high performance, since it is a simple data swap; however, the new design allows us to combine this data swapping functionality with the benefits of caching and coalescing. Overall, the new PCZ design provides improvement over the previous one by eliminating serial processing of the for-loop and providing more thorough enforcement of memory coalescing. Additionally, the workload in the new design is spread much more evenly throughout the GPU, and each thread encompasses a fine-grained operation (MAC) utilizing GPU's FMA floating-point operations.

A limitation of our proposed new design is the output data rate, which must be increased with some amount of resampling to achieve the WCDMA data rate. Given the 5-MHz sampling rate at the output of the PCZ, we would need to resample to at least twice the sample rate of the WCDMA data rate, which is 3.84 MSps. Therefore, a resampler is needed to dynamically achieve such a fractional rate. We presented a GPU-based arbitrary resampling method using polyphase filters in [7]. However, due to serialization within the underlying resampling approach, this approach does not allow us to achieve the target latency when it is integrated with our new PCZ design. Integrating a suitable resampling subsystem at the output of our proposed new PCZ design is a useful direction for further investigation.

7 Conclusions

In this paper, we have presented a novel GPU-based polyphase channelizer that achieves high throughput and low latency. The new architecture eliminates sequential processing and spreads the processing workload evenly across large numbers of blocks and threads in the targeted GPU. Furthermore, the design incorporates preprocessing of the input data to thoroughly enforce caching and coalescing prior to polyphase filter operation. We demonstrated our application of GPU technology as the basis for front-end transceiver implementation by processing multiple channels simultaneously and exploiting data parallelism across different channels, which provides large increases in throughput.

Our proposed new GPU-based polyphase channelizer design provides the high performance of a dedicated single receiver using a fully integrated receiver structure and without sacrificing flexibility. We demonstrated our design on an important wireless communication standard and demonstrated large speedups in both throughput and latency. We also compared the performance to that of a previous polyphase channelizer design and demonstrated nearly 70× improvement, while providing detailed analysis of how such speedup improvements have been obtained.

Collectively, the advances presented in this paper make use of off-the-shelf GPU devices as floating-point software radios that can compete in many design scenarios with fixed-point dedicated hardware radios, and help to bring GPUs one step closer to the antenna.

Competing interests
The authors declare that they have no competing interests.

Received: 5 March 2014 Accepted: 30 August 2014
Published: 15 September 2014

References

1. E Venosa, FJ Harris, FAN Palmieri, *Software Radio: Sampling Rate Selection, Design and Synchronization (Analog Circuits and Signal Processing)*. (Springer, New York, USA, 2011)
2. SS Bhattacharyya, E Deprettere, R Leupers, J Takala (eds.), *Handbook of Signal Processing Systems*, 2nd edn. (Springer, New York, USA, 2013). ISBN: 978-1-4614-6858-5 (Print); 978-1-4614-6859-2 (Online)
3. FJ Harris, C Dick, M Rice, Digital receivers and transmitters using polyphase filter banks for wireless communications. IEEE Trans. Microwave Theory Tech. **51**(4), 1395–1412 (2003)
4. FJ Harris, *Multirate Signal Processing for Communication Systems*. (Prentice Hall, New Jersey, USA, 2004)
5. PP Vaidyanathan, *Multirate Systems and Filter Banks*. (Prentice Hall, New Jersey, USA, 1993)
6. NVIDIA Corporation, *CUDA C Programming Guide*. (NVIDIA, 2013). http://docs.nvidia.com/
7. SC Kim, WL Plishker, SS Bhattacharyya, An efficient GPU implementation of an arbitrary resampling polyphase channelizer, in *Proceedings of the Conference on Design and Architectures for Signal and Image Processing* (Cagliari, Italy, 8 October 2013), pp. 231–238
8. M Wu, Y Sun, G Wang, JR Cavallaro, Implementation of a high throughput 3GPP turbo decoder on GPU. J. Signal Process. Syst. **65**(2), 171–183 (2011)
9. G Wang, M Wu, B Yin, JR Cavallaro, High throughput low latency LDPC decoding on GPU for SDR systems, in *Proceedings of IEEE Global Conference on Signal and Information Processing* (Austin, Texas, USA, 3 December 2013), pp. 1258–1261
10. M Wu, Y Sun, S Gupta, JR Cavallaro, Implementation of a high throughput soft MIMO detector on GPU. J. Signal Process. Syst. **64**(1), 123–136 (2011)
11. C del Mundo, V Adhinarayanan, W-C Feng, Accelerating fast Fourier transform for wideband channelization, in *IEEE International Conference on Communications* (Budapest, Hungary, 9 June 2013), pp. 4776–4780
12. P-H Horrein, C Hennebert, F Pétrot, Integration of GPU computing in a software radio environment. J. Signal Process. Syst. **69**(1), 55–65 (2012)
13. K van der Veldt, R van Nieuwpoort, AL Varbanescu, C Jesshope, A polyphase filter for GPUs and multi-core processors, in *Proceedings of the Workshop on High-Performance Computing for Astronomy* (Delft, Netherlands, 18 June 2012), pp. 33–40
14. V Adhinarayanan, W-C Feng, Wideband channelization for software-defined radio via mobile graphics processors, in *Proceedings of International Conference on Parallel and Distributed Systems* (Seoul, Korea, 15 December 2013), pp. 86–93
15. Y Wang, H Mahmoodi, L-Y Chiou, H Choo, J Park, W Jeong, K Roy, Energy-efficient hardware architecture and VLSI implementation of a polyphase channelizer with applications to subband adaptive filtering. J. Signal Process. Syst. **58**(2), 125–137 (2010)
16. M Awan, P Koch, C Dick, F Harris, FPGA implementation analysis of polyphase channelizer performing sample rate change required for both matched filtering and channel frequency spacing, in *Proceedings of the IEEE Asilomar Conference on Signals, Systems, and Computers* (Pacific Grove, CA, USA, 7 November 2010), pp. 414–418
17. M Awan, P Koch, Polyphase channelizer as bandpass filters in multi-standard software defined radios, in *Proceedings of the International Workshop on Cognitive Radio and Advanced Spectrum Management* (Aalborg, Denmark, 18 May 2009), pp. 59–63
18. M Awan, MM Alam, P Koch, N Behjou, Area efficient implementation of polyphase channelizer for multi-standard software radio receiver, in *Proceedings of the Karlsruhe Workshop on Software Radios* (Karlsruhe, Germany, 5 March 2008), pp. 123–130
19. R Mahesh, AP Vinod, M-K ELai, A Omondi, Filter bank channelizers for multi-standard software defined radio receivers. J. Signal Process. Syst. **62**(2), 157–171 (2011)
20. SC Kim, WL Plishker, SS Bhattacharyya, JR Cavallaro, GPU-based acceleration of symbol timing recovery, in *Proceedings of the Conference on Design and Architectures for Signal and Image Processing* (Karlsruhe, Germany, 23 October 2012), pp. 1–8
21. H Holma, A Toskala (eds.), *WCDMA for UMTS: HSPA Evolution and LTE*. (Wiley, NJ, USA, 2007)
22. N Whitehead, A Fit-Florea, Precision & performance: floating point and IEEE 754 compliance for NVIDIA GPUs, NVIDIA Technical White Paper.

doi:10.1186/1687-6180-2014-141
Cite this article as: Kim and Bhattacharyya: **Implementation of a high-throughput low-latency polyphase channelizer on GPUs.** *EURASIP Journal on Advances in Signal Processing* 2014 **2014**:141.