Harmonic-Summing Module of SKA on FPGA—Optimizing the Irregular Memory Accesses

Haomiao Wang, *Student Member, IEEE*, Prabu Thiagaraj, and Oliver Sinnen

***Abstract*— The Square Kilometer Array, which will be the world's largest radio telescope, will enhance and boost a large number of science projects, including the search for pulsars. The frequency-domain acceleration search is an efficient approach to search for binary pulsars. A significant part of it is the harmonic-summing module, which is the research subject of this paper. Most of the operations in the harmonic-summing module are relatively cheap operations for field-programmable gate arrays (FPGAs). The main challenge is the large number of point accesses to off-chip memory, which are not consecutive but irregular. Having the harmonic summing on the FPGA will avoid off-board communication with other pulsar search modules, which could destroy other acceleration benefits. Two types of harmonic-summing approaches are investigated in this paper: 1) storing intermediate data in off-chip memory and 2) processing the input signals directly without storing. For the second type, two approaches of caching data are proposed and evaluated: 1) preloading points that are frequently touched and 2) preloading all necessary points that are used to generate a chunk of output points. Open Computing Language (OpenCL) is adopted to implement the proposed approaches. In an extensive experimental evaluation, the same OpenCL kernel codes are evaluated on FPGA boards and GPU cards. Regarding the proposed preloading methods, preloading all necessary points method while reordering the input signals is faster than all the other methods. While in raw performance, a single-FPGA board cannot compete with a GPU. Regarding energy dissipation, GPU costs up to 2.6× times more energy than that of FPGAs in executing the same NDRange kernels.**

***Index Terms*— Field-programmable gate arrays (FPGAs), harmonic summing, irregular memory access optimization, Open Computing Language (OpenCL).**

I. Introduction

THE Square Kilometer Array (SKA) is built to extend our understanding of the Universe and ourselves and it will be the world's largest radio telescope array when finished [6]. Many key science goals are targeted by the SKA [2] project, and one of them is strong-field tests of gravity using pulsars, which are highly magnetized rotating neutron stars. Since most pulsar signals are weaker than white noise and their details are unknown, many techniques are employed to search for different types of pulsars over a wide range of searching scales (e.g., sky coverage, frequency, bandwidth, and integration time) [15]. The enormous signal rate of the SKA makes an efficient solution only using general processors to complete the searching tasks in the given period extremely difficult.

Taking the high-performance computing ability, power consumption, and flexibility into consideration, the field-programmable gate array (FPGA) seems to be an ideal device to accelerate the central signal processor (CSP) of the SKA project. The SKA stage 1 (SKA1) project plans to adopt high-end FPGAs to accelerate part of the function modules in the CSP regarding pulsar search such as frequency-domain acceleration search. However, the general hardware description language (HDL, e.g., Verilog HDL and VHSIC hardware Description Language)-based development process makes it hard to achieve fast prototyping design and design space exploration. In addition, developers of an internationally distributed team, including nonhardware experts, would need to understand the hardware structure of FPGA devices.

To address these problems, we employed a high-level approach by using a high-level language compared to HDL. In this paper, we take a pulsar search module called harmonic summing as a case study. The harmonic-summing module is a part of the Fourier-domain acceleration search (FDAS) module that contains a compute-intensive module. The compute-intensive module performs very well on FPGAs [14], so to avoid unnecessary data transfer, it is important to have the harmonic-summing module on the FPGA. The main feature of the harmonic-summing module is that access to the input signals is irregular and this affects the hardware accelerator in achieving high-performance computing. We investigate several methods and architectures to optimize the irregular memory accesses of the harmonic-summing module and using Open Computing Language (OpenCL) for the prototype design. The main contributions are as follows.

1) *Reducing Intermediate Data Accesses:* The straightforward and proposed approaches for the harmonic-summing module are investigated and designed. The proposed approach reduces the total number of off-chip memory accesses by changing the processing order and storing the intermediate data in on-chip memory.
2) *Preloading Data:* Based on the proposed approach, two preloading data methods are investigated by: 1) loading points with high touch frequency and 2) loading necessary points that are needed to calculate a block

Manuscript received April 23, 2018; revised September 7, 2018; accepted October 11, 2018. Date of publication December 28, 2018; date of current version February 22, 2019. This research was financially supported by the SKA funding of the New Zealand government through the Ministry of Business, Innovation and Employment (MBIE). *(Corresponding author: Haomiao Wang.)*

The authors are with the Department of Electrical and Computer Engineering, The University of Auckland, Auckland 1010, New Zealand (e-mail: hwan938@aucklnaduni.ac.nz; o.sinnen@auckland.ac.nz).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2018.2882238

of points. Both these methods preload data to on-chip memory before processing and further reduce the total amount of off-chip memory accesses.

3) *Reordering Input:* Based on the preloading necessary points method, we investigate reordering the input points to improve the memory access speed. After reordering the input, the data needed for each work group are from consecutive addresses and they can be streamed to the FPGA from off-chip memory.
4) *Across Device Evaluation:* The proposed methods are implemented on FPGA using OpenCL. We adjust and port the implementations to different devices and evaluate on different series of FPGAs, general-purpose graphics processing units (GPGPUs), and CPUs for comparison.

The rest of this paper is organized as follows. Section II gives related work on optimizing irregular memory accesses and high-level tools for developing for FPGAs. Section III provides the details of the harmonic-summing module and the design goals. In Section V, two approaches of OpenCL-based designs of the harmonic-summing module are proposed and compared. Section VI presents the evaluation and results are discussed. Finally, the conclusions are given in Section VII.

II. Related Work

A. Irregular Memory Access Optimization

In hardware-based high-performance computing, the efficiency of data transfer between the accelerator and the memory system is an important factor. A large amount of research has been done to improve the memory access efficiency for accelerators such as GPGPUs [10] and FPGAs.

For some applications, the accesses to memory are irregular that limits the performance of the accelerator, and this problem has been well-studied [8]. For most applications with irregular memory access, there are mainly two types of optimization techniques: 1) reducing the number of accesses and 2) scheduling as many accesses in parallel [21]. These two methods can be applied to various platforms such as FPGAs [22]. For some graph computation problems in [20], an on-chip distributed off-chip shard memory architecture with high-performance shuffle network was investigated and the intermediate buffers were reduced to save off-chip memory bandwidth. In [23], prefetching is researched to reduce the number of memory accesses. In [9], an irregular stream buffer that targets the irregular sequences of temporally correlated memory references is proposed. Data and computation reordering are employed in [11] to improve memory hierarchy performance.

The memory access pattern problem in SKA harmonic-summing module is similar to the bit-reversal permutation in fast Fourier transform optimization [3]. Regarding the optimization of 2-D harmonic-summing calculations done in this paper, we are not aware of any prior work which investigating it on a large scale, especially in the context of acceleration devices such as GPUs and FPGAs.

B. FPGA as an Accelerator

High-end FPGAs have been widely adopted as accelerators in many commercial applications and research areas. Because of the outstanding energy-efficient performance over GPGPU devices, Microsoft-applied high-end FPGAs in their data centres [14] and FPGA-based accelerators appear in other cloud data centres as well [18]. Several science projects of different areas such as SKA [19] and CERN [17] exist that employ a large number of FPGA devices for acceleration, connected through the PCI Express (PCIe) bus or Ethernet cable.

Besides these, FPGAs are widely employed in radio astronomy projects as accelerators. In [5], hundreds of FPGAs are used to implement the correlator of the SKA Molonglo Prototype project. In [16], FPGA platforms are employed to accelerate digital channelized receivers. The Berkeley CASPER group, MeerKAT, and NRAO released an FPGA-based acceleration device for implementing the FX correlator for radio telescope array [12].

C. High-Level Synthesis

One barrier of employing FPGAs as accelerators is the usual use of the HDL-based development process that makes the time to market longer than GPGPUs and multicore processors. To address this, many high-level synthesis (HLS) tools have been released. Two primary FPGA vendors, Intel and Xilinx, provide developers with their high-level tools. Intel released several high-level development tools such as HLS compiler, which supports C++-based development and FPGA software development kit (SDK) for FPGA, which supports OpenCL [4]-based development. Xilinx provides two main tools: 1) HLS of C/C++ and SystemC and 2) SDAccel that supports OpenCL. Besides these official tools, there are several open-source HLS tools such as LegUp [1].

1) OpenCL for Intel FPGA: OpenCL is an open parallel programming language. The main advantage of OpenCL is that it is compatible with different types of acceleration devices such as GPGPUs, CPUs, and FPGAs. Intel released a dedicated FPGA development tool using OpenCL, which is called Intel FPGA Altera SDK for OpenCL (AOCL). An FPGA-based OpenCL application is divided into two parts: the host programs and the kernels for devices. The host program is written in C/C++. Before launching an OpenCL kernel in the host program, the arguments of it are set, and all necessary data are sent to the off-chip memory of FPGA devices through PCIe bus. OpenCL mainly classifies memory into two types, local memory and global memory, with the understanding that access to local memory is faster than global but sharing is limited. For OpenCL on an FPGA, local memory corresponds to on-chip memory such as block random-access memory and global memory corresponds to off-chip memory such as DDR3 on the FPGA board. In this paper, the Intel FPGAs are adopted to implement the harmonic-summing module, so the optimization syntax and techniques that are mentioned in this paper are targeting Intel FPGAs and AOCL.

2) Single Work-Item and NDRange Kernels: NDRange is an important attribute of an OpenCL kernel that represents its index space. Based on OpenCL 1.0 [7], it contains three integer values, where each value specifies the extent of the index space in a dimension. The FPGA-based OpenCL kernels

can be classified into two types based on their NDRange sizes: single work-item kernel and NDRange kernel. For the single work-item kernel, its NDRange size is (1, 1, 1), which means the index space for all three dimensions is one, resulting in a single work group with one work item. The kernel code of a single work-item kernel looks more like C/C++ code than that of NDRange kernels. However, some OpenCL-based optimization attributes are included within the kernel code. Generally, there is at least one loop in a single work-item kernel and the number of iterations equals the global work size of the NDRange kernel. The ideal case of the single work-item kernel is to launch one iteration of the outermost loop per clock cycle, which is called loop pipelining. Regarding NDRange kernels, its NDRange size is larger than (1, 1, 1) and the overall work size has to be divided into small groups. In each small work group, a small group of data is processed. The size of an NDRange kernel is normally related to the details of a task. In our research, both two kernel types are studied, and the combination of single work-item and NDRange kernels is investigated.

III. Harmonic-Summing Module

The harmonic-summing module is a part of the frequency-domain acceleration search module [15] of the pulsar search engine (PSS), whose details are depicted in Fig. 1. In the Fourier transform (FT)-based convolution module, the overlap-save algorithm [13] is employed to process the input signals in the frequency domain, and the outputs are divided into chunks, several thousand values long. The final output from the FT-based convolution module, which is also the input of the harmonic-summing module, is called filter-output-plane (FOP). The size of the FOP equals $N_{\text{temp}}N_{\text{chan}}$, with N_{temp} being the number of templates in the FT-based convolution and N_{chan} being the number of channels N_{chan}. In essence, each template is a finite-impulse-response (FIR) filter, and the FIR filter lengths of different templates are different. The total N_{temp} templates can be divided into three groups, group one (index 1 to $(N_{\text{temp}}-1/2)$), group two (index -1 to $(1-N_{\text{temp}}/2)$), and the (unfiltered) input signals (index 0, one-tap FIR filter). The number of channels is the same as the length of the input array of the FT convolution module. In our previous work [19], the FT convolution module has been implemented in an FPGA using OpenCL. Based on current requirements, an FOP contains 85×2^{21} single-precision floating-point points, that is, $N_{\text{temp}} = 85$ and $N_{\text{chan}} = 2^{21}$.

The harmonic-summing module [Fig. 1 (right)] consists of two parts: 1) harmonic plane (HP) calculation and 2) candidate detection. The task of the HP calculation part is to generate N_{hp} HPs using the FOP. First, the FOP is stretched by an integer k to obtain the kth stretch plane SP_k, which is computed separately for template group one and template group two by generating N_{hp} stretch planes with the following equation:

$$\text{SP}_k(i,j) = \text{SP}_1\left(\left\lfloor \frac{i}{k} \right\rfloor, \left\lfloor \frac{j}{k} \right\rfloor\right), \quad k = 2, 3, \ldots, N_{\text{hp}} \quad (1)$$

where SP_1 is the FOP and the ranges of i and j are $[-(N_{\text{temp}}-1)/2,\ (N_{\text{temp}}-1)/2]$ and $[0,\ N_{\text{chan}}-1]$, respectively. After all $N_{\text{hp}}-1$ stretch planes are generated, the FOP and these $N_{\text{hp}}-1$ stretch planes are progressively added to form $N_{\text{hp}}-1$ HPs

$$\text{HP}_k(i,j) = \text{HP}_{k-1}(i,j) + \text{SP}_k(i,j), \quad k = 2, 3, \ldots, N_{\text{hp}}. \quad (2)$$

It can be seen that the size of each HP_k is the same as that of the FOP.

For the candidate detection, a threshold-detection logic is applied and the potential candidates are recorded. For each HP, a threshold array (TA) that contains N_{temp} thresholds is employed and one threshold corresponds to one row (N_{chan} points) of the HP. In each HP, at most N_{cand} candidates are stored, and the maximum size of the candidate list for each dedispersion measure (DM) trial is $N_{\text{hp}}N_{\text{cand}}$. The output from the candidate detection part is the candidate list and it will be sent to the Fourier-domain candidates optimization module for further processing (which is part of the postprocessing in Fig. 1).

The details of the harmonic-summing algorithm are given in Algorithm 1, where the order of the three `for` loops can be interchanged. The basic parameters of the harmonic-summing module are shown in Table I.

TABLE I
Specification of the Harmonic-Summing Module

Parameter	Description	Value
N_{temp}	Number of templates of the FOP (row)	85
N_{chan}	Number of channels of the FOP (column)	2^{21}
N_{hp}	Total number of harmonic planes	8
N_{cand}	Number of candidates per harmonic plane	200
t_{limit}	Computation time limit of each DM trial	$88ms$

Algorithm 1 General Harmonic-Summing Algorithm (SingleHP)

$SP_1 \leftarrow$(filter-output-plane)
$CL \leftarrow 0$ {initialize the detection output}
for k = 1 to N_{hp} **do**
  for $i = -(N_{temp} - 1)/2$ to $(N_{temp} - 1)/2$ **do**
    for j = 0 to $N_{chan} - 1$ **do**
      $SP_k(i, j) \leftarrow$stretch(SP_1, k, i, j) {generate the value in stretched plane}
      $HP_k(i, j) \leftarrow HP_{k-1}(i, j) + SP_k(i, j)$ {based on the stretched plane, generate the value in harmonic plane}
      $CL \leftarrow$append detection[$HP_k(i, j)$, $TA(k, i)$] {threshold-detection logic to identify valid peak signals}
    end for
  end for
end for
Candidate List $\leftarrow CL$

IV. Proposed Methods

The main problem for the harmonic-summing module is the irregular memory accesses of the HP calculation part and this limits the data transfer efficiency. We consider two types of memory access optimization methods while designing the HP calculation part: 1) increasing the used off-chip memory bandwidth and 2) reducing the number of off-chip memory accesses. Based on the number of processed HPs at a time, two approaches are investigated: the SingleHP method (processing a single HP at a time) and the MultipleHP method (processing multiple HPs at a time).

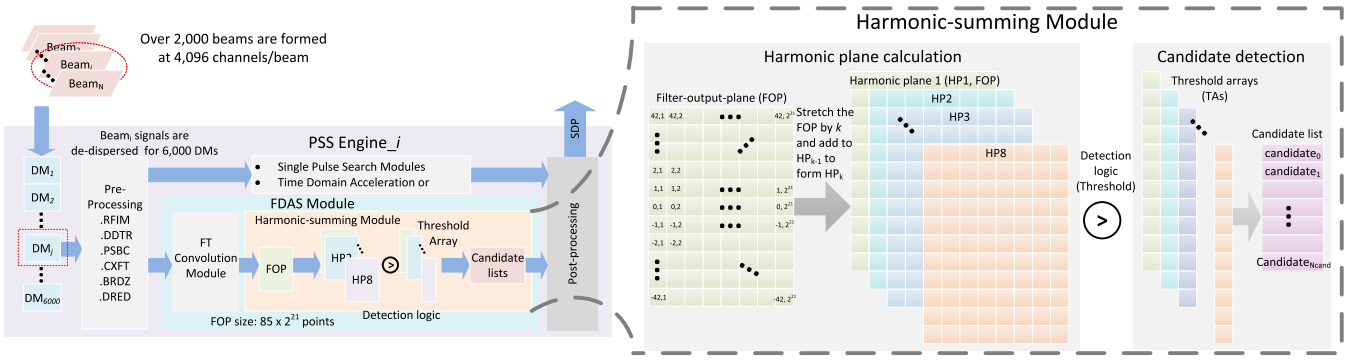


Fig. 1. Processing flow of the PSS of SKA1-MID CSP system and the details of harmonic-summing module.

A. Design Goals

Note that in Fig. 1, there are over 2000 beams that need to be processed in parallel for the SKA1-MID CSP, and they need to keep running 24/7/365 for several years. In designing the harmonic-summing module, we mainly consider the latency and energy dissipation for calculating the HPs and detecting the candidates using high-end FPGAs. There are two major factors that affect the execution latency and energy dissipation, which are: 1) parallelization capacity of an FPGA and 2) data transfer rate between the FPGA and off-chip memory. Most operations in the harmonic-summing module are floating-point operations; however, they are inexpensive functions such as floating-point additions and comparisons with a constant. For high-end FPGAs, there are hundreds of DSP blocks (to implement floating-point operations) and hundreds of thousands of logic elements that can handle these operations effectively.

In the HP calculation, the accesses to off-chip memory are not consecutive but irregular due to the index calculations in (1). Ideally, the data transfer bandwidth of any design equals the device's theoretical maximum bandwidth; however, this cannot be achieved easily in the harmonic-summing module. Taking a small size FOP (64×2^{12}) as an example, the touching frequencies of the FOP elements in calculating seven HPs are depicted in Fig. 2. Eight points from different positions are needed to calculate the point (1000, 60) of HP_8. In Fig. 2, the size of the deep red area is only 1.7% of the whole FOP; however, each value is touched 204×. The size of the high touching frequency area (zoomed-in area) is 16×2^{10} and the sum of the touching times of this area is 73.4% of the overall touching times. It can be seen that the distribution of the touching frequency and memory access while calculating do exhibit a very complex pattern. In this paper, we investigate a general design of the harmonic-summing module with low latency by optimizing memory accesses.

The input to the harmonic-summing module, which is the FOP, is up to 710 MB under current requirements and it exceeds the on-chip memory size of high-end FPGAs and other types of processors. Although the FOP can be transferred to FPGAs through the PCIe bus or Ethernet cable in practice, it is assumed in this paper that the FOP is stored in off-chip memory before processing the harmonic-summing module.

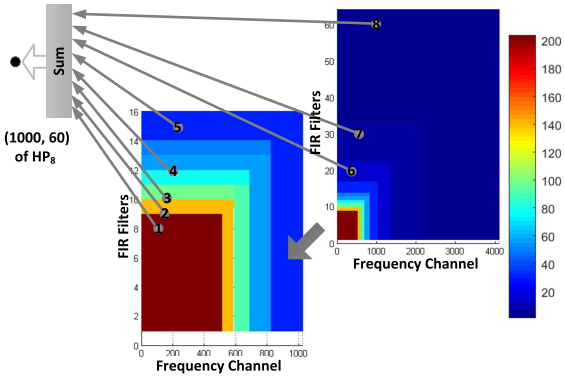


Fig. 2. Touching frequency of each point in the FOP and an example of calculating point (1000, 60) of HP_8.

Regarding the candidate detection of the harmonic-summing module, when there are more than N_{cand} candidates detected in one HP, the strategy of sorting candidates has not yet been settled in the PSS subproject. Due to the lack of a settle requirement, we investigate the methods of storing the last N_{cand} candidates. The FPGA device needs to go through all the candidates from each HP. When there are less than or equal to N_{cand} candidates in one HP, all the candidates will be recorded. Note that based on the method and process order of HP calculation, the recorded last N_{cand} candidates might vary between different approaches.

B. SINGLEHP

For the algorithm in Algorithm 1, the processor needs to calculate all HPs individually. The SINGLEHP method is a straightforward implementation of the harmonic-summing module.

To calculate the points of the kth HP HP_k ($k \geq 2$), points of the FOP and the $k-1$th HP HP_{k-1} are required. During processing, each generated point of HP_k is compared with a threshold. Since the FOP size, $N_{\text{temp}}N_{\text{chan}}$, exceeds the on-chip memory of FPGA devices, the FOP and other generated HPs have to be stored in the off-chip memory of the FPGA device.

The accesses of loading points from HP_{k-1} and storing points to HP_k are both in order and of consecutive addresses. However, the accesses of loading points from the FOP cannot

Algorithm 2 Multiple Harmonic-Summing Planes-Based Method (MULTIPLEHP)

$SP_1 \leftarrow$(filter-output-plane)
$CL \leftarrow 0$ {initialize the detection output}
for $j = 0$ to $N_{chan} - 1$**do**
  for $i = -(N_{temp} - 1)/2$ to $(N_{temp} - 1)/2$ **do**
    for $k = 1$ to N_{hp} **do**
      $SP_k(i, j) \leftarrow$stretch(SP_1, k, i, j) {generate the value in stretched plane}
      $HP_k(i, j) \leftarrow HP_{k-1}(i, j) + SP_k(i, j)$ {based on the stretched plane, generate the value in harmonic plane}
      $CL \leftarrow$detection[$HP_k(i, j)$, $TA(k, i)$] {threshold-detection logic to identify valid peak signals}
    end for
    discard[$HP_1(i, j)$, $HP_2(i, j)$, ..., $HP_{N_{hp}}(i, j)$]{discard the point of same index after detection}
  end for
end for
Candidate List $\leftarrow CL$

be calculated as a simple offset so the data cannot be streamed between off-chip memory and the device while processing. For example, eight points in Fig. 2 are loaded sequentially from point_1, which has address offset $128 + 4,096 \times (8 - 1)$, to point_8, which has $1000 + 4,096 \times (60 - 1)$.

To optimize the memory accesses of the SINGLEHP method, the overall pipeline can be parallelized to increase the used off-chip memory bandwidth, and we use that in our implementation.

C. MULTIPLEHP

In the harmonic-summing module, only the candidates are recorded for further processing and it is unnecessary to store the data of all HPs in off-chip memory. To reduce the number of off-chip memory accesses, we investigate the method to get rid of storing HPs except for the FOP. If the points of the same index in multiple or all N_{hp} HPs can be generated in parallel, these points can be discarded directly after candidate detection. Without storing the generated points back to off-chip memory, the number of overall off-chip memory accesses can be halved. For the MULTIPLEHP method, eight points in Fig. 2 are loaded in parallel.

By reordering the three `for` loops in Algorithm 1, we obtain Algorithm 2, where the innermost `for` loop can be parallelized and the points are discarded after detection.

To optimize the MULTIPLEHP method by reducing the off-chip memory accesses, part of the FOP can be loaded before calculating a chunk of points of all HPs. Two alternatives are proposed and based on the loaded data, they can be distinguished as: 1) high touching frequency (by loading as many points as possible in the high touching frequency area of the FOP) and 2) necessary points (by loading points that are needed to calculate a chunk of points in all HPs such as one or several columns of all HPs). For the second method, an FOP reordering method is proposed below to increase data transfer efficiency. Each of these three MULTIPLEHP-based methods adopted at least one type of memory accesses optimization method and the details of them are as follows.

1) Preloading Points With High Touching Frequency: To create and threshold test eight consecutive HPs, each point

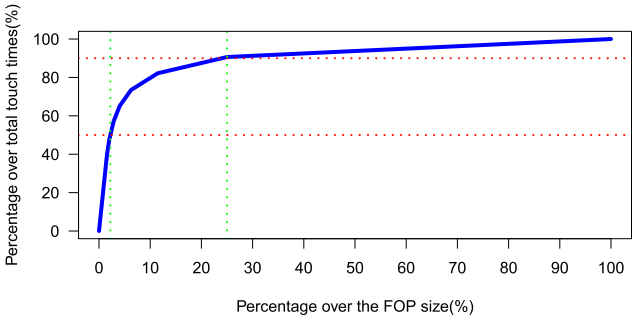


Fig. 3. Relationship between the size of preloaded points and the reduced number of GMA.

with the highest touching frequency needs to be loaded over $200\times$. If most points with high touching frequency can be preloaded, a large number of load operations can be saved. To further reduce the amount of off-chip memory accesses, part or all of the high touching frequency points can be preloaded in on-chip memory. We use MULTIPLEHP-H to represent the preloading points with the high touching frequency method.

The main factor of the MULTIPLEHP-H method is the number of preloaded high touching points $N_{\text{MultipleHP-H-preld}}$. If the points in the FOP are sorted by touching times, the relationship between the percentage of the FOP size and the percentage of overall touching times is depicted in Fig. 3. It can be seen that 2.2% points in the FOP have about 50% of overall touching times and 25% points have 90% of overall touching times.

2) Loading Necessary Points: For the Naïve MULTIPLEHP method, calculating one point with the same index of N_{hp} HPs, at most N_{hp} points need to be loaded from the FOP. However, calculating a chunk of points in all N_{hp} HPs needs less than N_{hp} times the number of points. For one column with N_{temp} points, it needs N_{temp} points for HP_1, however, $2\lceil(N_{\text{temp}} - 1)/2\rceil + 1$ points for HP_2, $2\lceil(N_{\text{temp}} - 1)/3\rceil + 1$ points for HP_3, and so on. To save loading operations, the HP calculation task can be decomposed into a number of work groups. The task of each work group is to generate a number of columns $N_{\text{MultipleHP-N-col}}$ of all N_{hp} HPs, where each column has N_{temp} points. In a pipeline, the loading part of a work group can overlap with the computing part of the previous work group. We use MULTIPLEHP-N to represent the loading necessary points method.

For the MULTIPLEHP-N method, $N_{\text{MultipleHP-N-col}}$ is an important factor that affects the reduced off-chip memory accesses. Assuming the task for each work group is to generate one column ($N_{\text{MultipleHP-N-col}} = 1$) of N_{hp} HPs ($N_{\text{hp}} N_{\text{temp}}$ points in total) and the maximum needed data is reduced to $2\sum_{i=1}^{N_{\text{hp}}}\lceil(N_{\text{temp}} - 1/2i)\rceil + N_{\text{hp}}$. When $N_{\text{MultipleHP-N-col}}$ is larger than one, more off-chip memory accesses can be reduced. However, the amount of data needed for the same HP varies based on the column index. To guarantee that the amount of data loaded for each work group is a constant (which is needed for efficient pipelining), the maximum number of points for each HP is chosen.

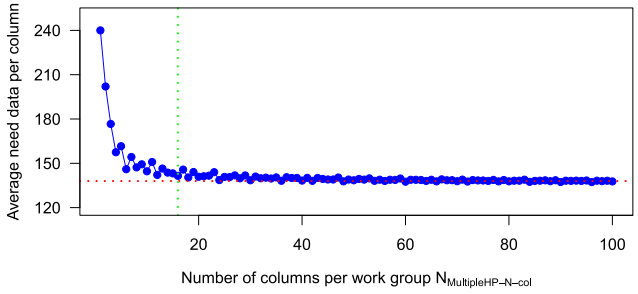


Fig. 4. Relationship between columns per work group and the number of points per column for the MULTIPLEHP-N method.

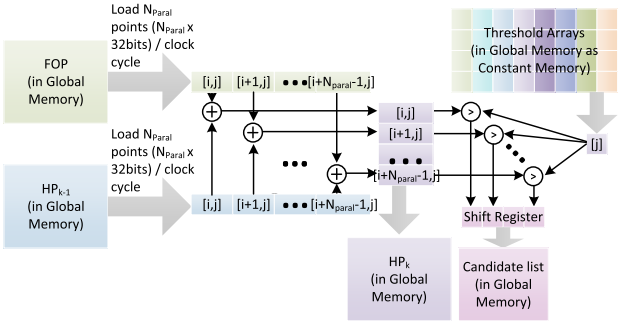


Fig. 5. Architecture of the SINGLEHP kernel.

When the $N_{\text{MultipleHP}-\text{N}-\text{col}}$ is specified, the needed number of columns for each HP can be listed and then the number of needed points for $N_{\text{MultipleHP}-\text{N}-\text{col}}$ columns can be calculated. The average needed points per column for a work group is plotted in Fig. 4. It can be seen that the average number drops fast when the value of $N_{\text{MultipleHP}-\text{N}-\text{col}}$ is smaller than 16 (green dotted line) and it decreases slightly toward 64 (red dotted line) as $N_{\text{MultipleHP}-\text{N}-\text{col}}$ increases. Besides these, the larger $N_{\text{MultipleHP}-\text{N}-\text{col}}$, the larger space it needs in the on-chip memory. If $N_{\text{MultipleHP}-\text{N}-\text{col}}$ is too large, the on-chip memory size might limit $N_{\text{MultipleHP}-\text{N}-\text{col}}$. Consequently, it is unnecessary to assign tens or hundreds of columns to a work group.

3) Reordering the FOP: Compared with the Naïve MULTIPLEHP method, the MULTIPLEHP-N method can further reduce the total amount of off-chip memory accesses. However, the points needed for each work group are from at least N_{hp} blocks in FOP and they are from nonconsecutive addresses. Thus, the points for each work group cannot be streamed between off-chip memory and the FPGA device.

To optimize the used off-chip memory bandwidth of the MULTIPLEHP-N method, we propose the MULTIPLEHP-R method that reorders the FOP to form the reordered FOP (RFOP). After reordering, the needed points to calculate $N_{\text{MultipleHP}-\text{R}-\text{col}}$ columns of all HPs are from consecutive addresses that can be streamed to the FPGA while processing. However, the size of the RFOP is larger than the standard FOP size. Theoretically, the number of rows in the RFOP is increased from N_{temp} to the average needed points per column in Fig. 4, and the larger the $N_{\text{MultipleHP}-\text{R}-\text{col}}$, the smaller the relative size of RFOP. The latency of extra data transfer and FOP reordering have to be considered in the evaluation of the MULTIPLEHP-R method.

V. ARCHITECTURE AND OPTIMIZATION

In this section, we investigate the architecture of the proposed methods and employ OpenCL as the high-level language, whose kernels can be executed on both FPGAs and GPUs. The optimization techniques and syntax are dedicated to FPGAs.

A. SINGLEHP *Kernel*

The basic structure of the SINGLEHP kernel while processing the kth HP HP_k is depicted in Fig. 5, where N_{paral} is the parallelization factor that is restricted by global memory (off-chip memory in this research) bandwidth (GMB) and the logic resources of the FPGA. One optimization goal for the SINGLEHP kernel is to find the maximum parallelization factor $N_{paral-max}$ that leads to a required GMB which equals the maximum physical off-chip memory bandwidth of a specific device.

The FOP, HP_{k-1}, HP_k, candidate list and TA are all stored in global memory before launching the kernel. When the kernel is launched, N_{paral} points from HP_{k-1} and $\lceil (N_{\text{hp}}/k) \rceil$ points from FOP are loaded per clock cycle. These points are summed, according to (1) and (2), to calculate N_{paral} points of HP_k. The generated N_{paral} points are compared with the corresponding thresholds and detected candidates are saved in a shift register or local memory (on-chip memory in this paper) of length N_{cand}, until all FOP points have been processed. Then, these N_{paral} points overwrite the values at the same address as HP_{k-1}.

In OpenCL, both single work-item and NDRange kernel types can be applied to implement the SINGLEHP kernel.

For the NDRange kernel, kernel vectorization, which increases the amount of work that a compute unit can perform in parallel, and compute unit replication, which increases the number of compute units, are techniques can be employed to parallelize the kernel.

The SINGLEHP kernel can be implemented as a generic kernel that needs to be launched N_{hp} times (multiple launches) or a specific kernel that only needs to be launched once (single launch) to generate the candidate list of H_{hp} HPs. The overhead of launching a kernel such as setting kernel arguments will affect the overall latency, especially when the kernel execution latency is short. Therefore, the kernel launch time is an important factor for the SINGLEHP kernel. Multiple launches provide more flexibility than the single launch SINGLEHP kernel, as it can be used for any HP configuration. Both single- and multiple-launch kernels are evaluated in Section VI.

B. MULTIPLEHP *Methods-Based Kernels*

Although parallelizing the SINGLEHP kernel can shorten kernel execution latency by increasing GMB, the total amount of global memory accesses (GMAs) is not reduced. The main advantage of the MULTIPLEHP method is the reduction of the required GMA by processing multiple HPs at the same

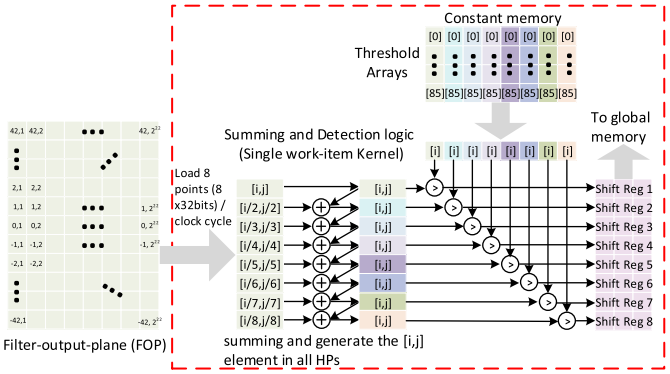


Fig. 6. Architecture of the Naïve MULTIPLEHP kernel (Single work-item).

time. Three optimization techniques are investigated for the MULTIPLEHP-based methods in the following.

1) Naïve MULTIPLEHP*:* The Naïve MULTIPLEHP kernel calculates N_{paral} points of all N_{hp} HPs with the same index, where N_{paral} is the parallelization factor. The architecture of the Naïve MULTIPLEHP kernel is shown in Fig. 6, where the operations in the red dotted rectangle have to be parallelized N_{paral} times to process N_{paral} points of all HPs. In OpenCL, this is implemented as a single work-item type, and the `#unroll pragma` N_{paral} is added before the main `for` loop in the kernel code.

The FOP is stored in global memory and N_{hp} points $((i,\ j),\ (\lfloor i/2\rfloor,\ \lfloor j/2\rfloor),\ldots,(\lfloor i/N_{\text{hp}}\rfloor,\ \lfloor j/N_{\text{hp}}\rfloor))$ are loaded in parallel to generate point $(i,\ j)$ of all N_{hp} HPs. Then, these N_{hp} points are compared with the corresponding thresholds and stored as constant memory. N_{hp} independent arrays of size N_{cand}, one corresponding to each HP, are employed to store the candidates. Both local memory and shift register can be adopted to implement N_{hp} arrays and the performance difference is evaluated in Section VI. After all N_{hp} HPs have been processed, the N_{hp} candidate arrays are sent back to global memory. Because the loading accesses to the global memory are irregular, a high memory stall percentage will impede the kernel from achieving a high performance.

2) MULTIPLEHP-H*:* The MULTIPLEHP**-**H kernel builds on the Naïve MULTIPLEHP kernel, which is a single work-item kernel. MULTIPLEHP**-**H is, however, split into two parts: preloading and computing. The $N_{\text{MultipleHP}-\text{H}-\text{preld}}$ preloaded points that can be seen as constant cache memory are loaded into an FIFO at runtime. In processing one FOP, there is no overlap between the prefetching and computing parts. The available local memory of the FPGA and the number of high touching frequency points affects the performance of the MULTIPLEHP**-**H kernel. If the FOP size is comparable to the available local memory, most of the points with high touching frequency can be loaded and then most of the GMAs can be reduced. However, if the number of high touching frequency points is significantly larger than the local memory size, it is impossible for the device to hold most of these important points. Besides this, the large proportion of the used on-chip memory might lead to a decrease in kernel frequency. In this case, it is necessary to search for the suitable $N_{\text{MultipleHP}-\text{H}-\text{preld}}$ for the target FPGA by testing a range of preloading data sizes. The relationship between the $N_{\text{MultipleHP}-\text{H}-\text{preld}}$ and the kernel performance is investigated in Section VI.

3) MULTIPLEHP-*N:* The MULTIPLEHP-N method is a memory accesses saving method, as discussed in Section IV-C2. It decomposes the overall task into a number of work groups, and the task for each work group is to process $N_{\text{MultipleHP}-\text{N}-\text{col}}$ columns of all HPs. The NDRange kernel type is employed and the preloading part of a work group overlaps with the computing part of the previous work group. For the NDRange kernel, different work groups do not share local memory and it is inefficient to save candidates in global memory during processing. The hybrid kernel type that contains both single work-item type and NDRange type is employed to implement the preloading necessary points kernel (MULTIPLEHP**-**N).

The relationship between the work-group size of the NDRange kernel and the execution latency is studied next. The task of each work group is to generate $N_{\text{MultipleHP}-\text{N}-\text{col}}$ columns of all HPs, which contain $N_{\text{MultipleHP}-\text{N}-\text{col}}N_{\text{chan}}$ points. For each work group, $N_{\text{hp}}N_{\text{MultipleHP}-\text{N}-\text{col}}N_{\text{chan}}$ points are stored in local memory using the OpenCL barrier technique. Some points in these $N_{\text{hp}}N_{\text{MultipleHP}-\text{N}-\text{col}}N_{\text{chan}}$ points are from the same index in the FOP and they only need to be loaded once.

The NDRange HP calculation kernel is connected with the single work-item candidate detection kernel through OpenCL channels, which is a FIFO buffer in essence. The OpenCL channel is an effective approach to transfer data between different kernels without touching global memory. The candidate detection part is the same as that of Naïve MULTIPLEHP kernel and MULTIPLEHP-H kernel.

4) MULTIPLEHP-R*:* The MULTIPLEHP**-**R kernel is based on the MULTIPLEHP**-**N kernel, and the main difference is the order of the data for each work group. After reordering, the points needed for a work group are from consecutive addresses.

The total amount of needed data for a work group ($N_{\text{total/wg}}$) is the product of average needed data per column times the number of columns per work group ($N_{\text{MultipleHP}-\text{R}-\text{col}}$) (see also Fig. 4). To achieve stream mode in GMA, the number of loaded points per clock cycle ($N_{\text{lpoints/cc}}$) has to be an integer constant, which makes the product of $N_{\text{lpoints/cc}}$ and work-group size ($S_{\text{workgroup}}$) usually larger than $N_{\text{total/wg}}$ and never less ($N_{\text{total/wg}} \le N_{\text{lpoints/cc}}S_{\text{workgroup}}$). In case of difference, the input array for each work group has to be padded with dummy values at the end. The relationship between $N_{\text{lpoints/cc}}$ and $N_{\text{MultipleHP}-\text{R}-\text{col}}$ is shown in Table II, where $N_{\text{points/wi}}$ is the executed points of all HPs per work item. The value in the bracket ($\times *$) represents the ratio of total loaded points over the FOP size:

$$\frac{N_{\text{lpoints/cc}}S_{\text{workgroup}}N_{\text{workgroup}}}{N_{\text{chan}}N_{\text{temp}}}$$

where $N_{\text{workgroup}}$ is the total number of work groups. We use MULTIPLEHP-R-($N_{\text{MultipleHP}-\text{R}-\text{col}}$, $N_{\text{points/wi}}$) to represent kernel MULTIPLEHP**-**R with the specified settings. The larger

TABLE II

NUMBER OF LOADED POINTS PER CLOCK CYCLE $N_{\text{lpoints/cc}}$ OF DIFFERENT $N_{\text{points/wi}}$ AND $N_{\text{MultipleHP-R-col}}$ COMBINATIONS FOR GENERAL AND OPTIMIZED MULTIPLEHP-R (NUMBER IN (×∗) SHOWS TOTAL LOADED POINTS IN RELATION TO FOP SIZE)

$N_{points/wi}$ / Columns	Opt.	×1	×2	×4	×8
1	×	3 (×3)	6 (×3)	12 (×3)	23 (×2.9)
	√	4 (×4)	8 (×4)	16 (×4)	32 (×4)
4	×	2 (×2)	4 (×2)	8 (×2)	15 (×1.9)
	√	2 (×2)	4 (×2)	8 (×2)	16 (×2)
16	×	2 (×2)	4 (×2)	7 (×1.8)	13 (×1.6)
	√	2 (×2)	4 (×2)	8 (×2)	16 (×2)
64	×	2 (×2)	4 (×2)	7 (×1.8)	13 (×1.6)
	√	2 (×2)	4 (×2)	8 (×2)	16 (×2)

$N_{\text{MultipleHP-R-col}}$ and $N_{\text{points/wi}}$, the less data needs to be loaded from global memory. Because of physical limitations, if the needed bandwidth of loading $N_{\text{lpoints/cc}}$ points exceeds the maximum device off-chip memory bandwidth, the performance will not increase, and the kernel was not implemented.

It is clear that $N_{\text{lpoints/cc}}$, $N_{\text{MultipleHP-R-col}}$, and $N_{\text{points/wi}}$ are the three main parameters for kernel MULTIPLEHP-R and they have to be balanced to achieve good performance. Using the AOCL compiler, it becomes apparent that using the number that is powers of 2 for $N_{\text{lpoints/cc}}$ results in more efficient implementations than other numbers. Hence, to make the value of $N_{\text{lpoints/cc}}$ equal to a power of 2, more data might need to be loaded for each work group. Taking the kernel MULTIPLEHP-R-$(8, 8)$ for example, the value of $N_{\text{lpoints/cc}}$ is 13 and it has to be increased to the nearest power of 2, which is 16. Since the amount of loaded data per work group is $N_{\text{lpoints/cc}}S_{\text{workgroup}}$, the increase in $N_{\text{lpoints/cc}}$ leads to an increase in loading operations. The optimized $N_{\text{lpoints/cc}}$, where $N_{\text{lpoints/cc}}$ is the lowest power of 2 greater or equal to the corresponding $N_{\text{lpoints/cc}}$ of values without optimization in Table II. When $N_{\text{MultipleHP-R-col}} \geq 4$, the total loaded data are twice the FOP size (value in the bracket).

For the hybrid kernels (combining NDRange and single work-item kernels) MULTIPLEHP-H, MULTIPLEHP-N, and MULTIPLEHP-R, adding vectorization or replication attributes can only parallelize the NDRange part but not the single work-item part, and it has to be parallelized manually in the kernel code.

VI. EXPERIMENTAL EVALUATION

To experimentally evaluate the harmonic-summing module, the straightforward SINGLEHP method and the proposed MULTIPLEHP-based methods are evaluated in this section. The FPGA-based harmonic-summing kernels are assessed according to their resource usage, execution latency, and energy dissipation. In addition, we compare those results to latency and energy dissipation of the kernels implemented on GPU and multicore CPUs.

A. Experimental Setup

Four different devices are employed to evaluate the performance of the proposed designs on CPU, GPU, and FPGAs. Two types of Intel FPGAs (Stratix V, referred to as **S5**, and Arria 10, referred to as **A10**) are compared with one midrange AMD R7 GPU, referred to as **R7**, and a general Intel *i*7 CPU, referred to as **I7**. The specifications of these platforms are given in Table III. The FPGA and GPU cards are connected to the host processor through the PCIe bus.

All FPGA-targeting OpenCL kernels are compiled using AOC version 16.0.0.222 and GPU-targeting kernels are compiled using AMD APP SDK version 3.0. For the CPU platform, the C code, which is based on the same kernel code, is compiled using GNU Compiler Collection, using OpenMP for parallelisation. Both OpenCL and OpenMP can be employed to parallelize operations in for loops for CPU. However, OpenCL is up to 2× times slower than that of C program on the employed CPU. To make it fair for CPU, we employed C code for CPU. Comparison between OpenCL and OpenMP on CPU is not investigated in this paper.

Since the top half (from row 1 to $(N_{\text{temp}} - 1/2)$) and the bottom half (from row $(1 - N_{\text{temp}}/2)$ to -1) are independent for the harmonic-summing module, we investigate the performance, in terms of the execution latency and energy dissipation, of half of the FOP as specified in Table I, which size is 42×2^{21}. Remember from Section III that the upper and lower halves of the FOP can be processed independently and the required processing is identical. The size of the candidate list is 200.

B. Resource Usage

Because the harmonic-summing module is not a compute-intensive application, the DSP block utilization of all implementations is less than 5%. We discuss the logic utilization, RAM blocks utilization, and kernel frequency in this section.

1) SINGLEHP*:* A series of SINGLEHP kernels with different parallelization factors N_{paral} is evaluated. These kernels are employed to generate eight HPs of half FOP. All these kernels are NDRange kernels and the work-group sizes are set to 256, which is the default size set by Intel offline compiler when there is a barrier in the kernel code. While the relationship between the work-group size and kernel performance was not investigated in detail in this paper, some preliminary results showed that there was no clear relationship between them, neither in relation to resource consumption. The usage of logic cells and RAM blocks of these kernels is given in Fig. 7, where "*S*" and "*M*" represent single launch and multiple launches, and "*V*" and "*R*" represent kernel vectorization and replication. The candidate detection part is included, and the local memory is employed to store the candidate during processing. When $N_{\text{paral}} = 1$, it means that the kernel is not parallelized and that vectorization and replication are not employed.

It can be seen that the usage of both resources increases as N_{paral} increases. These trends are similar to those observed for execution on *S*5. The kernel frequency drops as the resource usage increases across all kernels. Taking the kernel SINGLEHP-(M, V) on $A10$ as an example, its frequency

TABLE III
SPECIFICATIONS OF CPU, GPU, AND FPGA PLATFORMS

Device	Terasic DE5-Net (**S5**)	Nallatech 385A (**A10**)	Sapphire Nitro R7 370 (**R7**)	Intel CPU Host (**I7**)
Hardware	Intel Stratix V 5SGXA7	Intel Arria 10 GX1150	AMD Radeon R7 370	Intel Core i7-6700K
Technology	$28nm$	$20nm$	$28nm$	$14nm$
Compute resource	622,000 LEs 256 DSP blocks	1,506,000 LEs 1,518 DSP blocks	1,024 Stream Processors (16 Compute Units)	4 Cores (8 threads)
On-chip memory size	$50Mb$	$53Mb$	—	$64Mb$
Off-chip memory size	2 x $2GB$ DDR3	2 x $4GB$ DDR3	$4GB$ GDDR5	$64GB$ DDR4
Memory interface width	2 x 64-bit	2 x 72-bit	256-bit	—
Max clock frequency	$600MHz$	$1.5GHz$	$985MHz$	$4.2GHz$
Max power consumption	—	75W	150W	—

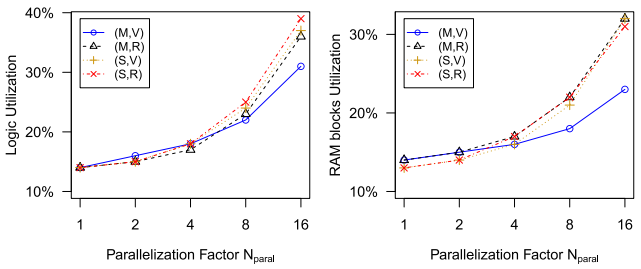


Fig. 7. Logic utilization and RAM block usage of SINGLEHP kernels on $A10$.

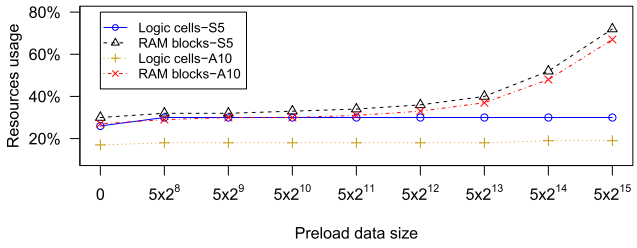


Fig. 8. Resource usage of MULTIPLEHP-H on $S5$ and $A10$.

decreases from 266.9 MHz at $N_{\text{paral}} = 1$ to 236.8 MHz at $N_{\text{paral}} = 16$.

2) MULTIPLEHP*:* In terms of the MULTIPLEHP designs, Naïve MULTIPLEHP, MULTIPLEHP-H, MULTIPLEHP-N, and MULTIPLEHP-R (Section V) are evaluated.

a) Naïve MULTIPLEHP *and* MULTIPLEHP-H*:* MULTIPLEHP-H is based on the Naïve MULTIPLEHP-H, and the main difference is that it preloads a block of data before calculating. The resource usages of these kernels are plotted over the preloaded data size in Fig. 8. The value points for $N_{\text{MultipleHP}-\text{H}-\text{preld}} = 0$ correspond to Naïve MULTIPLEHP. The logic utilization is not affected by the increase in $N_{\text{MultipleHP}-\text{H}-\text{preld}}$, however, the RAM blocks utilization increases. The largest $N_{\text{MultipleHP}-\text{H}-\text{preld}}$ that can be compiled successfully on $S5$ and $A10$ are both 5×2^{15}. The kernel frequency ranges from 207 to 219 MHz for $S5$-based implementations and 204 to 236 MHz for $A10$-based implementations.

b) MULTIPLEHP-N *and* MULTIPLEHP-R*:* In contrast to MULTIPLEHP-H, kernel MULTIPLEHP-N and MULTIPLEHP-R do not depend heavily on local memory size.

TABLE IV
RESOURCE USAGE AND KERNEL FREQUENCY OF MULTIPLEHP-N WITH CANDIDATE DETECTION ON $A10$

Columns	1	2	4	6	8
Logic cells	17%	25%	25%	29%	30%
RAM blocks	19%	44%	49%	56%	69%
Frequency (MHz)	276.5	193.4	171.1	148.7	165.5
Latency (ms)	328.0	469.0	530.1	610.1	548.1

MULTIPLEHP-R is based on MULTIPLEHP-N, however, it does not need to load points from different locations.

For MULTIPLEHP-N, different column numbers ($N_{\text{MultipleHP}-\text{N}-\text{col}}$) are evaluated, and the results are listed in Table IV. As can be seen with increasing $N_{\text{MultipleHP}-\text{N}-\text{col}}$, both logic cell and RAM block utilization increase. For most of the kernels, the kernel frequency is decreased as $N_{\text{MultipleHP}-\text{N}-\text{col}}$ increases. MULTIPLEHP-N-(8) has a higher kernel frequency than MULTIPLEHP-N-(6). The main reason is that it is more efficient for a compiler when the number of operations is a power of 2 such as 8.

Regarding MULTIPLEHP-R, to arrange the data for each work group into a consecutive address area, the half FOP is reordered into a half RFOP (Section IV-C), in the host program using `memcpy()`. The reordering latency on the employed host is 87.8 ms, which is achieved using a single thread and does not include the data transfer time. For the FDAS module, the output from the FT convolution in Fig. 1 needs to be revised based on the applied algorithm. If the MULTIPLEHP-R method is employed, the FOP reorder will be combined with other transformation functions and the reordering latency is not investigated in this paper.

The performance of two variants of MULTIPLEHP-R kernels (generating 16 and 64 columns of all eight HPs per work group) is evaluated, which is shown in Table V. Four different points per work-item values $N_{\text{points/wi}}$ (1, 2, 4, and 8) are tested in this paper. Since the values of $N_{lp/cc}$ for $N_{\text{points/wi}} = 1$ and $N_{\text{points/wi}} = 2$ are already powers of 2, so we focus on the other two conditions ($N_{\text{points/wi}} = 4$ and $N_{\text{points/wi}} = 8$) and the resource usage of the general and the optimized implementations with these values are given in Table V. For the optimized implementations, the values of $N_{\text{lpoints/cc}}$ are powers of 2 and this costs fewer logic cells than the general implementations. Since more points are loaded per clock cycle, the optimized implementations consume more RAM blocks.

TABLE V

RESOURCE USAGE AND EXECUTION LATENCY OF MULTIPLEHP-R (NDRANGE PART ONLY) WITH ($N_{lp/cc}$ IS POWER OF 2) AND WITHOUT OPTIMIZING GMB ON $A10$ (WITHOUT CANDIDATE DETECTION)

$N_{MultipleHP-R-col}$		16					64				
$N_{p/wi}$	$N_{lp/cc}$	Logic utilisation	RAM blocks	Freq. (MHz)	Latency (ms)	Occup.	Logic utilisation	RAM blocks	Freq. (MHz)	Latency (ms)	Occup.
1	2	14%	12%	269.2	350.8	93.4%	15%	12%	266.7	336.2	98.3%
2	4	15%	13%	286.5	176.7	87%	16%	12%	252.8	180.9	96.6%
4	7	22%	14%	196.3	189.1	59.4%	22%	15%	161.9	248.1	55%
4	8	19%	16%	263.0	107.7	77.8%	19%	30%	229.6	102.5	93.8%
8	13	35%	17%	130.3	163.9	51.6%	37%	17%	135.1	158.4	51.6%
8	16	28%	29%	168.1	93.1	70.5%	29%	48%	171.4	71.7	89.7%

Besides these, the kernel frequency of the optimized implementations is higher than that of general implementations.

Since $N_{\text{MultipleHP}-\text{R}-\text{col}}$, $N_{p/\text{wi}}$, and $N_{\text{lp/cc}}$ are three main parameters that affect the performance of MULTIPLEHP-R, we investigate the trend of changing these parameters, but here without candidate detection. Hence, the values in Table V are only for the NDRange part. We do this because after combining with the candidate detection, some of the MULTIPLEHP-R kernels such as MULTIPLEHP-R-$(64, 8)$ cannot be compiled because of the limited resources, and we wanted to explore the influence of the parameters in a good range. We employ the MULTIPLEHP-R-$(16, 4)$ kernel with candidate detection, which can be compiled on both $S5$ and $A10$, to compare with other methods. In the future, as FPGA technology upgrades, the number of on-chip logic cells and RAM blocks increases. The values of $N_{\text{MultipleHP}-\text{R}-\text{col}}$ and $N_{\text{points/wi}}$ can be raised, and the execution latency is likely to be faster than that achieved in Table V.

The depth of each channel that connects the harmonic-summing and threshold detection for MULTIPLEHP-N and MULTIPLEHP-R is 1. The influence of channel depth is not investigated in this research.

C. Latency Evaluation

1) Harmonic Plane Calculation on FPGA: We evaluate the overall execution latency of the harmonic-summing module, including the HP calculation and the candidate detection. The points of the eight HP are compared with the result of a MATLAB implementation to verify the correctness of the HP calculation in different designs.

a) SINGLEHP*:* The used GMBs and execution latencies of the SINGLEHP kernel with various N_{paral} in Section VI-B are shown in Fig. 9, where the green dotted line in Fig. 9 (left) is the theoretical maximum bandwidth that is supported by the board support package (BSP). Regarding the theoretical maximum bandwidth for a specific kernel, the kernel clock frequency restricts the bandwidth when it is lower than (2133/4) MHz, where 4 is based on quad rank, and it is not included in Fig. 10. As N_{paral} increases, the GMBs of all SINGLEHP kernels increase, however, not all execution latencies are decreased.

For the two multiple launches ("M") kernels, SINGLEHP-(M, V) and SINGLEHP-(M, R), the launching overhead is hundreds of times smaller than the kernel execution latency and hence negligible. For the single-launch kernel SINGLEHP-(S, V), the global memory occupancy decreased dramatically when N_{paral} is larger than 4. This is caused by a large number of irregular accesses to memory in parallel. For SINGLEHP-(S, R), the performance stops increasing when N_{paral} is larger than 8. When $N_{\text{paral}} = 8$, kernel SINGLEHP-(S, R) performs better than other kernels and the SINGLEHP-(M, R) kernel performs best when $N_{\text{paral}} = 16$, which is about $7.5\times$ faster than SINGLEHP-$(M,)$ with $N_{\text{paral}} = 1$.

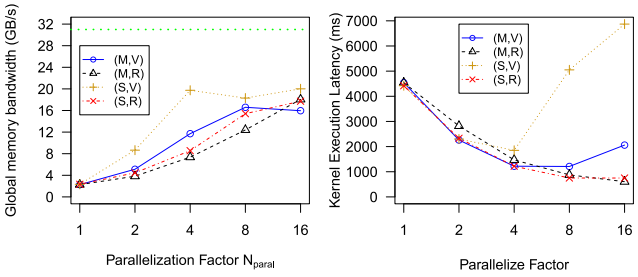


Fig. 9. GMBs and execution latency of SINGLEHP on $A10$.

b) Naïve MULTIPLEHP*:* The execution latency of kernel Naïve MULTIPLEHP on $S5$ is over 1 s (1210 ms); however, the same kernel achieves a better performance, which is less than 400 ms on $A10$. The main reason is the kernel frequency achieved on $A10$ is over two times higher than that on $S5$. This might be caused by BSPs provided by different vendors. Since $A10$ has over $2\times$ times the logic resource and DSP blocks, it might be easier for the compiler to optimize the placement and routing.

c) MULTIPLEHP-H*:* The relationship between the number of preloaded data points $N_{\text{MultipleHP}-\text{H}-\text{preld}}$ and the execution latency of MULTIPLEHP-H is investigated on both $S5$ and $A10$. The half FOP is transposed and then processed row by row (each row has $(N_{\text{temp}} - 1/2)$ points). The execution latencies of these kernels are depicted in Fig. 10. Preloading data reduces the number of accesses to off-chip memory; however, it does not reduce the number of clock cycles when processing a large amount of data. The increase in preload data size leads to different logic utilization, RAM block usage, and kernel frequency. The jaggy line in Fig. 10 is caused by the kernel frequency, and there is no clear relationship between kernel frequency and $N_{\text{MultipleHP}-\text{H}-\text{preld}}$.

Even the largest $N_{\text{MultipleHP}-\text{H}-\text{preld}}$ (5×2^{15}) used in the experiments, which is limited by the available RAM block

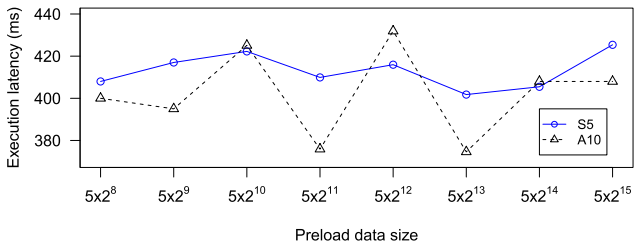


Fig. 10. Execution latencies of the MULTIPLEHP-H kernels with different sizes of preloaded points.

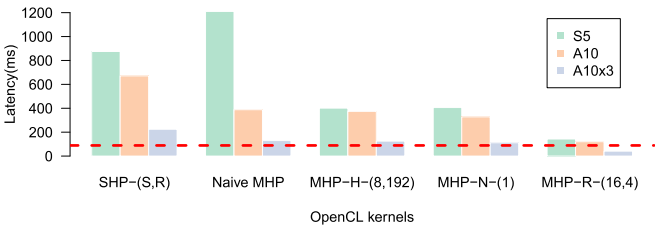


Fig. 11. Execution latency of proposed harmonic-summing methods with candidate detection on $A10$, where SHP represents SINGLEHP and MHP represents MULTIPLEHP.

on the FPGA, contains only 4.7% of the total number of all memory accesses. The best performance achieved on $S5$ and $A10$ is both by executing kernel MULTIPLEHP-H-(5×2^{13}). Some improvements could be made by overlapping the loading of the high touching frequency points with the computing part but not substantially. Overall, MULTIPLEHP-H is not gaining performance if the local memory size is not large enough to hold most of the points with high touching frequency.

d) MULTIPLEHP-N*:* For kernel MULTIPLEHP-N, the necessary data for each work group are from nonconsecutive addresses and this affects the loading section in achieving streaming mode, which is crucial to fully use the available theoretical maximum bandwidth. Although executing more columns per work group can reduce GMA, the value of $N_{\text{MultipleHP}-\text{N}-\text{col}}$ does not affect performance. The execution latency of MULTIPLEHP-N is affected by the kernel frequency, which is given in Table IV. We employ the kernel with the fastest execution latency to compare with other methods, which is MULTIPLEHP-N-(1).

e) MULTIPLEHP-R*:* The kernel execution latency and global memory occupancy during execution on $A10$ are given in Table V as well. When the value of $N_{\text{lpoints/cc}}$ is a power of 2, the execution latency decreases as $N_{\text{points/wi}}$ increases. Although the occupancy of loading operations drops, the values for the optimized kernels decrease slower than that of the general kernels. The fastest variant of MULTIPLEHP-R in Table V is MULTIPLEHP-R-(64, 8). By adding the candidate detection, the execution latency increases, however, faster than other MULTIPLEHP kernels. For kernel MULTIPLEHP-R-(16, 4), the execution latencies on a single $S5$ and $A10$ are 143 and 120 ms, respectively.

f) Overall Comparison: Based on the discussion earlier, the execution latency of each well-optimized method with candidate detection is shown in Fig. 11, and both types of FPGA devices are evaluated, where the red dashed line is the current time limitation for the SKA harmonic summing. We also evaluate a setting where three A10 FPGAs are used in parallel. The execution latency of MULTIPLEHP-R in Fig. 11 does not include the reordering overhead.

Note that SINGLEHP-(M, R) and $N_{\text{paral}} = 16$ on $S5$ cost a large number of RAM blocks and cannot be compiled. Hence, SINGLEHP-(S, R) with $N_{\text{paral}} = 8$ is used. The execution latency of MULTIPLEHP-N-(1) is faster than that of Naïve MULTIPLEHP and MULTIPLEHP-H-(8, 192); however, it is about 3× slower than MULTIPLEHP-R-(16, 4). Except for

TABLE VI
SPEEDUP OF MULTICORE CPU, GPU, AND FPGA PLATFORMS OVER SINGLE-CORE CPU IN PROCESSING SINGLEHP KERNEL INCLUDING CANDIDATES DETECTION

Device	Execution latency(ms)	Speedup over $I7-1C$
$S5$	875	4.8
$A10$	671	6.2
$R7$	119	**35.2**
$I7-4C$	1, 100	3.8
$I7-1C$	4, 174	1

Naïve MULTIPLEHP on $S5$, all MULTIPLEHP kernels perform better than SINGLEHP-(S, R) with $N_{\text{paral}} = 8$.

Although the performance is improved by adopting MULTIPLEHP kernels, none of these kernels on a single $A10$ meets the requirement. By installing three $A10$ FPGA cards, they can work in parallel by processing three different half FOPs. The average execution latencies of half FOP using three $A10$ cards are shown in Fig. 11 as well. It can be seen that kernel MULTIPLEHP-R on three $A10$ cards is over 2× times faster than the required time limitation, so three $A10$ cards can process the whole FOP while meeting the requirements.

2) Comparison With CPU and GPU: We are now comparing the performance of the proposed kernels on GPU (using adjusted OpenCL code) and CPU (using equivalent OpenMP implementations). Since single work-item kernels on GPUs cannot exploit their performance potential, for fairness we only compare the performance of NDRange kernels on FPGA and GPU devices. Regarding the optimization syntax for FPGAs, they are not employed in the NDRange kernels but the single work-item kernels. To make it further fair to compare with GPUs, there is no code that is best only for FPGA devices in the NDRange code such as shift registers.

SINGLEHP-$(M,)$ is evaluated on the $R7$ GPU, and the host argument settings are the same as for the FPGA-based implementation. The straightforward C code with OpenMP directives, using three levels of `for` loops, which is the same as Algorithm 1, is evaluated on the $I7$ CPU using all four cores. The execution latency of SINGLEHP using one core of $I7$ CPU $(I7-1C)$ is taken as the baseline, and the speedups over it on other devices are given in Table VI, where $I7-4C$ represents using four cores of the $I7$ CPU. It can be seen that $R7$ performs best among these devices and it is about 3.6× faster than the $A10$ FPGA. The $R7$ has two major advantages over $S5$ and $A10$: 1) operating frequency and 2) maximum off-chip memory bandwidth. Although the

maximum frequency of $A10$ is higher than $R7$, the maximum frequencies of the implemented kernels are less than 300 MHz in this paper.

Regarding the MULTIPLEHP kernels on GPU, a similar OpenCL code as used for the FPGA kernels of Naïve MULTIPLEHP and MULTIPLEHP-H are tested. The execution latencies of these kernels are both over 30 s, which is about a hundred times slower than that of a single $A10$ FPGA. Because these two variants are single work-item kernels, the GPU cannot parallelize operations on multiple stream processors. For the fastest MULTIPLEHP kernel on $A10$, which is MULTIPLEHP-R-(64, 8) (NDRange kernel part), the execution latency of it (without candidates detection) on $R7$ is 19.7 ms, and it is 3.7× faster than achieved on $A10$. After combining with the candidate detection, which is a single work-item kernel, the performance drops as N_{cand} increases. When $N_{\text{cand}} = 1$, the execution latency is 46.8 ms. However, when N_{cand} is increased to 200, the latency increases to 10 s.

Based on the above, an $R7$ is over 3.7× faster than an $A10$ in executing the same NDRange kernels. Regarding the single work-item kernels, GPU implementations cannot compete with FPGAs, being tens to hundreds of times slower than FPGAs.

D. Energy Dissipation and Power Consumption

The execution latency is a significant performance criterion for the harmonic-summing module. However, in the context of the PSS in SKA1-MID, there will be over 2000 beams that need to be computed in parallel, which is constantly done for many years. As a result, the power consumption is another essential criterion that we investigate in this section.

We calculate the difference between the system power consumption P_{idle}, including the acceleration device, in idle status and the power consumption P_{running} when the system is executing the kernel. To make sure the value of P_{running} is stable, each kernel is launched hundreds of times using a loop, which takes several minutes.

The power consumption is measured using a plug-in power meter. For the FPGA measurements, the calculated power consumption is the value of using three $A10$ cards in one host. The power consumption and energy dissipation of executing different kernels are given in Table VII. The energy cost is the dissipation of processing the input half FOP, and the energy saving ratio is compared with the $I7-1C$. Since the execution latencies of MULTIPLEHP kernels with the single work-item kernel (in Section VI-C2) on GPU are over 10 times larger than those on FPGA, the MULTIPLEHP kernels with a single work-item part are not compared with GPU.

Although the execution latency on $R7$ is faster than that of $A10$, the energy dissipation of $R7$ is over 1.8× higher than that of three $A10$s. An interesting observation from Table VII is that the power consumption of kernel SINGLEHP-(M, R) and MULTIPLEHP-R on $A10$ is significantly higher than other MULTIPLEHP kernels on $A10$. The main reason is that the used GMB of SINGLEHP-(M, R) and MULTIPLEHP-R is optimized and much higher than other kernels. Streaming data between off-chip memory and FPGA make the power consumption of a kernel up to three times higher than that of other MULTIPLEHP kernels.

TABLE VII
POWER CONSUMPTION AND ENERGY DISSIPATION OF FPGA, GPU, AND CPU PLATFORMS (WITHOUT CANDIDATE DETECTION)

Kernel-Setting (Device)	Power ($watts$)	Energy ($Joules$)	Saving ratio
SINGLEHP-(M, R) $(A10 \times 3)$	23	3.36	19.9
SINGLEHP-$(M,)$ $(R7)$	65	8.9	7.5
SINGLEHP$(I7-4C)$	43	47.85	1.4
SINGLEHP$(I7-1C)$	16	66.8	1
Naïve MULTIPLEHP$(A10 \times 3)$	7	0.91	73.4
MULTIPLEHP-H $(A10 \times 3)$	10	1.75	38.2
MULTIPLEHP-N $(A10 \times 3)$	14	1.11	60.0
MULTIPLEHP-R $(R7)$	49	0.965	69.2
MULTIPLEHP-R $(A10 \times 3)$	22	0.526	**127.0**

In summary, it can be found that a single $R7$ needs over 2× more power than three $A10$ cards. Regarding the energy dissipation, the cost of $R7$ is up to 2.6× higher than three $A10$ cards in executing the same kernels while providing similar performance.

VII. CONCLUSION

In this paper, we investigated FPGA designs of one module of the SKA PSS called harmonic summing. OpenCL was chosen to implement the proposed designs, and FPGAs and GPU were employed for evaluation. Two approaches of harmonic summing were studied: 1) store intermediate data in off-chip memory and 2) process the input signals directly without storing intermediate data. For the second approach, since a naïve implementation does not provide good performance, two approaches of preloading data were proposed and evaluated: 1) preloading points that are touched most and 2) preloading all necessary points that are used to generate a chunk of output points. For the necessary points approaches, the reorder of input signals is investigated as well.

The extensive experimental evaluation demonstrated that kernels with intermediate data storage perform worse than kernels without storing intermediate data in both execution latency and power consumption. A single FPGA can achieve 9.5× speedup over single-core CPU using the general SINGLEHP method. By using multiple FPGAs, the NDRange MULTIPLEHP kernels perform significantly better than a single GPU in power consumption while only being slightly slower regarding execution latency. To process the same amount of data using the same OpenCL kernel, GPU costs up to 2.6× more energy than multiple FPGAs. This paper shows that FPGA devices are a good solution for the SKA project for the processing parts of the pulsar search pipeline, and techniques discussed here can be transferred to other memory-intensive applications with irregular accesses.

ACKNOWLEDGMENT

The authors would like to thank Time-Domain Team, a collaboration between Manchester and Oxford Universities, and MPIfR Bonn and the work benefited from their collaboration. They would also like to thank P. Dobias and E. Casseau from IRISA, University of Rennes 1.

REFERENCES

[1] A. Canis *et al.*, "Legup: High-level synthesis for FPGA-based processor/accelerator systems," in *Proc. 19th ACM/SIGDA Int. Symp. Field Program. Gate Arrays*, 2011, pp. 33–36.

[2] C. Carilli and S. Rawlings. (2004). "Science with the square kilometer array: Motivation, key science projects, standards and assumptions." [Online]. Available: https://arxiv.org/abs/astro-ph/0409274

[3] L. Chen, Z. Hu, J. Lin, and G. R. Gao, "Optimizing the fast Fourier transform on a multi-core architecture," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, Mar. 2007, pp. 1–8.

[4] T. S. Czajkowski *et al.*, "From opencl to high-performance hardware on FPGAS," in *Proc. 22nd Int. Conf. Field Program. Logic Appl. (FPL)*, Aug. 2012, pp. 531–534.

[5] L. De Souza, J. D. Bunton, D. Campbell-Wilson, R. J. Cappallo, and B. Kincaid, "A radio astronomy correlator optimized for the Xilinx Virtex-4 SX FPGA," in *Proc. Int. Conf. Field Program. Logic Appl.*, Aug. 2007, pp. 62–67.

[6] P. E. Dewdney, P. J. Hall, R. T. Schilizzi, and T. J. L. W. Lazio, "The square kilometre array," *Proc. IEEE*, vol. 97, no. 8, pp. 1482–1496, Aug. 2009.

[7] *The Opencl Specification, Version 1.0. 29*, Khronos OpenCL Working Group, Beaverton, OR, USA, Dec. 2008.

[8] A. Hiba, Z. Nagy, and M. Ruszinko, "Memory access optimization for computations on unstructured meshes," in *Proc. 13th Int. Workshop Cellular Nanosc. Netw. Appl.*, 2012, pp. 1–5.

[9] A. Jain and C. Lin, "Linearizing irregular memory accesses for improved correlated prefetching," in *Proc. 46th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2013, pp. 247–259.

[10] B. Jang, D. Schaa, P. Mistry, and D. Kaeli, "Exploiting memory access patterns to improve memory performance in data-parallel architectures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 1, pp. 105–118, Jan. 2011.

[11] J. Mellor-Crummey, D. Whalley, and K. Kennedy, "Improving memory hierarchy performance for irregular applications using data and computation reorderings, "*Int. J. Parallel Program.*, vol. 29, no. 3, pp. 217–247, 2001.

[12] A. Parsons *et al.* (2009). "Digital instrumentation for the radio astronomy community." [Online]. Available: https://arxiv.org/abs/0904.1181

[13] K. Pavel and S. David, "Algorithms for efficient computation of convolution," in *Proc. Design Archit. Digit. Signal Process.*, 2013, pp. 191–195.

[14] A. Putnam *et al.*, "A reconfigurable fabric for accelerating large-scale datacenter services," in *Proc. ACM/IEEE 41st Int. Symp. Comput. Archit. (ISCA)*, Jun. 2014, pp. 13–24.

[15] S. M. Ransom, S. S. Eikenberry, and J. Middleditch, "Fourier techniques for very long astrophysical time-series analysis," *Astronomical J.*, vol. 124, no. 3, p. 1788, 2002.

[16] M. A. Sanchez, M. Garrido, M. López-Vallejo, J. Grajal, and C. López-Barrio, "Digital channelised receivers on FPGAs platforms," in *Proc. IEEE Int. Radar Conf.*, May 2005, pp. 816–821.

[17] S. Sridharan, P. Durante, C. Faerber, and N. Neufeld, "Accelerating particle identification for high-speed data-filtering using OpenCL on FPGAs and other architectures," in *Proc. 26th Int. Conf. Field Program. Logic Appl. (FPL)*, Sep. 2016, pp. 1–7.

[18] N. Tarafdar, T. Lin, E. Fukuda, H. Bannazadeh, A. Leon-Garcia, and P. Chow, "Enabling flexible network FPGA clusters in a heterogeneous cloud data center," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, 2017, pp. 237–246.

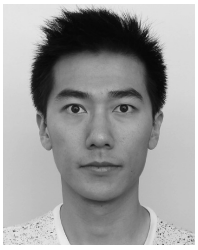
[19] H. Wang, M. Zhang, P. Thiagaraj, and O. Sinnen, "FPGA-based acceleration of FDAS module using OpenCL," in *Proc. Int. Conf. Field-Program. Technol. (FPT)*, Dec. 2016, pp. 53–60.

[20] X. Wang, L. Huang, Y. Zhu, Y. Zhou, H. Peng, and H. Xiong, "Addressing memory wall problem of graph computation in reconfigurable system," in *Proc. IEEE 17th Int. Conf. High Perform. Comput. Commun. (HPCC)*, Aug. 2015, pp. 302–307.

[21] M. Weinhardt and W. Luk, "Memory access optimization and RAM inference for pipeline vectorization," in *Proc. Int. Workshop Field Program. Logic Appl. (FPL)*, 1999, pp. 61–70.

[22] M. Weinhaudt and W. Luk, "Memory access optimisation for reconfigurable systems," *IEE Proc.-Comput. Digit. Techn.*, vol. 148, no. 3, pp. 105–112, May 2001.

[23] H.-J. Yang, K. Fleming, M. Adler, and J. Emer, "Optimizing under abstraction: Using prefetching to improve FPGA performance," in *Proc. 23rd Int. Conf. Field Program. Log. Appl. (FPL)*, Sep. 2013, pp. 1–8.



Haomiao Wang (S'16) received the B.Sc. degree from the Harbin University of Science and Technology, Harbin, China, in 2012 and the M.Sc. degree from the Harbin Institute of Technology, Harbin, in 2014. He is currently working toward the Ph.D. degree in computer system engineering at The University of Auckland, Auckland, New Zealand, under the supervision of Dr. O. Sinnen.

His current research interests include high-performance computing, optimization, and high-level synthesis.

Prabu Thiagaraj received the B.Sc. degree in computer engineering from Bharathiar University, Chennai, India, and the M.S. and Ph.D. degrees in radio astronomy instrumentation from the Indian Institute of Science, Bengaluru, India.

From 2014 to 2017, he was at JBCA, University of Manchester, Manchester, U.K., where he developed field programmable gate array-based acceleration prototypes along with the SKA Time-Domain Team for pulsar search with the square kilometer array. He is currently at the Raman Research Institute, Bengaluru. His current interests include digital signal processing with field programmable gate array.

Oliver Sinnen received the B.Sc. degree in electrical and computer engineering from RWTH Aachen University, Aachen, Germany, and the Ph.D. degree from the Instituto Superior Técnico, University of Lisbon, Lisbon, Portugal, in 2003.

Since 2004, he has been a Senior Lecturer at the Department of Electrical and Computer Engineering, The University of Auckland, Auckland, New Zealand, where he leads the Parallel and Reconfigurable Computing Laboratory.