On the Performance and Energy Efficiency of FPGAs and GPUs for Polyphase Channelization

Vignesh Adhinarayanan*, Thaddeus Koehn†, Krzysztof Kepa†, Wu-chun Feng*†, Peter Athanas†
*Department of Computer Science, †Department of Electrical and Computer Engineering
Virginia Polytechnic Institute and State University, Blacksburg, U.S.A.
{avignesh, tkoehn, kepa, wfeng, athanas}@vt.edu

***Abstract*—Wideband channelization is an important and computationally demanding task in the front-end subsystem of several software-defined radios (SDRs). The hardware that supports this task should provide high performance, consume low power, and allow flexible implementations. Several classes of devices have been explored in the past, with the FPGA proving to be the most popular as it reasonably satisfies all three requirements. However, the growing presence of low-power mobile GPUs holds much promise with improved flexibility for instant adaptation to different standards.**

Thus, in this paper, we present optimized polyphase channelizations for the FPGA and GPU, respectively, that must consider power and accuracy requirements in the context of a military application. The *performance* in mega-samples per second (MSPS) and *energy efficiency* in MSPS/watt are compared between the two classes of hardware platforms: FPGA and GPU. The results show that by exploiting the flexible datapath width of FPGAs, FPGA implementations generally deliver an order-of-magnitude better performance and energy efficiency over fixed-width GPU architectures.

I. Introduction

Wideband channelizers are widely used in software-defined radios (SDRs) to extract channels from a wideband spectrum. Military applications such as communication intelligence (COMINT) and signal intelligence (SIGINT) use these channelizers for real-time demodulation of wideband spread spectrum signals [1]. For the channelizers to be useful in such scenarios, the underlying hardware should be capable of providing high performance under small power budgets while allowing flexibility through reconfiguration [2]–[5].

Past research with several devices such as ASICs, multi-core CPUs, CellBE, and GPGPUs have shown that utmost two out of the three requirements (performance, power, and flexibility) can be met simultaneously [6]–[9]. FPGAs, on the other hand, can provide the necessary performance under the required power budget. Though adapting to different standards instantaneously remains a problem for the FPGAs owing to the time consuming reconfiguration process, static reconfiguration at the site remains an option. Considering the limitations of the alternatives, FPGAs remain the most popular choice.

Advances in mobile graphics processors promise high performance, low power consumption and instant adaptability to different standards required for SDRs with the additional advantage of significantly reduced monetary costs. However, the application characteristics can significantly affect the performance and energy efficiency obtained in practice. We seek to understand the differences in performance and energy efficiency in low-power FPGAs and mobile GPUs while performing wideband channelization using a polyphase filter bank (PFB) channelizer for a range of problem sizes. Our key contributions include the following:

- Optimized circuit architectures for FPGA implementations of a polyphase channelizer that trades off accuracy for performance.
- An optimized implementation of the PFB channelizer on a discrete GPU and an integrated GPU.
- A comparison of FPGAs and GPUs with respect to performance, energy efficiency (i.e., performance per watt), and performance per dollar at varying channel sizes and precision levels.

The results show that the optimized FPGA implementation outperforms the optimized GPU implementations by a factor of 11.0-12.3 in performance and by a factor of 7.8-12.0 in energy efficiency in most cases. The main reason for this large difference is that GPUs are optimized for floating-point performance and cannot obtain benefits by using fixed-point reduced-precision computations when applications require only limited accuracy, such as in PFB channelization [6]. In terms of performance per dollar, the GPUs were 2.35-fold better than the FPGAs.

The rest of the paper is organized as follows. Section II covers background on the PFB channelizer, GPU architecture, and FPGA architecture. The optimized FPGA and GPU implementations are presented in Sections III and IV, respectively. Results are discussed in V. Related work is presented in Section VI and conclusion in Section VII.

II. Background

In this section, we present the following background material to provide context for the rest of the paper: polyphase filter bank (PFB) channelization, FPGA architecture, and GPU architecture.

A. Polyphase Filter Bank (PFB) Channelization

Polyphase filter bank (PFB) channelization (Fig. 1) is an efficient technique to extract channels from a wideband sig-

Adhinarayanan and Koehn contributed to this work equally.

nal [10]. The major stages within PFB are briefly described below:

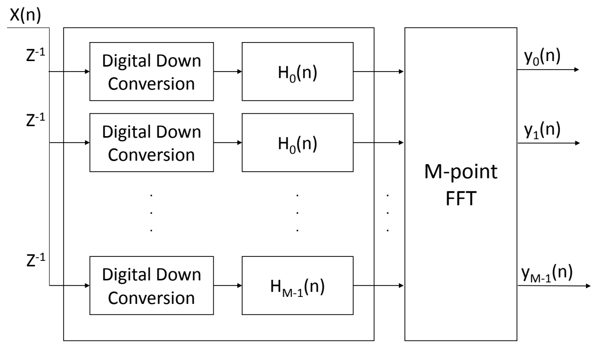

Fig. 1. **Polyphase filter bank (PFB) channelizer.** Wideband input signal is divided into M equally spaced channels. The first block represents FIR filter banks and the second block represents FFT.

FIR Filtering. In the first stage, a number of finite impulse response (FIR) filters operate in parallel. Each filter performs a dot product on the filter taps and input signal. The output of an FIR filter can be mathematically written as

$$y(n) = \sum_{i=1}^{T} t_i x(n-i)$$

where $x(n)$ and $y(n)$ represent input and output signals at time n; t_i, the filter coefficients; T, the number of taps.

FFT. The fast Fourier transform (FFT) transforms the input signal from the time domain to the frequency domain. As shown in Fig. 2, this stage has the highest computational complexity requiring $O(N\,log\,N)$ computations per iteration, where N is the number of points, and takes the most time. Mathematically, the output of an FFT operation can be written as

$$X_k = \sum_{n=0}^{N-1} x(n) e^{-i2\pi k \frac{n}{N}}$$

where $x(n)$ and $X(k)$ represent input and output.

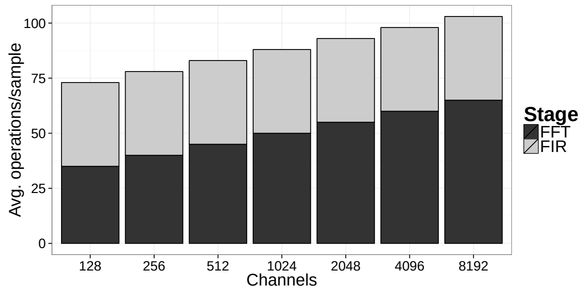

Fig. 2. **Number of operations per sample vs. number of channels.**

B. FPGA Architecture

FPGAs contain large amounts of configurable logic and memory along with a significant amount of silicon area dedicated to interconnects. Modern devices also contain highly pipelined multipliers that have been optimized to perform at clock speeds of 500MHz and greater. In the Virtex 6 and 7, pipelined multipliers are combined with both pre- and post-adders specifically for digital signal processing (DSP) applications. They are especially useful in filters where the additions can be performed in dedicated hardware.

Implementing an algorithm involves combining these resources in an efficient manner such that resources are balanced and timing constraints can be met. There are many tools for FPGA design. At a low level, similar to assembly for a CPU, are hardware description languages (HDL), which describe the hardware at the register-transfer level (RTL). Dedicated hardware components, such as block memory or multiplier, can be specified or can be inferred in synthesis.

There are also many tools that have been developed to decrease the time and difficulty of using such a low-level language. These tools include pre-generated cores as well as graphical user interfaces (GUIs), like the Xilinx System Generator, which abstract the hardware to a higher level. At this level, the designer is generally working with higher-level components such as an FFT block or an FIR filter block. The system generator is a library of higher-level components; they have been parametrized to allow the designer to construct their design more easily. These tools can save a significant amount of time, but they restrict the flexibility of the device, and thus, an optimal algorithm may be impossible at this level. In practical terms, this results in a design that may require more power. Also, these tools are impractical for designs requiring throughputs greater than the base clock speed.

C. GPU Architecture

GPUs consist of one or more compute units (CU) and each CU consists of an array of processing elements (PE). In our target devices, each PE is made up of multiple ALUs, which can all be simultaneously used via a VLIW instruction. The GPU's memory hierarchy is made up of five types of memory: private, constant, local, image, global. The peak read bandwidth of the different types of memory is shown in Table I.

Memory Type	Peak Read BW
Private	48 bytes/cycle
Constant	16 bytes/cycle
Local	8 bytes/cycle
Image	4 bytes/cycle
Global	0.6 bytes/cycle

TABLE I
PEAK READ BANDWIDTH OF DIFFERENT MEMORY TYPES

GPUs can be programmed through OpenCL, which is an open standard framework for programming a variety of devices, including CPUs, APUs (i.e., accelerated processing units), and co-processors like Intel Xeon Phi. An application written in OpenCL is composed of several OpenCL threads known as work-items. The application is mapped onto the GPU as follows. Work-items map to PEs with multiple ALUs, which makes vectorization necessary. A group of work-items make up a work-group and can share the fast local memory for communication within a work-group. A collection of 64 work-items (in AMD GPUs) make a wavefront which is mapped to a CU. To obtain good performance, an OpenCL application should leverage the GPU's memory hierarchy and use at least as many wavefronts as CUs and as many work-items as PEs.

III. FPGA Implementation

This section describes the FPGA implementation and design choices. When implementing an algorithm, the first consideration is the required data rate and clock speed. If the required data rate is higher than the clock rate, then the data will have to be processed in parallel. A higher clock rate will use fewer parallel processes, overall reducing power.

We designed two channelizer for data rates of 1000 and 4000 MSPS, respectively. The processing was parallelized four times (4x) and sixteen times (16x), respectively, with a clock rate of 250 MHz.

A. FIR Filter

Figure 3 shows a FIR filter implemented in a pipelined direct-form filter. This form lends itself well to polyphase decompositions as a single output is computed per pipe with no delays in the multiply-accumulate (MAC) chain. In addition, the add chain can take advantage of the post-adder in the DSP slice, which is already used for the multiply without requiring an extra adder tree. The adder tree would require additional logic resources and is not as power-efficient as the adder in the DSP. Furthermore, because the DSPs are built with 48-bit adders, the FIR computation can delay truncation until the last stage with minimal impact on resource and power utilization.

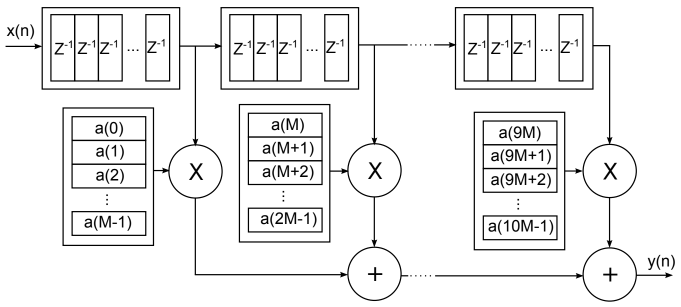

Fig. 3. **Direct Form Multi-MAC FIR Filter** The filter has 10-taps per channel, with M channels. This form utilizes the post-adder of the DSP48E.

The coefficients are stored in block memory, while the delay lines are written such that the synthesis tool can infer either distributed memory or block memory depending on the required delay. The advantage of distributed memory is that for short delay lines, only a pipeline of registers is required. Block memory requires address controllers and, depending on the size of delay, may underutilize the memory; memory blocks are available only in a small number of sizes.

B. FFT Design

The FFT was implemented as a radix-2 decimation in frequency algorithm. This algorithm has several benefits to the channelizer. First, because it breaks each FFT into a smaller FFT, each a power of two, it allows the channelizer size to be easily parameterized into any power of two. Also for the 1000 MSPS, which is parallelized at 4x, the radix-2 algorithm processes four samples at a time and requires minimal control logic as a result. For the 16x channelizer, a higher-radix algorithm might be more optimal, but as the radix increases, the control and data switching become more complicated.

Each stage of the FFT consists of a data routing switch and a layer of butterfly processing units. The data routing requires that for the Nth stage that the first and $N/2+1$ sample be both added and subtracted, followed by the second and $N/2+2$ sample. The data router takes the data in either sequential order or the order output from the previous stage and presents the data to the butterfly processors in the desired order.

Each butterfly unit takes in two complex inputs, computes both the sum and difference, and multiplies the difference by a root of unity or "twiddle factor." The twiddle factors are stored in a ROM and are parameterized to any number of bits in length. After each butterfly, an optional shift-right is available. Since the FFT is a fixed-point implementation, without this mechanism the data in the FFT can easily overflow. After the optional shift, the data is convergently rounded. This method treats positive and negative values symmetrically, and therefore is free of overall bias if the original numbers are positive or negative with equal probability. Asymmetric rounding methods accumulate error and result in DC bias.

Each non-trivial complex multiply $R + jI = (X + jY) \times (C + jS)$ can be implemented in three multipliers. By doing some factoring, the real and complex terms share a common term $Z = C(X - Y)$. Precomputing Z allows the number of multiplies to be reduced at the expense of an add:

$$R + jI = \ Y(C - S) + Z + j\left[X(C + S) - Z\right].$$

This reduction is particularly well suited for the Xilinx DSP48e blocks with their pre- and post-adders that allow the adds and subtracts to be performed within the DSP block.

The last few stages of the FFT have simple twiddle factors that reduce to a scalar multiply or a negation. In these stages, a modified butterfly has been substituted, which reduces the butterfly to the minimum number of computations. In these last stages, the routing is also greatly simplified as the required data samples all appear within one or two clocks.

The last part of the FFT is optional depending on the application. It takes the data from the last FFT in bit-reversed order and sorts it into sequential order. Bit-reversed order means converting the sample number, starting at zero, to binary and flipping all the bits left to right. For example, if the FFT size is 4096, sample 1 would become sample 4095, and sample 4095 would become sample 1. As the FFT size increases, the bit-ordering RAM size increases proportionally.

IV. GPU Implementation

This section describes the GPU implementation and architecture-specific optimizations.

A. System-wide Optimizations

FIR Implementation. In this GPU implementation, one GPU thread (work-item) computes one element in the output array, which represents one sample of an output channel. This is done to run many threads on the GPU cores simultaneously to hide GPU's high memory latency. The threads are organized such that the threads within a work-group all compute on samples that use the same set of taps. Such an organization ensures that there is high data reuse within a work-group. *Local memory*, which is shared by work-items within a work-group, is used to store the input elements temporarily. The data is laid out in the memory (global and local) to ensure *coalesced accesses* and to reduce bank conflicts, as shown in Fig. 4. *Constant memory* is used to store the filter taps. Both local memory and constant memory are size limited, and we employ a *blocking technique* for larger data sizes [11].

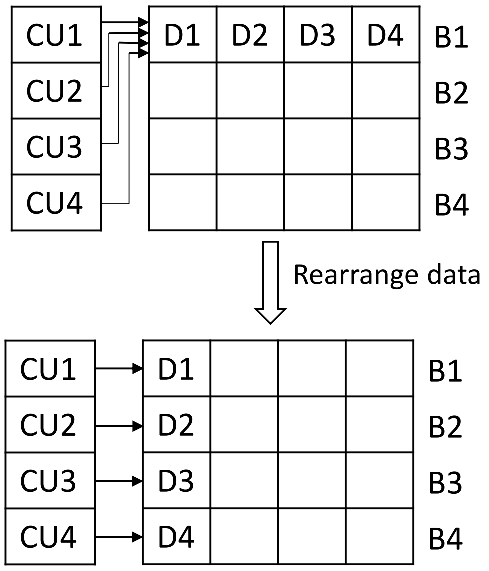

Fig. 4. **Data organizaztion.** In the default format, the data (D_i) is accessed from a single memory bank (B_i), which serializes the memory accesses. In the optimized version, the data is rearranged in the memory, so that the computational units (CU_i) can access them in parallel.

The GPUs used in this evaluation all have VLIW architectures. In these architectures, loading memory in vector types is faster than loading an equivalent number of scalars. Similarly, using *vector types* for computations ensures that the full potential of the VLIW architecture is unlocked. The `float4` data type is used for loading and performing mathematical operations. This also partially *unrolls loops*, which results in a reduced dynamic instruction count. Fused *multiply-add operations* are used to perform the multiply-accumulate operations of the FIR filters.

FFT Implementation. This PFB implementation makes use of the AMD FFT library for the FFT stage. This library employs an autotuner to find an efficient implementation from a design space of optimizations. Experiments revealed that an out-of-place algorithm consistently outperforms an in-place algorithm but at the cost of a larger memory footprint. This design uses an out-of-place algorithm, with the other architecture-specific optimizations decided by the autotuner.

B. Data Transfer Optimization

Data streaming is employed to overcome the data transfer overhead associated with transferring the data to and from the GPU through the PCI Express interface. In this technique, the data transfer associated with one iteration is overlapped with the computations in another iteration. This is depicted in Fig. 5, where the FFT stage is overlapped with the host-to-device transfer and the FIR stage is overlapped with the device-to-host transfer. This implementation avoids the double-buffering overhead that is normally associated with data streaming [9].

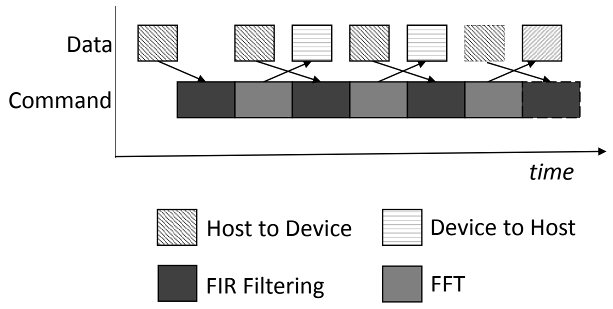

Fig. 5. **Data Streaming.** Two separate queues used for kernel execution (command) and data transfer (data). Dependencies across stages shown using arrows.

C. Other Implementations

The floating point GPU implementations provides a higher accuracy than what is demanded by the target application, which in turn affects the performance obtained and the power consumed. Therefore, implementations making use of `int` and `short` data types are evaluated, which represents fixed-point precisions comparable to the FPGA implementations that are designed to meet the accuracy requirements of the PFB channelizer.

V. Results and Discussion

In this section, we present the experimental results from our channelizer on a FPGA and on a GPU. We then discuss the different trade-offs between realizing the channelizer on a FPGA versus a GPU.

A. FPGA Evaluation

1) Experimental Platform: The performance of the FPGA implementation of a PFB channelizer is evaluated on a Hitech Global HTG-V7-PCIE board that is populated with a Virtex-7 XC7VX690T-2FFG1761 FPGA. The input is provided by an analog-to-digital converter, which provides data at either a rate of 1000 MSPS or 4000 MSPS. The output data channel is selectable via a Gen2 x4 PCIe interface.

The bit stream was generated using Xilinx's ISE Design Suite 14.6. Current and voltage were measured simultaneously at the input power connector to the board using two digital multimeters. The power was measured with the channelizer running and with the channelizer disconnected from clocks and data changes. This allowed us determine a baseline, where the difference between the power draws determines how much of the power was actually used by the channelizer as opposed to the other peripherals on the board.

2) FPGA Results: Figure 6 shows an increase in power with an increased number of channels for both the 1000 MSPS and 4000 MSPS PFB channelizers. This is consistent with expectations; the number of FFT stages is logarithmic in channel size, so resource and power usage is roughly linear on the log scale. The size of the ROMs for the filter coefficients and twiddle factors increases linearly with channel size. For the lower channel sizes, the logarithmic power increases dominate, but there are certain steps when power increased slightly more than linear when the synthesis tool replaced distributed memory with larger BRAM modules.

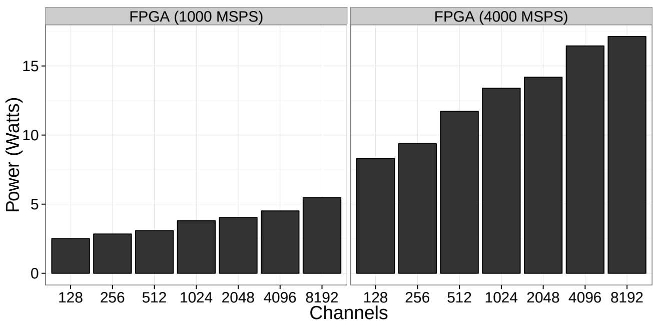

Fig. 6. **FPGA power consumption at 1000MSPS and 4000MSPS.**

Figure 7 shows the energy efficiency in MSPS per watt versus the number of channels for both FPGA channelizers. The energy efficiency decreases with the number of channels for both FPGA channelizers. This is because of the increased computations required for the larger channelizer size without a corresponding increase in the samples per second.

B. GPU Evaluation

1) Experimental Platforms: The performance and energy efficiency of the OpenCL implementation of a PFB channelizer are evaluated on an AMD Radeon HD 6480G (integrated GPU) and AMD Radeon HD 6470M (discrete mobile GPU). Table II shows the details of the hardware platforms.

The details of the software platform are as follows. OpenCL v1.1 and the AMD APP v2.8 SDK are used for the channelizer implementation. For the FFT, the accelerated parallel math library for FFT, clAmdFft-1.10.274, from AMD was employed. All implementations run on 64-bit Windows 7 OS.

The discrete GPU is connected to the host via second-generation PCIe x16. While experiments were conducted with different core-frequency settings, the performance, power, and energy efficiency are reported only for the highest frequency

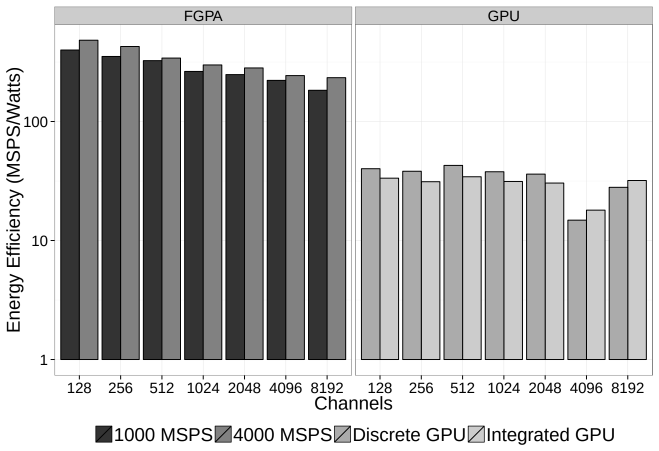

Fig. 7. Energy Efficiency (in MSPS/watt) of FPGA and GPUs.

H/W Type	Integrated GPU	Discrete GPU
Platform	HD 6480G	HD 6470M
Compute Units	3	2
Total Cores	240	160
Core clock	444 MHz	750 MHz
Memory size	512 MB (shared)	1024 MB
Memory clock	675 MHz	800 MHz
Memory type	DDR3	DDR3
TDP	9 W	9 W

TABLE II
HARDWARE PLATFORM CHARACTERISTICS

setting, as shown in Table II, as this setting corresponds to the most energy-efficient design. The dynamic power is measured through a "Watts Up? Pro" power meter. The static power is estimated based on the observation that static power contributes nearly one-third of the total power consumption for processors manufactured using 32 nm process technology [12]. Results reported in this section are median values of five runs.

2) GPU Results: Fig. 8 shows the performance obtained from the integrated GPU and the discrete GPU for three different implementations with varying accuracy: floating point, integer, and short integer. In the best case, a throughput of up to 326 MSPS on the integrated GPU and up to 375 MSPS on the discrete GPU was obtained. Across the channel sizes, the channelizer consumed 9.5 W and 9.0 W of power for the integrated GPU and the discrete GPU, respectively.

Owing to its slightly higher peak performance, the discrete GPU consistently outperformed the integrated GPU for up to 2048 channels. For larger channel sizes, the integrated GPU performs better. Also, for these problem sizes, the overall performance obtained drops significantly. The experiments revealed the performance obtained for the FIR stage remained consistent across problem sizes, while the performance obtained for the FFT stage dropped significantly. An analysis with the AMD APP Profiler [13] shows that for these problem sizes, the autotuner picks implementations that results in higher bank conflicts and cache misses. Also, the performance does not improve when accuracy is sacrificed through fixed-

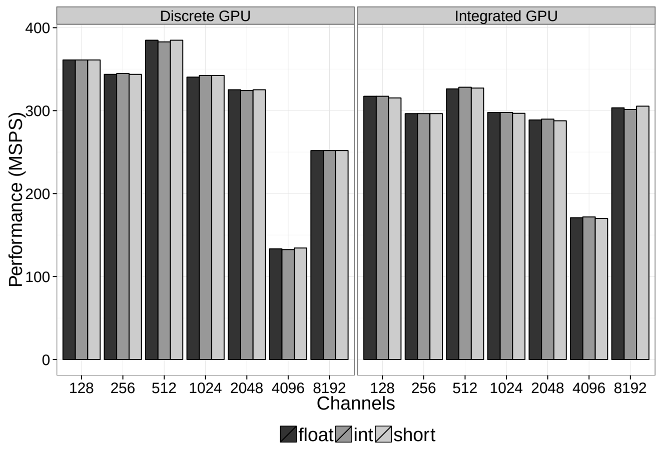

Fig. 8. Performance (in MSPS) obtained from GPUs.

point and reduced-precision implementations. Fig. 7 presents the energy efficiency of the implementations. The FPGA outperforms the GPU by 7.8 to 12.0 times in most cases.

C. Discussion

FPGAs can take advantage of the fact that the target application requires only limited accuracy. By limiting the bit-width for the adders and the multipliers, more computational units can be packed into the same power and area budget. However, such flexibility is not available in the GPUs. Though the programming model for GPUs provides a way to specify computations with limited accuracy through data types, e.g., `int` and `short`, the underlying hardware is floating point and cannot exploit the applications' requirements. This can be seen from Fig. 8 where there is no improvement in performance from reducing precision. However, techniques that pack several data elements into a single word may help perform multiple computations within a single fixed-width computational unit. Another reason for the relatively poor energy efficiency of the GPUs is the lack of energy proportionality. The application, when operating at only 15% utilization, consumes nearly 100% of the peak power.

Cost and Programmability. Apart from performance, power, and energy efficiency, a few other factors such as device cost, performance per dollar, and productivity governs the choice of device. High-end FPGAs cost tens of thousands of dollars whereas high-end GPUs cost only a few thousand dollars [14]. For the Xilinx Virtex-6 FPGA and the AMD Radeon HD6470M GPU under study, the market prices are $1995 and $80, respectively. This means that the performance per dollar for the discrete GPU (4.69 MSPS/dollar) is *2.35 times* better than the FPGA (2.00 MSPS/dollar) in the best case. On a per-dollar basis, the GPU delivers better results than the FPGA, even though the FPGA outperforms the GPUs in terms of raw performance.

While we do not explicitly compare the ease of programmability of the two devices, the general consensus is that the productivity for FPGAs is significantly lower than for general-purpose GPU computing devices [15], [16].

VI. Related Work

Many studies have sought to accelerate the compute-intensive wideband channelization. Song et al. [17] present an ASIC implementation of the polyphase channelizer that operates at an energy efficiency of 41 GOPS per watt. More recently, Wang et al. [6] present an ASIC implementation with power optimizations that deliver a throughput of 480 MSPS while consuming 352 mW of power. Though these designs are more energy efficient than our implementations, ASICs lack the flexibility required for SDR application.

A number of FPGA implementations of PFB channelizers have been presented in the literature [18]–[20]. However, it is difficult to draw comparisons from these studies as they explore a different set of problem sizes and apply to a different set of application areas. Savir [1] presents a scalable FPGA implementation for military applications; but our FPGA implementation scales better and delivers better performance.

Implementations on the Cell Broadband Engine have shown poor performance [7]. Implementations of SDR applications on experimental DSPs, on the other hand, have shown excellent performance and energy efficiency [21]–[23]. An important takeaway from these studies is that SIMD-based architectures are well suited for these applications. Similar benefits can be expected from SIMD-based GPU architectures, which also provide better programmability.

Researchers have started to adopt GPUs for accelerating various stages of SDR pipeline, including wideband channelization. Kim et al. [8] present a GPU implementation of the polyphase channelizer for a different SDR pipeline. van der Veldt et al. [24] investigate the acceleration of a PFB channelizer using high-powered GPUs and multicore processors for radio astronomy applications. While they report a high efficiency in the computational stages, their overall performance gains were adversely affected from the data-transfer overhead. The GPU implementations here could show high performance gains by hiding almost all of the data transfers. While a number of attempts have been made to accelerate wideband channelization using FPGAs and GPUs, a direct comparison between these devices is lacking. Our work helps in picking the right device for this particular task by comparing several metrics for a range of problem sizes.

VII. Conclusions and Future Work

Our work seeks to provide an objective comparison between FPGAs and GPUs with respect to performance, energy efficiency, and performance per dollar in the context of performing polyphase filter bank (PFB) channelization. Towards this goal, optimized implementations of the PFB channelizer on FPGAs and GPUs are presented. The channelizer is evaluated for a range of problem sizes that are commonly used in military applications. Results show that the FPGAs outperform GPUs by more than 11-fold and provide an energy efficiency that is more than 7.8 times better than GPUs. Results obtained from multiple implementations on the GPU with varying precision reveal that the GPU architecture's fixed-width data pipeline as the major reason for this difference. We also observe that while FPGAs exhibit better raw performance and performance per watt, the GPUs show a better performance per dollar.

VIII. Acknowledgement

This work was supported in part by NSF I/UCRC IIP-1266245 via the NSF Center for High-Performance Reconfigurable Computing (CHREC).

References

[1] G. Savir, "Scalable and Reconfigurable Digital Front-End for SDR Wideband Channelizer," Master's thesis, Delft Univ. of Tech., 2006.

[2] T. Ulversøy, "Software Defined Radio: Challenges and Opportunities," *IEEE Communications Surveys and Tutorials*, vol. 12, no. 4, pp. 531–550, Oct. 2010.

[3] M. Woh, S. Seo, H. Lee, Y. Lin, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner, "The Next Generation Challenge for Software Defined Radio," in *SAMOS '07, Lecture Notes in Computer Science*. Springer, 2007, vol. 4599, pp. 343–354.

[4] K. C. Zangi and R. D. Koilpillai, "Software Radio Issues in Cellular Base Stations," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 4, pp. 561 –573, Apr 1999.

[5] M. Woh, S. Mahlke, T. Mudge, and C. Chakrabarti, "Mobile Supercomputers for the Next-Generation Cell Phone," *Computer*, vol. 43, no. 1, pp. 81–85, 2010.

[6] Y. Wang, H. Mahmoodi, L.-Y. Chiou, H. Choo, J. Park, W. Jeong, and K. Roy, "Energy-efficient hardware architecture and vlsi implementation of a polyphase channelizer with applications to subband adaptive filtering," *J. Signal Processing Systems*, vol. 58, no. 2, pp. 125–137, 2010.

[7] B. K. Hamilton, "Implementation and Performance Evaluation of Polyphase Filter Banks on the Cell Broadband Engine Architecture," Undergraduate Thesis, University of Cape Town, Oct 2007.

[8] S. Kim, W. Plishker, and S. Bhattacharyya, "An efficient gpu implementation of an arbitrary resampling polyphase channelizer," in *2013 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, Oct 2013, pp. 231–238.

[9] V. Adhinarayanan and W.-c. Feng, "Wideband channelization for software-defined radio via mobile graphics processors," in *19th IEEE Int'l Conference on Parallel and Distributed Systems (ICPADS)*, Seoul, Korea, December 2013.

[10] L. Pucker, "Channelization Techniques for Software Defined Radio," in *Proc. of SDR Forum Conference*, Jun 2003, pp. 1–6.

[11] M. Kowarschik and C. Wei, "An overview of cache optimization techniques and cache-aware numerical algorithms," in *Algorithms for Memory Hierarchies*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2003, vol. 2625, pp. 213–232.

[12] S. Rodriguez and B. Jacob, "Energy/power breakdown of pipelined nanometer caches (90nm/65nm/45nm/32nm)," in *Proceedings of the 2006 International Symposium on Low Power Electronics and Design (ISLPED)*, 2006, pp. 25–30.

[13] AMD APP Profiler, accessed July 2013. [Online]. Available: http://developer.amd.com/tools-and-sdks/heterogeneous-computing/amd-app-profiler/

[14] J. Fowers, G. Brown, J. Wernsing, and G. Stitt, "A performance and energy comparison of convolution on gpus, fpgas, and multicore processors," *ACM Trans. Archit. Code Optim.*, vol. 9, no. 4, pp. 25:1–25:21, Jan. 2013.

[15] B. E. Nelson, M. J. Wirthlin, B. L. Hutchings, P. M. Athanas, and S. Bohner, "Design productivity for configurable computing," in *ERSA*, vol. 8, 2008, pp. 57–66.

[16] D. Thomas, J. Coutinho, and W. Luk, "Reconfigurable computing: Productivity and performance," in *2009 Conference Record of the Forty-Third Asilomar Conference on Signals, Systems and Computers*, Nov 2009, pp. 685–689.

[17] W. Song, M. Vai, H. Nguyen, and A. Horst, "High-performance low-power polyphase channelizer chip-set," in *Conference Record of the Thirty-Fourth Asilomar Conference on Signals, Systems and Computers*, vol. 2, Oct 2000, pp. 1691–1694 vol.2.

[18] S. A. Fahmy and L. Doyle, "Reconfigurable Polyphase Filter Bank Architecture for Spectrum Sensing," in *Proc. of the 6th Int'l Conference on Reconfigurable Computing: Architectures, Tools and Applications (ARC)*. Berlin, Heidelberg: Springer-Verlag, Mar 2010, pp. 343–350.

[19] R. M. Monroe and R. Jarnot, "Broad-Bandwidth FPGA-Based Digital Polyphase Spectrometer," Tech Brief, Jet Propulsion Laboratory, Aug 2011.

[20] M. Awan, F. Harris, and P. Koch, "Time and power optimizations in fpga-based architectures for polyphase channelizers," in *Conference Record of Forty-Fifth Asilomar Conference on Signals, Systems and Computers*, Nov 2011, pp. 914–918.

[21] Y. Lin, H. Lee, M. Woh, Y. Harel, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner, "SODA: A Low-power Architecture For Software Radio," in *Proc. of the 33rd Annual Int'l Symposium on Computer Architecture (ISCA)*. Los Alamitos, CA, USA: IEEE Computer Society, 2006, pp. 89–101.

[22] Y. Lin, H. Lee, M. Woh, Y. Harel, S. Mahlke, T. Mudge, , C. Chakrabarti, and K. Flautner, "SODA: A High-Performance DSP Architecture for Software-Defined Radio," *IEEE Micro*, vol. 27, no. 1, pp. 114–123, Jan. 2007.

[23] M. Woh, S. Seo, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner, "AnySP: Anytime Anywhere Anyway Signal Processing," in *Proc. of the 36th Annual Int'l Symposium on Computer Architecture (ISCA)*. ACM, 2009, pp. 128–139.

[24] K. van der Veldt, R. van Nieuwpoort, A. L. Varbanescu, and C. Jesshope, "A Polyphase Filter for GPUs and Multi-Core Processors," in *Proc. of the 2012 Workshop on High-Performance Computing for Astronomy Date (Astro-HPC)*, Jun 2012, pp. 33–40.