FPGA-based Acceleration of FT Convolution for Pulsar Search Using OpenCL

HAOMIAO WANG, University of Auckland, New Zealand
PRABU THIAGARAJ, University of Manchester, UK
OLIVER SINNEN, University of Auckland, New Zealand

The Square Kilometre Array (SKA) project will be the world's largest radio telescope array. With its large number of antennas, the number of signals that need to be processed is dramatic. One important element of the SKA's Central Signal Processor package is pulsar search. This article focuses on the FPGA-based acceleration of the Frequency-Domain Acceleration Search module, which is a part of SKA pulsar search engine. In this module, the frequency-domain input signals have to be processed by 85 Finite Impulse response (FIR) filters within a short period of limitation and for thousands of input arrays. Because of the large scale of the input length and FIR filter size, even high-end FPGA devices cannot parallelise the task completely. We start by investigating both time-domain FIR filter (TDFIR) and frequency-domain FIR filter (FDFIR) to tackle this task. We applied the overlap-add algorithm to split the coefficient array of TDFIR and the overlap-save algorithm to split the input signals of FDFIR. To achieve fast prototyping design, we employed OpenCL, which is a high-level FPGA development technique. The performance and power consumption are evaluated using multiple FPGA devices simultaneously and compared with GPU results, which is achieved by porting FPGA-based OpenCL kernels. The experimental evaluation shows that the FDFIR solution is very competitive in terms of performance, with a clear energy consumption advantage over the GPU solution.

CCS Concepts: • **Computer systems organization → Multiple instruction, multiple data**; • **Hardware → Hardware-software codesign**; • **Applied computing** → *Astronomy*;

Additional Key Words and Phrases: SKA, OpenCL, FIR filter, high-level design

1 INTRODUCTION

Pulsar (Pulsating Radio Source), as a highly magnetized rotating neutron star, is an ideal research object for physics and astrophysics research. It has been used in a wide range of areas, such as tests of general relativity, galactic studies, and cosmology [7]. Unlike other visible astronomical

objects, electromagnetic radiation beams emitted from pulsars are hard for the optical telescope to detect unless the beam is directed toward the telescope and there are no obstacles between them.

Hence, employing radio telescopes becomes the main approach to observe pulsars, and, in practice, most known pulsars have been recorded by radio telescopes, including the first detected pulsar. The pulsar signals are weak radio sources and some are even weaker than thermal noise. So the physical scale of a radio telescope has to be large, and the integration time of a specific space area has to be long enough. For basic pulsar search, the input signals are Fourier transformed and analysed in the frequency domain, due to the regularity of pulsar beams. Since some pulsar beams might be scattered by clumpy interstellar medium, a group of dispersion measure (DM) trails of one input array has to be tested.

If the period of a pulsar is not a constant during the integration period, such as in binary pulsars (which science is most interested in), then the Doppler effect makes it more difficult to be observed. For such pulsars, acceleration search is applied by assuming a group of accelerations of a pulsar. This article focuses on the Fourier Domain Acceleration Search (FDAS). It is an effective approach to remove (sweep up) the smearing of signals by using the correlation technique [24, 32].

The Square Kilometer Array (SKA)[1] will be the world's largest radio telescope array and is currently in its phase one (SKA1) pre-construction. Based on the covered frequency bandwidth, it is divided into LOW, MID, and SURVEY. Because of the huge size and number of antennas, the workload of digital signal processing becomes the main challenge for the hardware development [15]. The Central Signal Processor (CSP) package of the SKA1-MID contains many sub-elements, and the pulsar search engine (PSS) is one of them. The PSS searches for pulsars over a range of dispersion measure, acceleration, and period search space using various approaches. The FDAS module, which is a compute-intensive application, is an essential part of the SKA1-MID PSS. The core computation part of the FDAS module is to convolve with a large number of input signals with a group of lengthy templates. Because of the need for the precision, the input signals and coefficients arrays are complex single-precision floating-point data. The large amount of complex floating-point operations and restrict time limitation are big problems for processors. Even for the powerful high-performance computing systems, the overall workload is very challenging, especially its very high power consumption, considering the remote locations of the telescopes.

It is essential for the SKA project to employ high-performance computing devices in accelerating the compute-intensive signal processing tasks. For such a large-scale project, it is necessary to consider all alternatives before deployment. To evaluate the performance regarding execution latency and energy dissipation for different hardware platforms, employing the Field-programmable gate array (FPGA) for prototype design is necessary. However, the traditional hardware development flow using Hardware Description Languages (HDLs) such as VHDL or Verilog is a large barrier to proper software engineering. It excludes non-hardware experts from participating and following the development. Solutions often become very device specific. This is especially problematic in the SKA project, where the final hardware architecture and devices (e.g., FPGA versus GPU) are not finalized yet. As an international research project, many teams are working worldwide on different aspects, and high-level approaches can strongly aid the interaction between different research teams, especially for prototype designs.

In this article, we are investigating the use of FPGAs for the efficient high-performance computing of the core computation part of the FDAS module–FT convolution. This not only produces low-power processing solutions for the demanding pulsar search modules but also evaluates the viability of using high-level approaches to achieve the needed efficiency and performance and its ability to support the sweeping of a large design space. The main contributions are as follows:

---

[1]www.skatelescope.org.

- Investigation and proposal of various differing designs for FT convolution. In contrast to previous work the designs are tailored to the demanding nature of the underlying FIR filters: large filters (i.e., a large number of coefficients), multiple filters for same input data, long input data stream, complex floating point values, and demanding real-time limit.
- Exploration of the design space in several directions: convolution methods (time versus frequency-based), area versus time efficient designs, optimising for single filter versus multiple filters, FFT parameters, single and multiple FPGAs, and so on.
- Implementation of designs using high-level approach OpenCL and analysis of achieved performance relative to upper bounds (based on available resources).
- Extensive experimental evaluation of all designs on the real host system with up to three FPGA boards; porting of OpenCL implementation to GPU; performance comparisons regarding speed (i.e., execution latency) and energy consumption.

This article studies the FPGA-based acceleration of the FT convolution part of the FDAS module using high-level synthesis approach, and it is organized as follows. In Section 2, the one-dimensional convolution that is the key element of the FT convolution and hardware acceleration in the radio astronomy area are discussed. We introduce the basic FDAS module and time-domain and frequency-domain-based algorithms to handle its FT (Fourier Transform) convolution part in Section 3. The FPGA-based FPGA-based FIR filter structure and optimisation for FT convolution module are proposed in Section 4. The FPGA-based OpenCL development technique is mentioned in Section 5, alongside a discussion of the portability of the FPGA-based kernels to other platforms. In Section 6, the performance of a group of FIR filter implementations are evaluated, and the fastest design is used to compared with GPU-based kernels. The conclusions are drawn in Section 7.

2 RELATED WORK

2.1 One-dimensional Convolution

In the FT convolution module, the compute-intensive part is a large number of lengthy FIR filters, whose essence is one-dimensional (1D) convolution. The 1D convolution and FPGA-based acceleration of 1D convolution have been well researched.

Both Intel and Xilinx (time domain transpose direct form) provide FIR compiler. For the non-symmetry coefficient array, they employ the transpose multiply-accumulate architecture that implements the FIR filter in time-domain. The SPIRAL project provides a multiplierless FIR/IIR generator that uses only additions/subtractions and shifts instead of multiplications. However, the data types of input signals and coefficient array have to be fixed-point and it supports maximum 10-tap FIR filter (as opposed to several hundreds in this work).

A thorough investigation of 1D convolution across different platforms is done in Reference [18]. The 1D convolution is implemented in both time-domain using the overlap-save algorithm and frequency-domain using the overlap-save algorithm. The evaluation showed that when the template size is several hundred, the standalone frequency-domain FPGA performs faster than GPUs and multicore processors. The data types of the input signals and coefficient array are single-precision floating points. Regarding the optimisation of a group of 1D convolutions with complex single-precision floating-point operations, we believe that it has not been addressed in literature before.

2.2 Accelerator in Radio Astronomy

High-end FPGAs, as accelerators, are widely employed in accelerating large-scale computation such as Microsoft's hyperscale datacenters [31] and IBM's Supervessel Cloud [10]. In many radio astronomy projects, FPGA accelerators are employed to handle large-scale computation as well.

In Reference [14], hundreds of Xilinx Virtex-4 FPGAs are used to implement the correlator of the SKAMP project. In LOFAR [36, 37], multi-core CPUs and many-core architectures are evaluated to implement the correlator, however, the power consumption is a problem. The Berkeley CASPER group, MeerKAT, and NRAO released the FPGA-based acceleration hardware for implementing the FX correlator of a radio telescope array [29]. In Reference [33], FPGA platforms are employed to handle digital channelised receivers. On the GPU side, the NVIDIA GTX480 GPU-based cross-correlation implementation for radio astronomy is evaluated in Reference [11]. Although each device could achieve over 1TFLOPS performance, the thermal design power (TDP) of each device is over 250W.

2.3 High-level Synthesis Approaches

To reduce the time-to-market and increase the portability of source code for FPGAs, a number of research works regarding high-level development techniques have been undertaken [16]. Known examples are LegUp [5, 6], ROCCC [27], and Nimble [26], all of them can compile C-based code and generate bitstream files to program FPGAs. LegUp is an open source project that is available for researchers to use. In terms of commercial tools, Maxeler provides a compiler for its FPGA products that are based on both Java and C/C++. LabVIEW [4, 25] provides a graphical programming environment to develop for its Xilinx FPGA-based devices.

Besides these, the two primary FPGA vendors, Intel and Xilinx, provide high-level development tools as well. The Vivado HLS, which is based on AutoESL [40], is widely adopted for high-level Xilinx FPGA development, targeting C, C++ and System C [34]. Another high-level development environment is Xilinx's SDAccel [39], which is designed for OpenCL applications targeting Xilinx FPGA-based accelerator cards [17, 21]. Intel released a development tool called Altera SDK for OpenCL [8, 9, 12, 13], which is based on OpenCL standard version 1.0. This OpenCL approach seems very promising as it is not only supported by the major FPGA vendors but also a major technology used for the programming of GPUs, with the corresponding programming environments and experience in the community. In this article, we, therefore, employ OpenCL for the development of relatively simple signal processing tasks on FPGAs. We want to explore the advantage of a high-level approach to cover a large design space, by testing many different approaches. It will be interesting to see whether OpenCL can exploit the FPGA resources efficiently for this task. We are encouraged by the successful use of the high-level development technique in many research areas regarding hardware acceleration, such as high-speed data compression [1], Map/Reduce, and computationally demanding control algorithms [28].

3 FDAS MODULE AND FT CONVOLUTION

In the SKA1-MID CSP element, over 2,000 beams are formed at 4,096 channels per beam, and the signals of each beam are processed independently, as depicted in Figure 1. Hence, each beam needs a dedicated pulsar search engine (PSS). Because the dispersion measure (DM) is unknown (we are looking for pulsars at unknown locations), about 6,000 trial values are tested, and several pulsar search approaches are employed for each trial value. These approaches include single pulse search module, time domain acceleration search module, and frequency domain acceleration search (FDAS) that we are investigating.

The FDAS module consists of two main sub-modules: the FT convolution module and the harmonic summing module. In the FT convolution module, 85 templates with different lengths are applied in each trial. The 2^{22} input points, which data type is complex single precision floating-point (SPF), are accumulated in an integration time of 536.87s. With 6,000 DM trials to perform until the next input set is ready, the time limitation t_{limit} for processing each DM trial is 89.5ms (536.87/6,000). In this 89.5ms, the main computing task is to convolve the 2^{22} complex SPF points
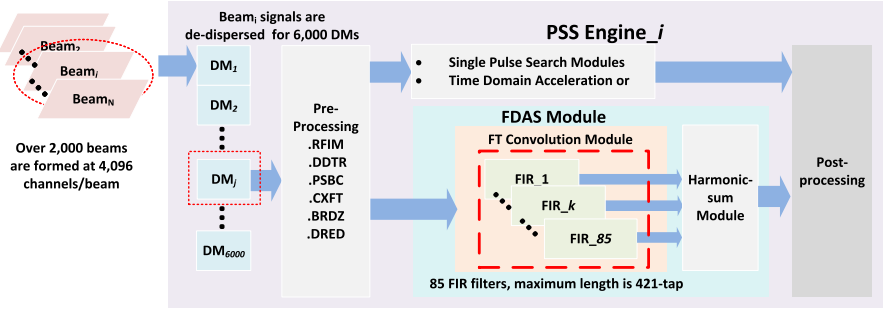
Fig. 1. Data flow of SKA1-MID CSP PSS engine.

with 84 templates and calculate the power of each complex output points. Each template can be seen as an FIR filter, whose input signals, coefficients, and output points are all complex SPF points. The output points from all 84 FIR filters plus the input signals are combined into a filter-output-plane (FOP, 85×4M complex points) and the spectral power of each complex point is sent to the harmonic-summing module for candidate detection. The process that described above is illustrated Figure 1. In these 84 FIR filters, the lengths of them are different, and the longest FIR filter has 421 taps. In case of the uncertainty of the FIR filter length, we investigate the implementation of 84 421-tap FIR filters in this research.

The real-time compute-intensive task of the FT convolution module described above is a large challenge for efficient computation, which essentially makes the use of acceleration hardware necessary. Based on the specifications of the FT convolution module, the performance needed of the 84 FIR filters is

$$\frac{8NKM}{t_{limit}} = 13.26\text{TFLOPS},$$

where $N = 2^{22}$ is the input size, $K = 421$ is the length of each FIR filter, $M = 84$ is the number of FIR filters, and eight is derived that one complex multiplication needs eight operations (four multiplications and four additions). The 13.26TFLOPS per beam is based on the straightforward implementation of 1D convolution. Since there are over 2,000 beams, the overall needed performance for one pulsar search module is over 26.5PFLOPS. In Reference [38], relaxation of requirements was studied to ease the requirements, such as changes to the input data type, size, number of filters, and so on. From this early work, it was clear that very efficient implementations of the filtering task need to be investigated. In this section and the remainder of this article, we, therefore, investigate different algorithms for FPGA-based acceleration of the FT convolution of the FDAS module. Based on the processing domain of an FIR filter, this is divided into time-domain and frequency-domain.

3.1 Time-domain FIR Filter (TDFIR)

3.1.1 Naïve TDFIR. Based on the discrete-time convolution, an K-tap FIR filter can be represented as

$$y[i] = \sum_{k=0}^{K-1} x[i-k]h[k], \ for \ i = 0, 1, \ldots N-1, \tag{1}$$

where $x[\cdot]$, $h[\cdot]$, and $y[\cdot]$ are complex SPF input signals, coefficients, and output results, respectively, and N is the input size [35]. In an FPGA implementation, the SPF multipliers are instantiated by DSP blocks and logic resources. If there are enough resources on an FPGA, then the K multiplications and additions in Equation (1) can be parallelised in a pipeline completely to achieve high-performance.
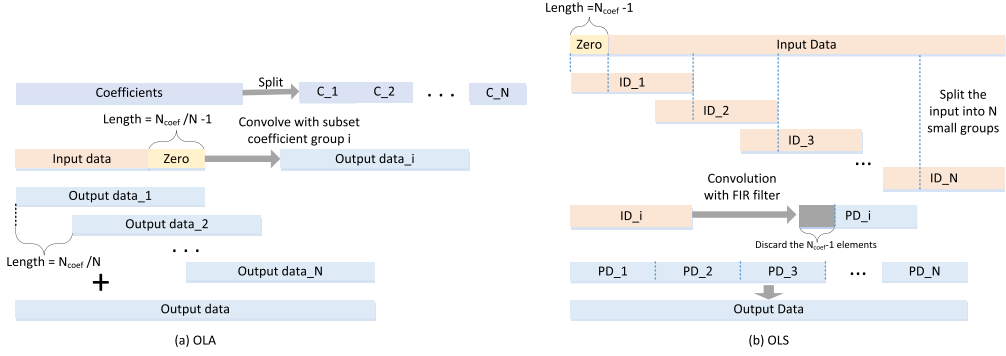
Fig. 2. Process of the OLA and OLS algorithms.

3.1.2 Overlap-add Algorithm-based TDFIR. The amount of logic resource and DSP blocks in a specific FPGA are fixed. If the FIR filter size K is too large, then an FPGA might not have enough logic resources and DSP blocks to parallelise K complex multiplications and then fails to achieve a pipeline structure. To make an FIR filter fit into the targeted FPGA and maintain high-performance, we apply the overlap-add algorithm (**OLA**) to split the coefficient array into a group of sub-arrays [30].

ALGORITHM 1: Overlap-add Algorithm

$(x, h) \leftarrow$(padded input data, coefficients)
$h \rightarrow h_1, h_2, \ldots, h_R$ {split h into a group of R disjoint sub-arrays evenly}
$y \leftarrow 0$ {create the output y and fill it with zeros}
for $i = 1$ *to R* **do**
  $\quad$ $y_i \leftarrow$convolve(x, h_i) {general convolutionin time-domain}
  $\quad$ $y \leftarrow y+$shift(y_i) {add y_i to y}
end
Output$\leftarrow y$

Algorithm 1 and Figure 2(a) outline the OLA process of splitting an FIR filter into R small sub-FIR filters. The coefficient array is evenly decomposed into R disjoint sub-arrays. If filter size K as $K = R \times K'$, where R and K' are integers, then $K' - 1$ zero points will be padded at the end of the input array in the OLA algorithm. Each output array has to be shifted by K' points and then added to the previous output array.

3.2 Frequency-domain FIR Filter (FDFIR)

3.2.1 Naïve FDFIR. Based on the convolution theorem, Equation (2), the output of an FIR filter can be obtained by the following steps [35]: Fourier transform of the input array and coefficient array, element-wise multiplication of these two arrays, and inverse Fourier transform of the output array.

$$x * h = \mathcal{F}^{-1}\{\mathcal{F}\{x\} \cdot \mathcal{F}\{h\}\}, \tag{2}$$

where $\mathcal{F}\{\cdot\}$ and $\mathcal{F}^{-1}\{\cdot\}$ are Fourier transform and inverse Fourier transform. For FPGA implementations, a fast Fourier transform (FFT) engine will be instantiated to handle Fourier and inverse Fourier transform.

3.2.2 Overlap-save Algorithm-based FDFIR. For Fourier transforming large size input, such as the targeted 4M points (2^{22}) FFT, the on-chip memory of an FPGA is unable to store all points, which makes it impossible to perform the complete process as described in Section 3.2.1 in one

Table 1. Workload of TDFIR and FDFIR

Domain	Algorithm	Workload (Single filter)	Average Workload (M filters)
TD	Naive	$8KN$	$8KN$
	OLA	$8K'N \left\lceil \frac{K}{K'} \right\rceil$	$8K'N \left\lceil \frac{K}{K'} \right\rceil$
FD	Naive	$N(6 + 2C \cdot log_2 N)$	$N(6 + C \cdot log_2 N)$
	OLS	$\left\lceil \frac{N}{N_{FT}-K} \right\rceil N_{FT}(6 + 2C \cdot log_2 N_{FT})$	$\left\lceil \frac{N}{N_{FT}-K} \right\rceil N_{FT}(6 + C \cdot log_2 N_{FT})$

go. Hence, we apply the overlap-save algorithm (OLS) to split the input signals into chunks [30]. Each chunk overlaps with its two neighbour chunks, and the extent of the overlap is $K - 1$, where K is the FIR filter length. For the first input chunk, $K - 1$ zero points have to be padded at the beginning. After convolving in frequency-domain, the overlap, which is the first $K - 1$ points of each chunk, are discarded. The OLS algorithm in Algorithm 2 and Figure 2(b) illustrate the process of splitting the input array of size N into S sub-arrays [30] of size N/S.

ALGORITHM 2: Overlap-save Algorithm

$(x, h) \leftarrow$ (padded input data, coefficients)
$x \rightarrow x_1, x_2, \ldots, x_S$ {split x into groupof sub-arrays; successive sub-arrays overlap by $K - 1$ points}
$y \leftarrow 0$ {initialize output y with zeros}
$\mathscr{F}\{h\} \leftarrow$ FFT(h) {Fourier transform of coefficient array}
for $i = 1$ *to* S **do**
  $\mathscr{F}\{x_i\} \leftarrow$ FFT(x_i) {Fourier transform of input sub-array x_i}
  $\mathscr{F}\{y_i\} \leftarrow$ times$(\mathscr{F}\{x_i\}, \mathscr{F}\{h\})$ {element-wise multiplication of Fourier transformed input sub-array and coefficient array}
  $y_i \leftarrow$ IFFT$(\mathscr{F}\{y_i\})$ {inverse Fourier transform}
  $y_i \leftarrow$ discard_overlap (y_i) {discard the front $K - 1$ points}
  $y \leftarrow$ shift(y_i) {concatenate y_i to y}
end
Output$\leftarrow y$

3.3 Workload Comparison

The main advantage of FDFIR is that its workload growth is slower than that of TDFIR as the FIR filter length increases. Assuming that the input length N is several magnitudes larger than the FIR filter length K, which means $N \gg K$ and $N + K - 1 \approx N$. This holds for our FDAS filtering task. Table 1 compares the workload of the four different approaches discussed in the previous sections. In the FDFIR, the complexity of computing the FFT is $O(Nlog_2 N)$. For single complex multiplication, six general operations are needed (four multiplications and two additions), and two additional additions for the accumulation are used for summing in case of TDFIR.

In Table 1, K' denotes the tap length that an FPGA can parallelise completely (also the sub-FIR filter length in Algorithm 1), N_{FT} is the Fourier transform length (also the chunk length in Algorithm 2), and C is a constant depending on the applied FFT algorithm, which is typically less than 5. For the FD algorithms, the one-off workload cost for the Fourier transform of the coefficient array is not included, as it is negligible with the assumption $N \gg K$. The table compares the workload for a single filter and the average workload for M filters ($M \gg 1$), which is relevant to the FT convolution module. The essential difference is that the forward Fourier transform only needs to be performed once for all M filters.

For the OLA-TD, if K can be divided by K', the workload equals to the Naïve-TD workload. For FD algorithms, the workload of Naïve-FD is not affected by FIR filter length K, but the workload

of the OLS-FD will rise with the filter size K. When K is fixed, the smaller the N_{FT}, the larger the overall workload. However, if the N_{FT} is too large, the performance might drop because of the on-chip memory size and the off-chip access efficiency. Hence, finding the suitable N_{FT} for a specific FPGA device is investigated in the evaluation section.

Based on the theoretical workload, FD algorithms have a clear advantage over TD algorithms in implementing multiple FIR filters. The evaluation will show if this advantage can be achieved in practice.

4 FPGA-BASED FIR FILTER STRUCTURE

In this section, we discuss the structures of the two non-naïve FIR filters introduced in Sections 3.1 and 3.2: the OLA algorithm-based TDFIR, referred to as **OLA-TD**, and the OLS algorithm-based FDFIR, referred to as **OLS-FD**.

4.1 OLA-TD

To handle large size FIR filters, we investigate here the structure of OLA-TD. It is based on the Naïve TDFIR, the input signals are loaded into a shift register, the core computation part is entirely parallelised using DSP blocks, and the structure can achieve loop pipelining. However, each output array of the OLA-TD is shifted and accumulated to the previous output array.

Assuming an FPGA can completely parallelise K' complex SPF multiplications, then the same input array has to be executed by OLA-TD structure R times to implement a K-tap FIR filter, where $R = \lceil \frac{K}{K'} \rceil$. For the Intel Stratix V FPGAs, one complex SPF multiplication needs four DSP blocks, so K' is decided by the number of DSP blocks on an FPGA. If there are N_{DSP} DSP blocks, then $K' = \lfloor \frac{N_{DSP}}{4} \rfloor$. Because of loop pipeline, it takes $R \times N$ clock cycles to process N points with a K-tap FIR filter.

4.2 OLS-FD

For the OLS-FD structure, two important components are Fourier transform and inverse Fourier transform. In our work, we employ a complex SPF radix-4 feedforward FFT/IFFT engine [19] provided by Intel. The single FFT engine can be configured so that it provides both FFT and IFFT. The input signals are in general order, and the output array is in bit-reversed order. The workload (number of operations) of the employed FFT engine in processing N_{FT} points is $5N_{FT}log_2N_{FT}$, so here the constant C in Table 1 is 5. It can process multiple points, referred to as N_{FT-PC}, per clock cycle, where N_{FT-PC} is a power of 2 such as 4 and 8. It takes the engine $\frac{N_{FT}}{N_{FT-PC}} - 1$ clock cycles to produce N_{FT-PC} points output for a corresponding N_{FT-PC} points input. If the output array has to be of general order, then a bit-reverse module needs to be added after the FFT engine to reorder the output array.

There are two factors that limit the FFT engine on a specific device: the bandwidth of off-chip memory and the number of DSP blocks. The bandwidth of off-chip memory limits the number of points that can be loaded and stored per clock cycle, and the amount of DSP blocks determines the number of instantiated FFT engines. Two different OLS-FD structures are proposed differing in the number of instantiated FFT engines: area-efficient OLS-FD (**AOLS-FD**) and time-efficient OLS-FD (**TOLS-FD**).

4.2.1 AOLS-FD. The AOLS-FD structure, as shown in Figure 3(a), consists of three separate parts: data fetch and multiplication, FFT/IFFT, and bit-reverse. These three parts are connected through FIFO buffers in an FPGA. The core computation part of it is the reconfigurable FFT engine. The multiplication performed in data fetch and multiplication part is the element-wise multiplication. To process one input array, the AOLS-FD structure has to be executed twice. For the first
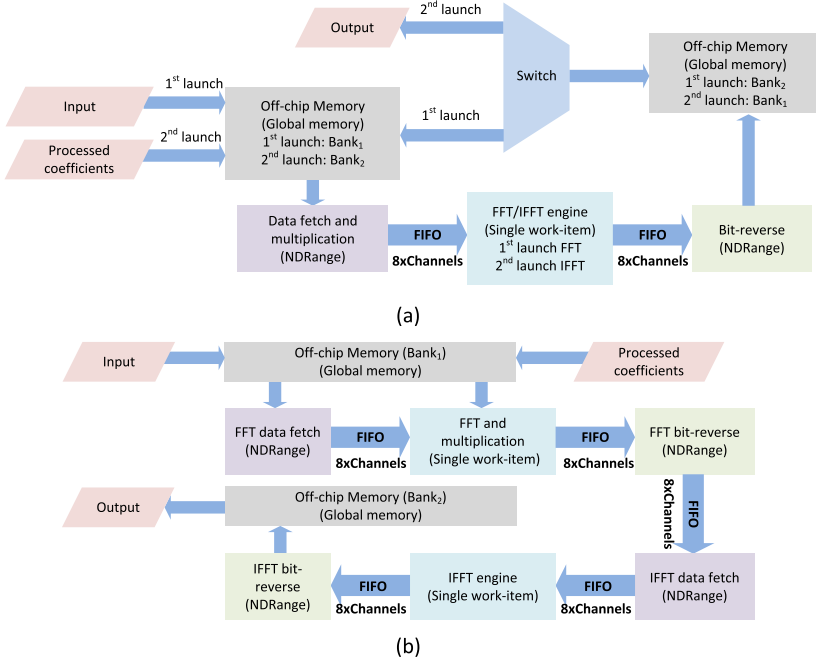
Fig. 3. Structures of (a) AOLS-FD and (b) TOLS-FD.

time, the input array and initial array are stored in one off-chip memory bank (Bank$_1$) and the Fourier transformed coefficient array is stored in another memory bank (Bank$_2$). The initial array is an array that is initialized with neutral elements $(1 + j \cdot 0)$ so that the multiplication does not affect in the first round. After the first execution, the intermediate data generated by the bit-reverse kernel are stored in Bank$_2$. In the second round, the FFT engine is configured as IFFT. The data fetch and multiplication part loads intermediate output array and pre-processed coefficient array from Bank$_2$ and the bit-reversed kernel stores the final output array in Bank$_1$. The first round is only necessary once for multiple filters, hence becomes less important with growing M, which are discussed in Section 5.2.3.

4.2.2   TOLS-FD. The TOLS-FD structure, Figure 3(b), is based on the AOLS-FD structure; however, it only needs to be executed once. TOLS-FD contains two FFT engines, one for FFT and another for IFFT. In this case, there is no need to store the intermediate results in off-chip memory, which reduces the frequency of memory usage. Different from the AOLS-FD structure, the element-wise multiplication is put after FFT engine and before bit-reverse, so the pre-processed Fourier transformed coefficient array needs to be in bit-reversed order. The input array and pre-processed coefficient array are stored in Bank$_1$ and the output array is stored in Bank$_2$.

4.3   Optimisation for FT Convolution Module

The resource usage of the acceleration device plays an important role when multiple FIR filters need to be processed as it is meaningful in parallelising the design. The off-chip memory bandwidth and the amount of DSP blocks N_{DSP} are two factors that affect the proposed structures.

In this article, we employ the Terasic DE5-Net acceleration card as the target acceleration device, and the detailed analysis is based on it. A DE5 card has one Intel Stratix V 5SGXA7 FPGA,
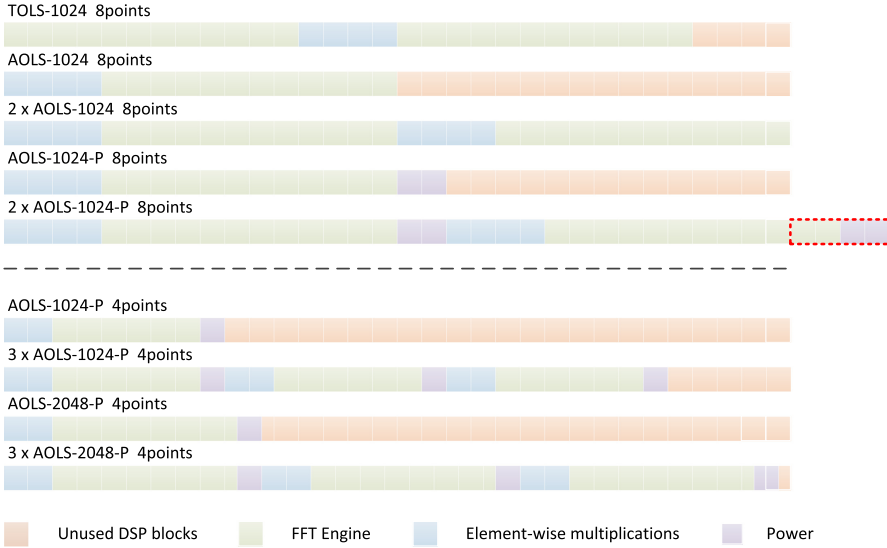
Fig. 4. DSP blocks usage of OLS-FD structures.

referred to as *A7*. The *A7* FPGA possesses 256 DSP blocks, which means 256 SPF or 64 complex SPF multiplications can be performed in parallel. The DE5 card has two DDR3 memory banks (Bank$_1$ and Bank$_2$) and each bank is connected with *A7* FPGA through a 64-bit data bus. The maximum frequency of DDR3 SDRAM supported by the DE5 card is $1,066$MHz, thus the theoretical peak data transfer rate for one memory bank is $64 \times 2 \times 1,066 \times 10^6 = 136$Gbps (2 for double data rate).

For the TDFIR, $\frac{N_{DSP}}{4}$, which is 64, is smaller than the FIR filter length K, which is 421, so only one input signal can be processed per clock cycle using the OLA-TD method. During processing, one input signal is loaded, and one result point is stored in one clock cycle, which needs 128 bits/cycle in total. Based on the theoretical transfer rate, the off-chip memory will become the barrier only when the FPGA operation frequency is higher than 1GHz, which is impossible for Stratix V FPGAs.

Regarding the FDFIR, the segment length N_{FT} and N_{FT-PC} are two important parameters that influence both TOLS-FD and AOLS-FD structures. Let us look at the DSP block usage of TOLS-N_{FT} and AOLS-N_{FT} using 8 points FFT engine on an *A7* FPGA as depicted in Figure 4. The DSP cost of the FFT engine is decided by N_{FT} and N_{FT-PC}, and the cost of element-wise multiplication part is decided by N_{FT-PC} only. We see a symbolic representation of the DSP block consumption of the different settings, distinguished by the components of the structures as FFT engine, element-wise multiplication, and so on.

For an 8-point 1,024 FFT engine, it has 96 multiplications that cost 96 DSP blocks. In Figure 4, TOLS-1024 consumes 224 ($96 \times 2 + 32$, which costs 88% of overall DSP blocks) DSP blocks and such an implementation takes a large amount of off-chip memory bandwidth, which is 1,024 bits/cycle (8 points×64-bit×2). AOLS-1024 only consumes 128 (50%) DSP blocks, and therefore it is possible to parallelise two AOLS-1024 structures on one *A7* FPGA. However, this increases the required off-chip memory bandwidth to 1,536 bits/cycle (8 points×64-bit×3). Since the theoretical peak data transfer is fixed, the increase of the required off-chip memory bandwidth leads to the decrease of FPGA operation frequency. Due to that, the performance of two AOLS-1024 (2 x AOLS-1024 using 8 points FFT engine) might not be 2x times faster than that of a single AOLS-1024 on an *A7* FPGA.

In the FT convolution module, only the spectral power values of the FIR filter (i.e., one SPF) are required, which essentially halves the output bandwidth requirement. Calculating the power of

each complex value requires simple floating-point multiplications (the square root is not necessary for this processing), which consume some more DSPs.

We name **AOLS-N_{FT}-P** to represent the AOLS-N_{FT}-based structure while calculating the power of complex value. By calculating the power, the required bandwidth of 8-point FFT engine-based structure is reduced from 1,024 bits/cycle to 768 bits/cycle. Although it is possible for an A7 FPGA to parallelise two AOLS-1024 structures, there are no more DSP blocks to calculate the power of complex value. This is indicated by the overrun on the red dot part in Figure 4. The numbers of DSP blocks used for implementing element-wise multiplication and power calculation are decided by the number of processed points per clock cycle N_{FT-PC} of the FFT engine.

To best exploit the resources on the FPGA, we reduce the points that are simultaneously processed by the FFT engine N_{FT-PC} from eight to four, and the resources usage of such kernels is illustrated in the bottom half of Figure 4. It can be seen that up to three AOLS-1024-P or AOLS-2048-P structures can be instantiated in parallel on an A7 FPGA. By employing the 4-point FFT engine, the simultaneously processed points increased from 8 (a single 8-point FFT engine) to 12 (3×4-point FFT engines). The required off-chip memory bandwidth for 4-point FFT engine-based 3x AOLS-N_{FT}-P structure is reduced from 1,024 bits/cycle (AOLS-N_{FT} using 8-point FFT engine and without the power computation) to 640 bits/cycle (4 points×64-bit+4 points×32-bit×3).

5 HIGH-LEVEL APPROACHES AND IMPLEMENTATION

For the implementation of the various FIR filtering algorithms, we have selected the high-level approach of using OpenCL. It is employed for multiple purposes, which are as follows:

- It allows for fast prototyping of the proposed implementations and to systematically explore a large design space while achieving high-performance computing;
- It provides portability to other platforms (i.e., CPUs and GPUs) and between generations of the same devices (e.g., Stratix V and Arria 10);
- It makes the developed implementation accessible to non-hardware-design experts, which is essential in such a large cross-discipline project as the SKA.

5.1 OpenCL for FPGA

The open computing language (OpenCL) is based on standard ANSI C (C99) and can be executed on heterogeneous platforms. The OpenCL platform consists of two components: host and devices [20]. In our research, one FPGA board can be seen as a compute device, and multiple acceleration cards can be installed to the host processor. OpenCL uses the concept of two types of memory, local (fast) memory and global (slower) memory. The off-chip memory of the FPGA board such as SDRAM and QDRII SRAM is instantiated as the global memory of OpenCL kernels and FPGA on-chip memory such as BRAM is used as the local memory of OpenCL kernels.

Developing for FPGA(s) using OpenCL mainly contains two parts: OpenCL kernels for the devices and software programs for the host. To compile the FPGA-targeted OpenCL kernels, a dedicated compiler is required. In our research, the Altera SDK for OpenCL (AOCL) is employed, and the AOCL offline compiler (AOC) is used. The current AOCL conforms to OpenCL specification version 1.0 and some functions of versions 1.2 and 2.0, such as clGetKernelArgInfo and pipe [23]. Furthermore, the AOCL has optimization techniques for FPGAs, including unrolling for loops, using channels to connect different kernels, and optimizing floating-point operations. The FPGA-based OpenCL kernels can be distinguished into two types: single work-item kernels, which are recommended by Altera [22], and NDRange kernel, where several work-items are processed together.

The host programs, which is written in C or C++, are responsible for the management and the remaining work, such as organizing data between the host processor and device(s), setting the arguments of OpenCL kernels, and launching kernels. In practical execution, before launching OpenCL kernels, related data arrays are transferred from the host processor to the global memory of the FPGA board through the PCI Express (PCIe) bus [12]. Multiple devices can be connected to the host through the PCIe bus. When an OpenCL kernel is launched, it loads data into the global memory of the FPGA device. Depending on the kernel's function, part of the data or intermediate results might be stored in local memory. After executing, the output array that is stored in the global memory is sent back to the host processor.

5.2 FIR Filter Kernel

Based on the discussed structures in Section 4, we investigate the implementations of these using OpenCL kernels.

5.2.1 OLA-TD. The OLA-TD kernel can be implemented as both the single work-item kernel or NDRange kernel, and the difference between these two kernel types are investigated in this article. The kernel codes are given in Figure 5, where we set $K' = 64$ (SFL). The core computation part can be completely unrolled by adding `#pragma unroll`. For the NDRange kernel, the global work size that is defined in the host program is the length of input signals, and the work-group size is specified in the device kernel (`__attriute__((reqd_work_group_size(SFL, 1, 1))))`, i.e., here 64 (SFL) in one dimension.

Although the NDRange kernel executes the same amount of complex SPF multiplication per clock cycle as the single work-item kernel, their structures are different. For the NDRange OLA-TD kernel, 64 work-items compose one work-group. By using the OpenCL barrier (`barrier()`), all the related input of one work-group have to be loaded before executing the core computation part. The FPGA executes all work-groups sequentially, and for each work-group, one work item will be executed every clock cycle.

5.2.2 OLS-FD. Based on the structures of AOLS-FD and TOLS-FD in Figure 3, they can be implemented using OpenCL kernels directly. Each function block is a kernel, and different function kernels are connected with the channels. The data fetch and multiplication kernel and bit-reverse kernel are simple that can be implemented using NDRange kernel. For the FFT engine, the single work-item kernel type is employed. To execute the same amount of input signals, AOLS-N_{FT} kernel needs to be launched twice, and TOLS-N_{FT} kernel only needs to be launched once.

Though the pipeline of TOLS-N_{FT} has more stages than AOLS-N_{FT}, they take about the same amount of clock cycles in one launch when $N \gg N_{FT}$, which is $\lceil \frac{N}{N_{FT}-K} \rceil \times \frac{N}{N_{FT}-PC}$. Since the TOLS-FD structure contains two FFT engines and they can work simultaneously, it needs fewer clock cycles than AOLS-FD in processing the same input signals. For example, if the FFT engine is set to process 8 points per clock and the operation frequency of the FPGA is a constant, then the time costs of using AOLS-FD and TOLS-FD to handle, say, 3 N_{FT} segment arrays are illustrated in Figure 6. Although TOLS-FD structure costs fewer clock cycles, the latency in practice might not be noticeable, since the operation frequency of AOLS and TOLS are not the same.

5.2.3 Multiple FIR Filters. The studied acceleration task for the FT convolution module is to implement multiple FIR filters, rather than a single FIR filter. We discussed the implementation of multiple FIR filters as the proposed structures in Sections 4.1 and 4.2. Launching an OpenCL kernel on an FPGA (i.e., the kernel has already been synthesized by AOCL, and the FPGA has been configured correspondingly) induces a certain time overhead in the order of milliseconds. The numbers of kernel launches in implementing a single and multiple FIR filters are given in Table 2.

```
#define SFL 64    //Sub-Fliter Length

__attribute__((task))
__kernel void OLA-64S (
__global float *restrict dataPtr ,   //Input
__global float *restrict filterPtr ,//Coefficients
__global float *restrict resultPtr ,//Output
const int offset ,  // 1 to num_chunks
const int num_chunks ,
const int totalInputLength ){

float i_re[SFL];
float i_im[SFL];
int ilen , k;

#pragma unroll
  for (k=0; k < SFL; k++){
    coef_real[k] = filterPtr[2*(SFL*(offset -1)+k)];
    coef_imag[k] = filterPtr[2*(SFL*(offset -1)+k)+1];
  }

// Main loop to process input signals
  for(ilen = 0; ilen < totalInputLength; ilen++){
    float r_re = resultPtr[2*(ilen + offset * SFL)];
    float r_im = resultPtr[2*(ilen + offset * SFL)+1];
#pragma unroll
    for (k=0; k < SFL-1; k++){
      i_re[k] = i_re[k+1];
      i_im[k] = i_im[k+1];
    }
    // Shift in 1 complex data point to process
    i_re[SFL-1] = dataPtr[2*ilen ];
    i_im[SFL-1] = dataPtr[2*ilen +1];

    //unroll core computation part of OLA-TD
#pragma unroll
    for (k=SFL-1; k >=0; k--){
      r_re += i_re[k] * coef_real[SFL-1-k]
              - i_im[k] * coef_imag[SFL-1-k];
      r_im += i_re[k] * coef_imag[SFL-1-k]
              + i_im[k] * coef_real[SFL-1-k];
    }
    resultPtr[2*(ilen + offset*SFL)] = r_re;
    resultPtr[2*(ilen + offset * SFL)+1] = r_im;
  }
}
```

```
#define SFL 64    //Sub-Fliter Length

__attribute__((reqd_work_group_size(SFL, 1, 1)))
__kernel void OLA-64N(
__global float *restrict dataPtr ,   //Input
__global float *restrict filterPtr ,//Coefficients
__global float *restrict resultPtr ,//Output
const int offset ,  // 1 to num_chunks
const int num_chunks ){

unsigned int i_g = get_global_id (0);
unsigned int i_l = get_local_id (0);
unsigned int load_i = i_g + (offset - 1)*SFL;

//Load input to local memory (on-chip)
__local float i_re[SFL*2];
__local float i_im[SFL*2];
i_re[i_l] = dataPtr[2*load_i];
i_im[i_l] = dataPtr[2*load_i + 1];
i_re[i_l+SFL] = dataPtr[2*(load_i+SFL)];
i_im[i_l+SFL] = dataPtr[2*(load_i+SFL) + 1];

//To make sure the local memory is loaded properly
barrier(CLK_LOCAL_MEM_FENCE);

//Load the results generated during last launch
float r_re = resultPtr[2*i_g];
float r_im = resultPtr[2*i_g + 1];
unsigned int tap = (num_chunks - offset)*SFL - 1;


#pragma unroll
for(unsigned int ilen = 0; ilen<SFL; ilen ++){
load_i = i_l + ilen;
  r_re += i_re[load_i]*filterPtr[2*(tap-ilen)]
          - i_im[load_i]*filterPtr[2*(tap-ilen)+1];
  r_im += i_re[load_i]*filterPtr[2*(tap-ilen)+1]
          + i_im[load_i]*filterPtr[2*(tap-ilen)];
}

//Save the output
resultPtr[2*i_g] = r_re;
resultPtr[2*i_g + 1] = r_im;
};
```

Fig. 5. OpenCL code (using Single work-item(Left) and NDRange(Right)) of OLA-TD kernel.

For TDFIR kernels, each launch takes about N clock cycles. For OLS-FD kernels, each launch takes about $\lceil \frac{N}{N_{FT}-K} \rceil \times \frac{N_{FT}}{N_{FT}-PC}$ clock cycles. Different from the OLA-TD kernel, the overlap-add operations of the Naïve-TD kernel need to be handled in software by the host program. For the OLA-TD kernel, its performance is restricted by the number of parallelised complex SPF multipliers, which is decided by the available number of DSP blocks, and, to a lesser extent, by the available amount of the logic resources. To process the same input array with M FIR filters, the kernel needs to be launched RM times, which is M times implementing single FIR filter. Similar to the OLA-TD kernel, the available logic resources and DSP blocks might be a problem for the performance of the TOLS-FD kernel. In implementing M FIR filters, the TOLS-FD kernel has to be launched M times. However, the same input array is Fourier transformed M times as well.

Regarding the Naive-FD kernel, it executes the 4M points Fourier transform and element-wise multiplication during each launch. The 4M FFT engine, which is provided by Intel, is based on the general FFT engine and the 4M points are treated as a $2^{11} \times 2^{11}$ matrix. The engine executes the 2K FFT on all rows and reorders the output matrix, followed by another 2K FFT on all rows to generate the Fourier transformed matrix. Different from OLA-TD and TOLS-FD kernels, the Navie-FD kernel is a generic kernel that can be configured as FFT or IFFT engine. To implement M
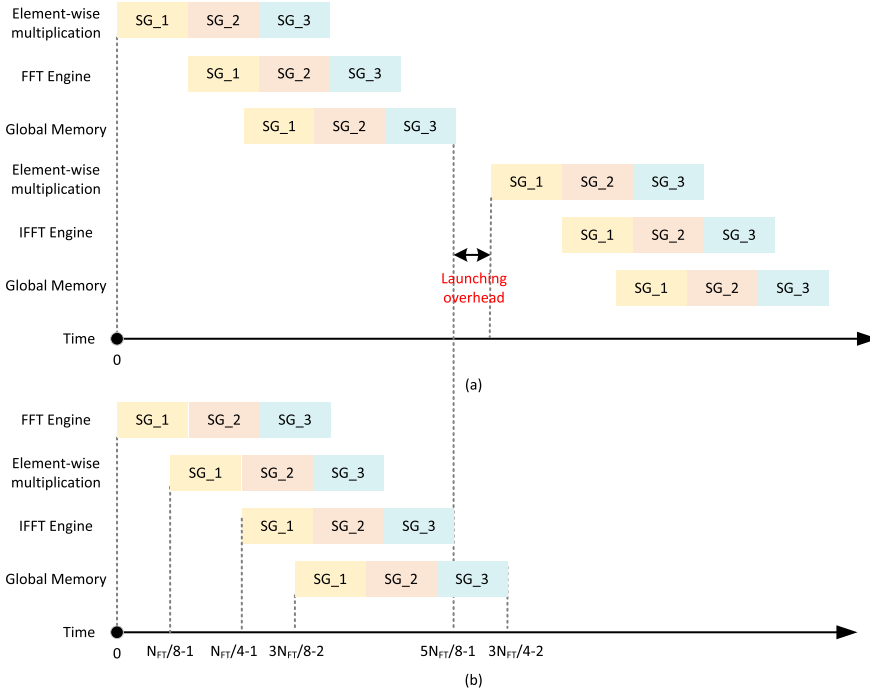
Fig. 6. Time cost of (a)AOLS-FD and (b)TOLS-FD.

Table 2. Kernel Launch Times in Implementing Single Filter
and Multiple Filters

Kernels	TD		FD		
	Naïve	OLA	Naïve	AOLS	TOLS
Single filter	R	R	2	2	1
M filters	RM	RM	$M + 1$	$M + 1$	M

FIR filters, the input array only needs to be Fourier transformed once, and the kernel needs to be launched $M + 1$ times instead of $2M$ times. The AOLS-FD kernel has the same advantage as Naive-FD, and it needs to be launched $M + 1$ times for M FIR filters as well. As the number of FIR filters M increases, the average launching times $(\frac{M+1}{M})$ of each FIR filter using Naive-FD or AOLS-FD will be halved.

5.3 Kernel Portability

OpenCL is designed for developing codes for different target platforms, however, directly using FPGA-based FIR kernels on other platforms, such as GPUs, might not achieve high-performance computing. While there is general portability regarding functionality, performance portability of FPGA-based FIR filter kernels is affected by three main factors.

5.3.1 Single Work-item vs. NDRange. The single work-item Naïve-TD and OLA-TD kernels include several optimization techniques specific to the FPGA architecture, such as unrolling of `for` loops and shift registers. For GPU-based implementation, the single work-item kernel will be executed sequentially, which is similar to a general CPU-based implementation. For example, the

performance of single work-item Naïve kernel in Reference [38] (removing FPGA-based optimization code and commands) on a mid-range AMD GPU is only 0.026GFLOPS, which is hundreds of times slower than that of on mid-range FPGA. Regarding the NDRange-based Naïve-TD and OLA-TD (Figure 5) kernels, although the for loop cannot be unrolled by GPU, it can still achieve high-performance, because hundreds to thousands of processing elements in the GPU can work in parallel.

5.3.2 Channel and Pipe. The AOCL channels are used to connect different kernels in OLS-FD kernels. Compared with OpenCL pipes, the channel is relatively simple to use, since it is unnecessary to enable the usage of channels in the host program. For porting to other platforms, it is better to use the OpenCL pipe construct, since it conforms to the OpenCL standard. However, the frequency of channel-based kernels is higher than that of pipe-based kernels on FPGA devices. Take the TOLS-FD kernel as an example, it has two FFT engines and all function modules can be connected using Intel channels or OpenCL pipes. When the N_{FT} is set as 2048, the channel-based kernel frequency is 1.1 times higher than pipe-based kernels. In our work, the connections between different kernels are all channels.

5.3.3 OpenCL Library. The employed FFT engine is dedicated for FPGAs, since it contains several FPGA-based optimization techniques and is implemented as a single work-item kernel. When the employed FPGA-based FFT engine code is used on GPU or CPU platforms, the performance will be hundreds of times slower as well. The alternative solution for GPU and CPU platforms is to use the OpenCL-based FFT library called clFFT. Even though the current AOCL supports OpenCL library technique [23], the clFFT still cannot be used in FPGAs, mainly because it uses features from OpenCL 1.2, which are not yet supported by AOCL.

6 EVALUATION

In this section, we evaluate the proposed FIR filter designs and their implementations with OpenCL. We do this on two different, but comparable types of mid-range acceleration devices, namely, FPGA and GPU. Our objective is to determine which of the designs achieves the best performance and lowest power/energy consumption on FPGAs and put that into relation to a comparable GPU.

6.1 Experimental Setup

The essential characteristics of the employed FPGA and GPU platforms, both PCIe boards, are given in Table 3. For better comparability, the process technology of the Intel Stratix V 5SGXA7 FPGA and AMD Radeon R7 370 GPU, referred to as **R7**, were chosen to be the same, which is 28nm.

For the official provided board support package (BSP) of the DE5 card, the maximum frequency of DDR3 SDRAM is 800MHz that makes the theoretical peak data transfer rate down to $64 \times 2 \times 800 = 102$Gbps. When the OpenCL kernel frequency f_{max} is smaller than 200MHz, the maximum data transfer rate is affected by f_{max}, and the maximum bandwidth is $64 \times 2 \times 4 \times f_{max} = 512 f_{max}$, where factor 4 is the quarter rate, the largest rate supported by the FPGA soft memory controller.

The clock frequency of an $A7$ FPGA-based OpenCL kernel is decided by many factors, and generally around 150–300MHz, while the clock of an $R7$ GPU goes up to 985MHz. It can be seen that the $R7$ GPUs have several advantages over $A7$ FPGAs in operation frequency, global memory frequency, and global memory bandwidth. For the FIR filter kernel, the global memory bandwidth is not a barrier, however, the operating frequency plays an important role for both FPGA and GPU.

In our evaluation, both FPGA and GPU devices are connected with the host through 8 lanes (×8) PCIe bus (Gen2.0, 4GB/s) and the operating system of the host is Ubuntu 14.04LTS. In terms of the compiler, the FPGA-based kernels are compiled by AOCL version 15.0.0.145, and GPU-based kernels are using AMD APP SDK version 3.0 [2, 3].

Table 3. Details of FPGA and GPU Platforms

Device (Board)	Terasic DE5-Net	Sapphire Nitro R7 370
Hardware	Intel Stratix V 5SGXA7	AMD Radeon R7 370
Technology	28nm	28nm
Compute resource	622,000 LEs 256 DSP blocks	1024 Stream Processors
On-chip memory size	50Mb	—
Global memory size	2×2GB DDR3	4GB GDDR5
Global memory frequency	800MHz	$5,600$MHz
Memory interface width	2×64-bit	256-bit
Max clock frequency	—	985MHz
OpenCL	1.0	1.2
Max power consumption	—	150W

Table 4. Resource Usage of FPGA-based FIR Filter Kernels

	Kernels	Logic	DSP	RAM	f_{max} (MHz)	Theoretical latency (ms)	rRMSE ($\times 10^{-7}$)
TD	TD-Naive-64S	49%	100%	15%	254.77	115.24	**0.695**
	TD-Naive-64N	51%	100%	18%	**270.05**	**108.72**	0.699
	OLA-64S	50%	100%	16%	236.01	124.40	1.94
	OLA-64N	51%	100%	20%	255.29	115.01	1.77
FD	FD-Naïve	59%	87%	90%	183.55	–	2.89
	AOLS-1024	52%	50%	46%	**222.17**	7.98	**1.69**
	AOLS-2048	54%	59%	88%	205.59	6.41	1.78
	AOLS-4096	59%	60%	72%	173.97	6.72	1.86
	TOLS-1024	83%	88%	77%	168.26	**5.27**	3.26

Regarding the measurement made in this research, we measure the execution latency from starting the FT convolution module in the host program until the acceleration devices finish processing the input points. Having that said, input and output points transfer to and from the host processor is not included in the measurements as the filtering task is a part of the signal processing pipeline of the FDAS module, and it is assumed that previous and subsequent modules are also executed on the acceleration device. Note that, the DE5 card and the graphics card that employed in this article are not the final devices for SKA1 CSP PSS deployment. Regarding the proposed structures, when the optimised approach is specified, it will be implemented using HDLs.

6.2 FPGA Resource Usage

Before evaluating the execution latency and the energy dissipation that are discussed in the next section, we focus on the FPGA resource consumption and the correctness. Table 4 lists the resource usage, maximum kernel frequency, and the theoretical latency of all proposed FIR filter kernels. The latency is calculated based on processing 2^{22} complex SPF points using a single 421-tap FIR filter. To confirm the correctness of the SPF outputs, we calculate the relative Root Mean Square Error (rRMSE) of the FPGA output points in comparison to the Matlab results, whose data type is double precision floating-point. For the TDFIR kernels, the "S" and "N" after the parallelisation factor 64 represent the **S**ingle work-item kernel and **N**DRange kernel types. For all FDFIR kernels in Table 4, the FFT engine processes 8 points per clock cycle.

Table 5. Resource Usage of Power Calculation Kernels

Kernels	Number	Logic utilization	DSP blocks	RAM blocks	f_{max} (MHz)	Theoretical latency (ms)
AOLS-1024-P	2	54%	57%	37%	216.35	4.12
AOLS-1024-P	3	71%	85%	54%	214.27	2.77
AOLS-2048-P	2	58%	66%	59%	**220.21**	3.00
AOLS-2048-P	3	68%	98%	81%	205.68	**2.14**
AOLS-4096-P	2	59%	66%	92%	182.68	4.56
AOLS-4096-P	3	72%	99%	88%	177.24	3.13

Note: P represents that output is power of complex numbers.

For the theoretical latencies, they are calculated based on the required clock cycle (based on the values in Tables 1 and 2) and kernel frequency f_{max} (in Table 4). The kernel launching overhead is unpredictable and it is not included in the theoretical latencies. Due to the substantial difference in the approaches, the performances regarding GFLOPS values of the TDFIR kernels are not comparable with those of the FDFIR kernels. For OLS-FD kernels, part of the points from each N_{FT} Fourier transformed values have to be discarded (due to the overlap). Hence, the real performance of these kernels is higher than the valid performance. Take AOLS-1024 as an example, over 40% of each 1,024 Fourier transformed points are discarded, which means the real performance is up to 1.6× higher than the valid performance in Table 4.

All TDFIR kernels are configured to the maximum size as they use 100% of the available DSP blocks, which is the same as theoretical analysis. For FDFIR kernels, the DSP blocks usage does not reach 100%, and the resource usage is balanced. The real DSP blocks usage of AOLS-N_{FT} and TOLS-N_{FT} is the same as discussed in Section 4.3. As one can expect the TOLS variant uses significantly more resources than the comparable (ALOS-1024) variant, where the DSP usage is about 1.75× higher.

It can be seen that the theoretical latency of Naïve-64N is the best among TDFIR kernels, however, it can only execute one FIR filter that is up to 64 taps long. Positive is that the flexible OLA-64N filter has very similar theoretical latency. In terms of the FDFIR kernel, TOLS-1024 performs better than other FDFIR kernels, and AOLS-2048 performs better than other AOLS-FD with different N_{FT}. However, the advantage of kernel TOLS-1024 over AOLS-FD kernels might disappear when implementing multiple FIR filters (see Section 5.2.3).

Regarding the kernels that calculate the power of complex value, we implemented six different combinations of AOLS-N_{FT}-P kernels employing a 4-point FFT engine. The details of resource usage, maximum frequency, and theoretical latency are provided in Table 5, where the theoretical latency is the average latency for a single FIR filter. Based on the theoretical latency, kernel AOLS-2048-P performs better than all kernels in Table 4, when the kernel is replicated three times. For all configurations, where the structure is replicated three times, the DSP block usages are over 80%, and especially for 3xAOLS-4096-P, most resources on an $A7$ FPGA reach exhaustion. Because of the high percentage of resource usage, the achieved kernel frequency is lower than other kernels.

6.3 Performance Comparison

In this section, we now evaluate and compare the execution time or latency of processing the entire input completely. While we provide performance values regarding GFLOPS for the TDFIR implementations, comparing this value between TDFIR and FDFIR kernels would be misleading, as the algorithms and necessary computations are significantly different, not incurring the same number of operations to perform the filtering.
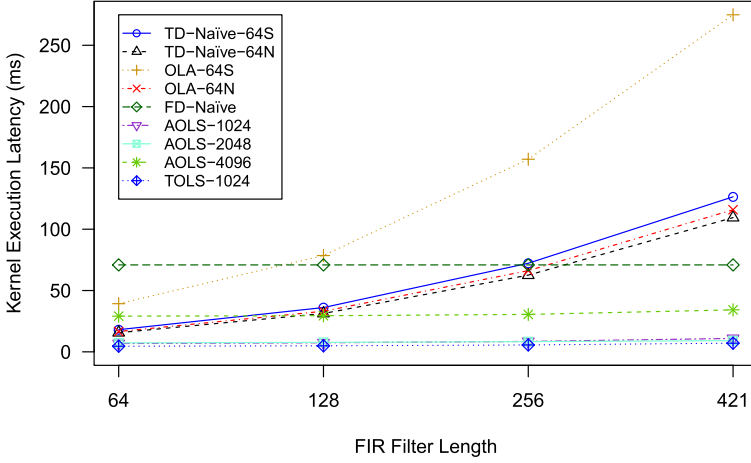
Fig. 7. Kernel execution latencies of FPGA-based FIR filter kernels.

6.3.1 TDFIR vs. FDFIR. The execution latencies of all nine kernels in Table 4 are plotted in Figure 7 over different filter lengths. All these kernels are launched to process 4M complex SPF numbers using 64-, 128-, 256-, and 421-tap FIR filters, respectively. It can be seen that TOLS-1024 is the fastest of all nine kernels and AOLS-2048 is the second when implementing a 421-tap FIR filter. For TDFIR kernels, additional operations are needed to accumulate the results by the host when using kernel Naïve-64S and kernel Naïve-64N to implement an FIR filter larger than 64 taps. With this in mind, kernel OLA-64N is the best of 4 TDFIR kernels in implementing large FIR filter. The FIR filter length does not affect the performance of FDFIR kernels too much, while the latencies of all TDFIR kernels are raised steadily as the filter size increases. In implementing a single 421-tap FIR filter, all FDFIR kernels perform better than the TDFIR kernels, so we focus on FDFIR-based kernels now.

When comparing the real results with the theoretical latencies in Table 4, the real results of NDRange-based TDFIR kernels is about the same as the theoretical latencies (1.01 × slower). Regarding the OLS-FD kernels, the actual results are over 1.3 × slower than the theoretical latencies. The main reason is that the kernel launching overhead is of the same order of magnitude with the execution latency. AOLS-2048 performs better than AOLS-1024, because the proportion of invalid points per N_{FT} points of AOLS-2048 is smaller than that of AOLS-1024. For both AOLS-2048 and AOLS-1024, the points can be streamed between the off-chip memory and the FPGA. Regarding AOLS-4096, it performs worse than AOLS-2048 and much worse than the estimated latency. The distances between addresses of necessary points to Fourier transfer 4,096 points are too large, which makes AOLS-4096 fail to achieve the streaming mode.

6.3.2 FDFIR with Spectral Power Calculation. All kernels in Table 5 are evaluated and compared with two pure FDFIR kernels (TOLS-1024 and AOLS-2048 as of Figure 7). All these kernels are employed to implement 84 different 421-tap FIR filters, and the range of input array size is from 2^{18} to 2^{22}. The average execution latency of one 421-tap FIR filter is plotted in Figure 8. It can be seen that kernel AOLS-2048 performs better than kernel TOLS-1024 in implementing multiple FIR filters, which stems largely from the fact that the forward FFT only needs to be executed once with AOLS-2048. The average latencies of four kernel configurations with appended power calculation are smaller than that of pure AOLS-2048, namely 3xAOLS-1024-P, 2xAOLS-2048-P, 3xAOLS-2048-P, and 2xAOLS-4096-P. All these four kernels not only process additional operations
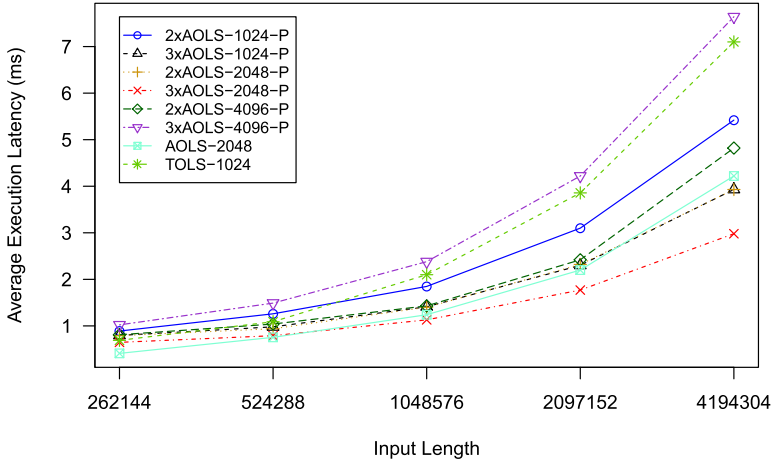
Fig. 8. Average execution latencies of two FDFIR kernels and six (multiple) kernels with appended power calculation.
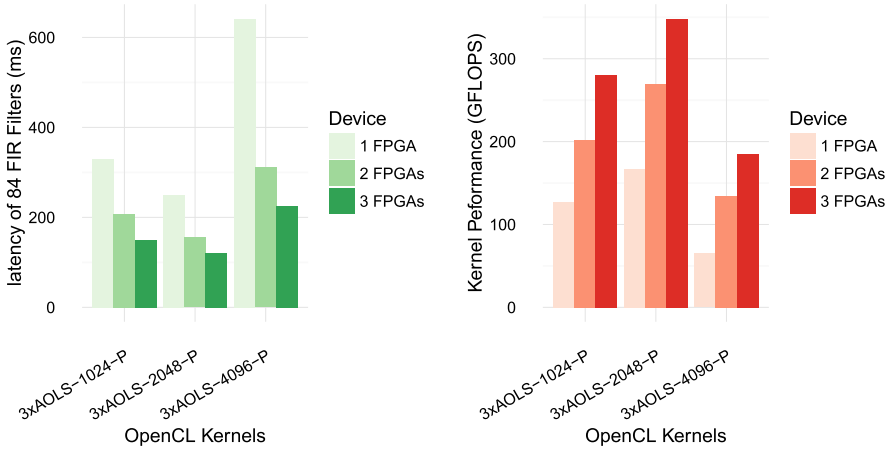

Fig. 9. Latency and performance of OpenCL kernels using multiple FPGA devices.

(calculate the power of each complex value) but also perform better than all generic FIR kernels in implementing multiple FIR filters. Kernel 3xAOLS-2048-P performs best among these eight kernels, and the average latency of processing 2^{22} points is 2.98ms, which is the same as predicted in Table 5. In total, it takes over 250ms to apply 84 FIR filters.

6.3.3 Multiple FPGAs. While intensively exploiting the FPGA resources and memory bandwidth, 250ms is still significantly larger than the time limit of the FT convolution module, which is under 100ms. Thus, we investigate using multiple FPGA devices to reduce the latency. Up to three FPGA devices (DE5-Net boards) are used in our work, and the latency and performance are plotted in Figure 9. We compare the three kernels with three instances each from the previous experiment, i.e., 3xAOLS-1024-P, 3xAOLS-2048-P, 3xAOLS-4096-P varying the number of FPGAs (boards) used in one single host system. As in the single FPGA case, 3xAOLS-2048-P performs better than the other two kernels on multiple FPGA devices. Using three FPGA devices, kernel 3xAOLS-2048-P can apply 84 FIR filters in 120ms, and the effective performance is around 350GFLOPS.
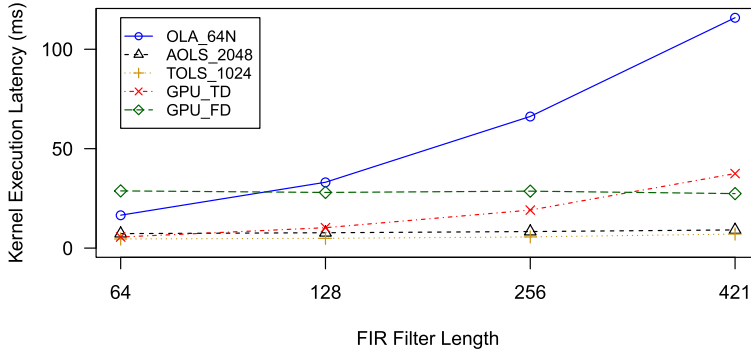
Fig. 10. Execution latencies of GPU and FPGA-based kernels in implementing a single FIR Filter.

6.3.4   FPGA vs. GPU. Now the FPGA kernels are compared to a GPU implementation. The GPU operating frequency is locked at the maximum frequency, which is 985MHz, and all 16 compute units are set to process in parallel. Since the $R7$ GPU has over 1,000 stream processors, there is no need to use OLA algorithm on it. Regarding FDFIR, it takes tens of milliseconds for $R7$ GPU to process a 4M FFT, so the OLS algorithm is employed for GPU.

The single work-item kernels are unfair for GPUs, which takes several seconds to execute, so we only compare the NDRange kernels on FPGA and GPU. Two FPGA-based kernels are evaluated on $R7$ GPU: NDRange TD-Naïve and AOLS-2048. The FPGA-based NDRange TD-Naïve kernel can be ported directly to the $R7$ GPU, referred to as **GPU-TD**. For GPU-based AOLS-2048, referred to as **GPU-FD**, the FFT engine, which is a single work-item kernel, is replaced with clFFT. The clFFT is an API designed for AMD's graphics card to perform FFT, which is well-optimised for GPU.

Now to the actual kernel performance comparison, we compared the execution latency of the fastest FPGA-based TDFIR and FDFIR kernels, which are OLA-64N, AOLS-2048, and TOLS-1024, with GPU-TD and GPU-FD. The latencies of these kernels in implementing a single 421-tap FIR filter are plotted in Figure 10 over different filter lengths.

We observe that as the FIR filter length increases, the trends of GPU-based kernels are similar to those of FPGA-based kernels. However, the performance of GPU-TD kernel is over three times higher than the fastest $A7$ FPGA-based TDFIR kernel, which is over **450GFLOPS** when implementing a 421-tap FIR filter. This is caused by the high operating frequency of GPU device. In terms of the GPU-FD kernel, the performance of it is mainly decided by the operation frequency of the GPU and not by the FIR filter length. For a single FIR filter, the GPU-FD kernel performs worse than the two FPGA-based FDFIR kernels. However, in comparison with 1D convolution results in Reference [18], the GPU-FD performs much better than the best solution of it, where the data type is SPF instead of complex SPF.

For applying multiple FIR filters, we compare the best performance we achieved using multiple FPGA devices (three FPGAs with 3xAOLS-2048-P) with a single $R7$ GPU, whose execution latencies are charted in Figure 11. Two different input sizes are evaluated, which have 2M and 4M points. Three FPGA-3xAOLS-2048-P implement 9 FIR filters during each launch. So the latency of executing a single FIR filter has no difference with that of 9 FIR filters. This is the reason for the steps that can be observed in the curves in Figure 11. The execution latency of a single GPU is about the same as that of three FPGA devices in processing both 2M and 4M points. For 4M points, the 3xAOLS-2048-P kernel on the three $A7$ FPGAs performs relatively better when the total number of FIR filters is larger than 64. Also remember that the 3xAOLS-2048-P kernel performs more computations than the GPU-FD, as the calculation of the power value of each complex point is included.
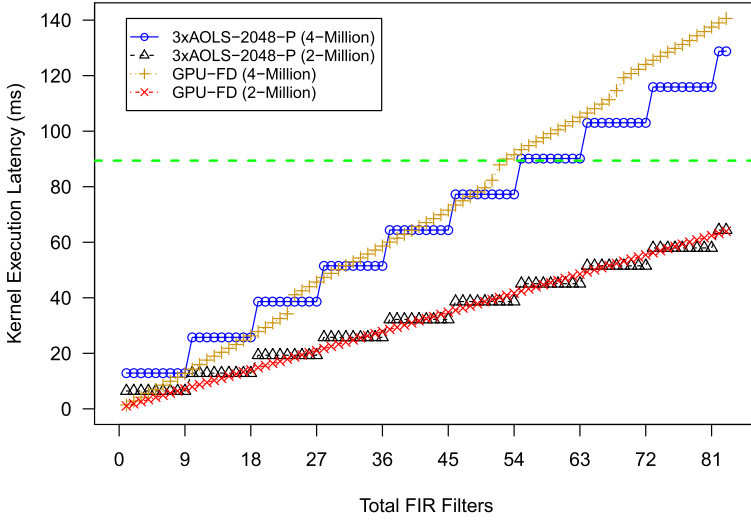
Fig. 11. Execution latencies of GPU- and FPGA-based kernels in implementing multiple FIR filters.

It should be noted that none of the considered implementations can finish processing 4M points in the specified time limitation of 89.4ms (green dot line). If the input size can be reduced to 2M points or the number of FIR filters can be reduced to 53 or less, then it is possible for a single $R7$ GPU or three $A7$ FPGAs to handle the FT convolution module for one beam.

6.4 Power and Energy Consumption

In the previous section, we saw that the execution time performance of three $A7$ FPGAs and one $R7$ GPU is very similar. It is now very interesting to compare their power consumption.

6.4.1 Power Measurment. Both FPGA and GPU devices are employed as acceleration hardware in our research. To compare the power consumption of such computation, we need to take the power consumption of both device and host into consideration.

A plug-in power meter (Efergy Ego Smart Power Socket) is used to measure the power consumption of the overall system. When the acceleration devices, both FPGA and GPU, are idle, they still need additional power, referred to as $P_{device-idle}$, especially for FPGA devices. After configuring the FPGA, the value of $P_{device-idle}$ for different kernels are different. For DE5 board, it costs 10–20W without executing any tasks and 10–15W for an $R7$ GPU board.

There are two main steps to measure the power consumption of an acceleration device-based computing. First, before installing acceleration devices, the power consumption of the basic host in the idle state is measured, referred to as $P_{host-idle}$. Then, we install the devices and launch a kernel for up to 5 minutes by usingloops till the measured power consumption in watt becomes stable. The constant power consumption of the running system is recorded as P_{total}. The real power consumption of $Kernel_i$ can be calculated as

$$P_{Kernel_i} = P_{total} - P_{host-idle}.$$

The value of P_{Kernel_i} is not only the power cost of the devices but the overall cost of using acceleration devices to process a task. It consists of two parts: the power consumption of the acceleration devices and the power consumption of the host in setting kernel arguments and launching kernels. However, the power consumption of acceleration devices is the largest of these three parts.
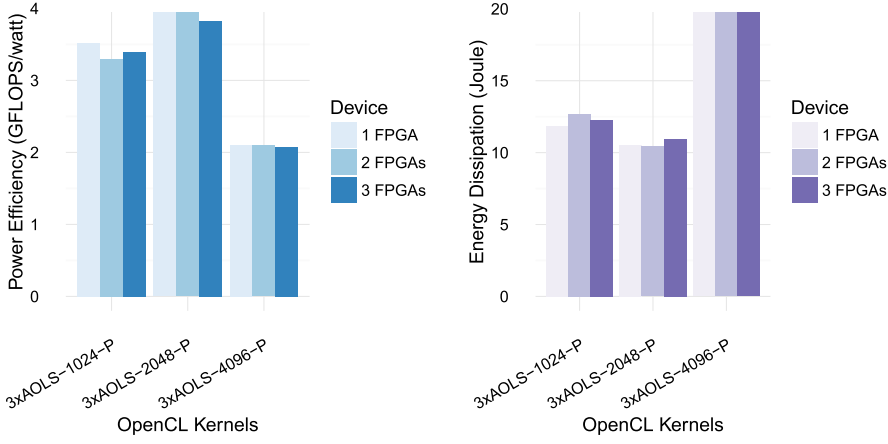
Fig. 12. Power efficiency and energy dissipation of executing multiple FIR filters.
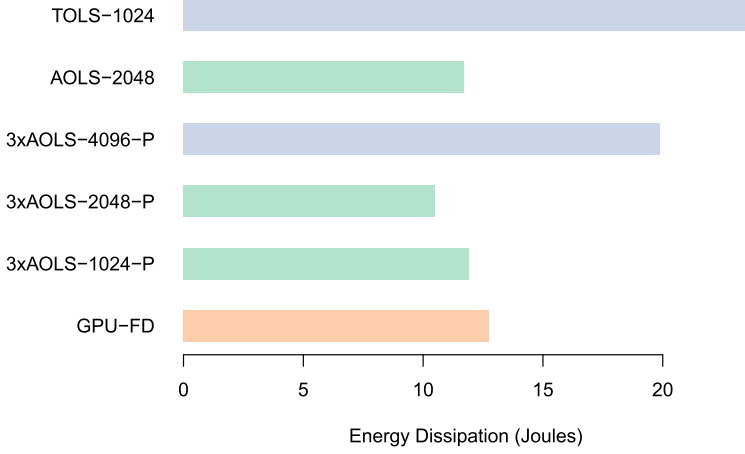

Fig. 13. Energy consumption of executing multiple FIR filters.

6.4.2 Power Comparison. The power efficiency and energy dissipation of multiple FPGA-based AOLS-N_{FT}-P kernels are depicted in Figure 12. It can be noted that the number of FPGA devices does not influence the power efficiency and energy dissipation of AOLS-N_{FT}-P kernels too much. For all three AOLS-N_{FT}-P kernels, the power efficiency remains stable as the number of FPGA devices increases. For a specified task, the energy dissipation of it is decided by the power efficiency of the kernels. When the workload of the task is fixed, the higher the value of power efficiency of a kernel, the less energy it dissipates. So, we mainly investigate the energy dissipation of single device-based kernels.

The average power consumption of running an $R7$ GPU device is 90–105W and higher than that of a single $A7$ FPGA device, which ranges from 20W to 40W. The energy dissipation of five high-performance FPGA-based kernels is compared with the GPU-based GPU-FD kernel in processing 4M points with 84 different 421-tap FIR filters, which is shown in Figure 13. The energy dissipation of kernel AOLS-2048 and 3xAOLS-1024-P is both fewer than that of kernel GPU-FD, even though the performance is worse than that of GPU-FD. Kernel 3xAOLS-2048-P has advantages over kernel

GPU-FD in both performance and energy dissipation, and the energy dissipation of it the fewest among all evaluated kernels.

The power needed for three DE5 boards-based 3xAOLS-2048-P is about the same as a single $R7$ GPU-based GPU-FD, which is $91W$. However, each DE5 board has an individual power module and cooling system. If multiple FPGAs can be integrated into one board, then the power cost might drop, and the advantages of FPGAs over GPUs would further increase. In processing the same input signals, the FPGA acceleration cards costs less energy than the GPU card, while providing similar execution performance. The extremely large-scale nature of the SKA and its longevity of many years make this an essential advantage of FPGA-based solution.

7 CONCLUSION

This article investigated the FPGA-based acceleration of the FT convolution for Pulsar Search in the SKA project using OpenCL as a high-level development technique. Because of the limitation of memory bandwidth and resources of an FPGA, the OLA and OLS algorithms were investigated to make it possible for an FPGA to implement multiple large FIR filters and process large size input signals. Different approaches to implementing TDFIR and FDFIR were designed and experimentally evaluated. The results given evidence that OpenCL can well be used to development FPGA solutions efficiently while achieving high performance for such computations. The FPGA-based FDFIR kernels perform better than TDFIR kernels for lengthy FIR filters. Even though the achieved GFLOPS are higher for TDFIR, the FDFIR computation is more efficient. We studied different designs and configurations of the proposed filters to exploit the available FPGA resources as much as possible. To evaluate the portability of FPGA-based kernels, and to put the FPGA performance into relation with GPUS, two appropriate kernels were tested on a mid-range GPU device. The experiments demonstrate that the latency of FPGA-based FDFIR kernels is smaller than that of GPU-based kernels for a single FIR filter. We also investigated the use of multiple FPGA devices and the computation of multiple filters. Three $A7$ FPGA devices perform better than single $R7$ GPU device while being more power efficient.

ACKNOWLEDGMENTS

REFERENCES

[1] Mohamed S. Abdelfattah, Andrei Hagiescu, and Deshanand Singh. 2014. Gzip on a chip: High performance lossless data compression on fpgas using opencl. In *Proceedings of the International Workshop on OpenCL*. ACM, 4.

[2] AMD. 2013. APP SDK-A Complete Development Platform. Retrieved from http://developer.amd.com/tools-and-sdks/opencl-zone/amd-accelerated-parallel-processing-app-sdk/.

[3] AMD. 2015. AMD APP SDK OpenCL Optimization Guide. Retrieved from http://developer.amd.com/wordpress/media/2013/12/AMD_OpenCL_Programming_Optimization_Guide2.pdf.

[4] Hugo A. Andrade and Scott Kovner. 1998. Software synthesis from dataflow models for G and LabVIEW/sup TM. In *Proceedings of the 32nd Asilomar Conference on Signals, Systems & Computers*, Vol. 2. IEEE, 1705–1709.

[5] Andrew Canis, Jongsok Choi, Mark Aldham, Victor Zhang, Ahmed Kammoona, Jason H. Anderson, Stephen Brown, and Tomasz Czajkowski. 2011. LegUp: High-level synthesis for FPGA-based processor/accelerator systems. In *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. ACM, 33–36.

[6] Andrew Canis, Jongsok Choi, Mark Aldham, Victor Zhang, Ahmed Kammoona, Tomasz Czajkowski, Stephen D. Brown, and Jason H. Anderson. 2013. LegUp: An open-source high-level synthesis tool for FPGA-based processor/accelerator systems. *ACM Trans. Embed. Comput. Syst.* 13, 2 (2013), 24.

[7] Christopher Carilli and Steve Rawlings. 2004. Science with the square kilometer array: Motivation, key science projects, standards and assumptions. *arXiv preprint astro-ph/0409274*.

[8] Doris Chen and Deshanand Singh. 2012. Invited paper: Using OpenCL to evaluate the efficiency of CPUS, GPUS, and FPGAS for information filtering. In *Proceedings of the 22nd International Conference on Field Programmable Logic and Applications (FPL'12)*. IEEE, 5–12.

[9] Doris Chen and Deshanand Singh. 2013. Fractal video compression in OpenCL: An evaluation of CPUs, GPUs, and FPGAs as acceleration platforms. In *Proceedings of the 18th Asia and South Pacific Design Automation Conference (ASP-DAC'13)*. IEEE, 297–304.

[10] Fei Chen, Yi Shan, Yu Zhang, Yu Wang, Hubertus Franke, Xiaotao Chang, and Kun Wang. 2014. Enabling FPGAs in the cloud. In *Proceedings of the 11th ACM Conference on Computing Frontiers*. ACM, 3.

[11] Michael A. Clark, P. C. La Plante, and Lincoln J. Greenhill. 2013. Accelerating radio astronomy cross-correlation with graphics processing units. *Int. J. High Perform. Comput. Appl.* 27, 2 (2013), 178–192.

[12] Tomasz S. Czajkowski, Utku Aydonat, Dmitry Denisenko, John Freeman, Michael Kinsner, David Neto, Jason Wong, Peter Yiannacouras, and Deshanand P. Singh. 2012. From OpenCL to high-performance hardware on FPGAs. In *Proceedings of the 22nd International Conference on Field Programmable Logic and Applications (FPL'12)*. IEEE, 531–534.

[13] Tomasz S. Czajkowski, David Neto, Michael Kinsner, Utku Aydonat, Jason Wong, Dmitry Denisenko, Peter Yiannacouras, John Freeman, Deshanand P. Singh, and Stephen D. Brown. 2012. OpenCL for FPGAs: Prototyping a compiler. In *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA'12)*. The Steering Committee of the World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 1.

[14] Ludovico De Souza, John D. Bunton, Ducan Campbell-Wilson, Roger J. Cappallo, and Bart Kincaid. 2007. A radio astronomy correlator optimized for the Xilinx Virtex-4 SX FPGA. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL'07)*. IEEE, 62–67.

[15] Peter E. Dewdney, Peter J. Hall, Richard T. Schilizzi, and T. Joseph L. W. Lazio. 2009. The square kilometre array. *Proc. IEEE* 97, 8 (2009), 1482–1496.

[16] Stephen A. Edwards. 2006. The challenges of synthesizing hardware from C-like languages. *IEEE Design Test Comput.* 23, 5 (2006), 375–386.

[17] Jeff Fifield, Ronan Keryell, Hervé Ratigner, Henry Styles, and Jim Wu. 2016. Optimizing OpenCL applications on Xilinx FPGA. In *Proceedings of the 4th International Workshop on OpenCL*. ACM, 5.

[18] Jeremy Fowers, Greg Brown, John Wernsing, and Greg Stitt. 2013. A performance and energy comparison of convolution on GPUs, FPGAs, and multicore processors. *ACM Trans. Architect. Code Optim.* 9, 4 (2013), 25.

[19] Mario Garrido, Jesús Grajal, M. A. Sánchez, and Oscar Gustafsson. 2013. Pipelined radix-feedforward FFT architectures. *IEEE Trans. Very Large Scale Integr. Syst.* 21, 1 (2013), 23–32.

[20] Khronos OpenCL Working Group et al. 2008. The OpenCL Specification, version 1.0. 29. Retrieved from https://www. khronos.org/registry/cl/specs/opencl-1.0.29.pdf.

[21] Giulia Guidi, Enrico Reggiani, Lorenzo Di Tucci, Gianluca Durelli, Michaela Blott, and Marco D. Santambrogio. 2016. On how to improve FPGA-based systems design productivity via SDAccel. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium Workshops*. IEEE, 247–252.

[22] Intel. 2016. Intel SDK for OpenCL Best Practices Guide. Retrieved from https://www.intel.com/content/www/us/en/ programmable/documentation/mwh1391807516407.html.

[23] Intel. 2016. Intel SDK for OpenCL Programming Guide. Retrieved from https://www.intel.com/content/www/us/en/ programmable/documentation/mwh1391807309901.html.

[24] S. Jouteux, R. Ramachandran, B. W. Stappers, P. G. Jonker, and M. Van Der Klis. 2002. Searching for pulsars in close circular binary systems. *Astron. Astrophys.* 384, 2 (2002), 532–544.

[25] Nasser Kehtarnavaz and Sidharth Mahotra. 2010. *Digital Signal Processing Laboratory: LabVIEW-Based FPGA Implementation*. Universal-Publishers.

[26] Yanbing Li, Tim Callahan, Ervan Darnell, Randolph Harr, Uday Kurkure, and Jon Stockwood. 2000. Hardware-software co-design of embedded reconfigurable architectures. In *Proceedings of the 37th Annual Design Automation Conference*. ACM, 507–512.

[27] Walid Najjar and Jason Villarreal. 2013. FPGA code accelerators-the compiler perspective. In *Proceedings of the 50th Annual Design Automation Conference*. ACM, 141.

[28] Denis Navarro, Oscar Lucia, Luis Angel Barragan, Isidoro Urriza, and Oscar Jimenez. 2013. High-level synthesis for accelerating the FPGA implementation of computationally demanding control algorithms for power converters. *IEEE Trans. Industr. Info.* 9, 3 (2013), 1371–1379.

[29] Aaron Parsons, Dan Werthimer, Donald Backer, Tim Bastian, Geoffrey Bower, Walter Brisken, Henry Chen, Adam Deller, Terry Filiba, Dale Gary et al. 2009. Digital instrumentation for the radio astronomy community. *arXiv preprint arXiv:0904.1181* (2009).

[30] Karas Pavel and Svoboda David. 2013. *Algorithms for Efficient Computation of Convolution*. INTECH Open Access Publisher.

[31] Andrew Putnam, Adrian M. Caulfield, Eric S. Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray et al. 2014. A reconfigurable fabric for accelerating large-scale datacenter services. In *Proceedings of the ACM/IEEE 41st International Symposium on Computer Architecture (ISCA'14)*. IEEE, 13–24.

[32] Scott M. Ransom, Stephen S. Eikenberry, and John Middleditch. 2002. Fourier techniques for very long astrophysical time-series analysis. *Astronom. J.* 124, 3 (2002), 1788.

[33] M. A. Sanchez, Mario Garrido, Marisa López-Vallejo, Jesús Grajal, and Carlos López-Barrio. 2005. Digital channelised receivers on FPGAs platforms. In *Proceedings of the IEEE International Radar Conference*. IEEE, 816–821.

[34] Moritz Schmid, Christian Schmitt, Frank Hannig, Gorker Alp Malazgirt, Nehir Sonmez, Arda Yurdakul, and Adrian Cristal. 2016. Big data and HPC Acceleration with Vivado HLS. In *FPGAs for Software Programmers*. Springer, 115–136.

[35] Steven W. Smith et al. 1997. Digital signal processors. *The Scientist and Engineer's Guide to Digital Signal Processing*. 503–534. Retrieved from http://www.dspguide.com/CH28.PDF.

[36] Rob V. Van Nieuwpoort and John W. Romein. 2009. Using many-core hardware to correlate radio astronomy signals. In *Proceedings of the 23rd International Conference on Supercomputing*. ACM, 440–449.

[37] Rob V. van Nieuwpoort and John W. Romein. 2011. Correlating radio astronomy signals with many-core hardware. *Int. J. Parallel Program.* 39, 1 (2011), 88–114.

[38] Haomiao Wang and Oliver Sinnen. 2015. FPGA-based acceleration of FDAS module for pulsar search. In *Proceedings of the International Conference on Field Programmable Technology (FPT'15)*. IEEE, 240–243.

[39] Loring Wirbel. 2014. *Xilinx SDAccel: A Unified Development Environment for Tomorrow Data Center*. Technical Report. Technical Report, The Linley Group, Inc.

[40] Zhiru Zhang, Yiping Fan, Wei Jiang, Guoling Han, Changqi Yang, and Jason Cong. 2008. AutoPilot: A platform-based ESL synthesis system. In *High-Level Synthesis*. Springer, 99–112.