Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Reduced-Precision Acceleration of Radio-Astronomical Imaging on Reconfigurable Hardware

STEFANO CORDA[1][2] (Student, IEEE), BRAM VEENBOER[3], AHSAN JAVED AWAN[4], JOHN ROMEIN[3], ROEL JORDANS[1], AKASH KUMAR[2] (Senior Member, IEEE), ALBERT-JAN BOONSTRA[3] AND, HENK CORPORAAL[1].

[1]Electronic Systems Group, Eindhoven University of Technology, 5600 MB Eindhoven, The Netherlands
[2]Chair of Processor Design, CFAED, Technische Universität (TU) Dresden, 01062 Dresden, Germany
[3]Netherlands Institute for Radio Astronomy (ASTRON), 7991 PD Dwingeloo, The Netherlands
[4]Department of Cloud System and Platforms Ericsson Research, 16480 Stockholm, Sweden

Corresponding author: Stefano Corda (e-mail: s.corda@tue.nl, stefano.corda@mailbox.tu-dresden.de).

The European Commission partially supported this work in the framework of the Horizon 2020 program for the project "Near-Memory Computing (NeMeCo)" under Marie Sklodowska-Curie Innovative Training Networks European Industrial Doctorate (Project ID: 676240). This work was also partially supported by the OPRECOMP Summer of Code 2020, the HiPEAC Collaboration Grant, and the Xilinx Donation Program.

ABSTRACT Radio telescopes produce large volumes of data that need to be processed to obtain high-resolution sky images. This is a complex task that requires computing systems that provide both high performance and high energy efficiency. Hardware accelerators such as GPUs (Graphics Processing Units) and FPGAs (Field Programmable Gate Arrays) can provide these two features and are thus an appealing option for this application. Most HPC (High-Performance Computing) systems operate in double precision (64-bit) or in single precision (32-bit), and radio-astronomical imaging is no exception. With reduced precision computing, smaller data types (e.g., 16-bit) are used to improve energy efficiency and throughput performance in noise-tolerant applications. We demonstrate that reduced precision can also be used to produce high-quality sky images. To this end, we analyze the gridding component (Image-Domain Gridding) of the widely-used WSClean imaging application. Gridding is typically one of the most time-consuming steps in the imaging process and, therefore, an excellent candidate for acceleration. We identify the minimum required exponent and mantissa bits for a custom floating-point data type. Then, we propose the first custom floating-point accelerator on a Xilinx Alveo U50 FPGA using High-Level Synthesis. Our reduced-precision implementation improves the throughput and energy efficiency of respectively 1.84x and 2.03x compared to the single-precision floating-point baseline on the same FPGA. Our solution is also 2.12x faster and 3.46x more energy-efficient than an Intel i9 9900k CPU (Central Processing Unit) and manages to keep up in throughput with an AMD RX 550 GPU.

INDEX TERMS Accelerator architectures, Approximation methods, Astronomy, Central Processing Unit, Field programmable gate arrays, Graphics Processing Units, High level synthesis, High performance computing, Reconfigurable architectures, Scientific computing

I. INTRODUCTION

THE future generation of radio telescopes, such as the Square Kilometre Array (SKA) [1], will have to process a massive quantity of data (in the order of TeraBytes per second) using high-performance computing systems (in the order of Exaflops per second) [2] with high energy efficiency [3]. The demanding data and computation requirements are mainly caused by the high-resolution images that must be processed to discover new objects in the sky such as stars, supernovas, galaxies, etc. [4].

The most dominant compute kernels of the radio-astronomical imaging pipeline are the gridding, and degridding algorithms [5]. These kernels can be executed highly efficiently in single-precision floating-point accuracy using

GPUs. A GPU accelerated imaging solution is shown to meet the computing power and energy efficiency requirements of SKA [6]. FPGAs (Field Programmable Gate Arrays) are well known to be energy-efficient platforms for fixed-point computation [7, 8]. However, recent FPGA platforms can also be efficiently used for floating-point computations, e.g., Intel Arria 10 and Stratix 10 [9]. Furthermore, High-Level Synthesis (HLS) toolchains enable FPGAs to be programmed much more easily compared to using a Hardware Description Language (HDL). This makes FPGAs an attractive accelerator platform [10]. However, the prior art shows that FPGAs are less energy-efficient than GPUs for the radio astronomy application domain [11].
Reduced precision computing is a technique where smaller data types are used to reduce area usage, execution time, and power consumption within noise-tolerant applications without losing information [12]. It has been widely applied in different application domains, especially, in deep learning applications [13, 14]. Existing studies propose the use of reduced precision also for the deconvolution kernel [15], apply mixed precision to other steps of the radio-astronomical imaging acquisition pipeline, e.g., correlator [16], or other radio-astronomy domains, e.g., computation of tomographic reconstructors [17]. However, the works mentioned above employ standard data types supported by CPUs and GPUs, such as double-, single- and half-precision floating-point, and do not evaluate custom data types. While Intel FPGAs such as Arria 10 and Stratix have native support for single-precision floating-point operations (2 flops per DSP per cycle), Xilinx FPGAs do not. Xilinx Ultrascale FPGAs, and the Alveo U50 in particular, have many DSPs though [18], and it is tempting to employ these to implement reduced-precision arithmetic.

This paper evaluates the noise tolerance in a state-of-the-art radio-astronomical imaging algorithm to reduce precision data types, highlighting possible optimization opportunities. We evaluate the performance of the radio-astronomical imaging algorithm on a Xilinx Alveo U50 FPGA employing High-Level Synthesis and traditional floating-point data types. Then, we demonstrate how custom floating-point can be efficiently employed to improve performance while maintaining high output quality. To the best of our knowledge, this is the first work that focuses on assessing the applicability of reduced precision for radio-astronomical imaging.

The main contributions of this work are:

- An in-depth analysis (*Section V*) to determine the precision requirements of radio-astronomical Image-Domain Gridding [19], included in the state-of-the-art imager WSClean [20]. It highlights that standard data types such as half-precision and brain floating-point do not meet them. Furthermore, we show how the analysis can be carried out faster (4x in our case).
- A custom floating-point gridding accelerator for radio-astronomical imaging on reconfigurable hardware. To the best of our knowledge, this is the first FPGA implementation applying reduced precision for radio astronomical imaging with negligible quality loss (*Section VI*). Furthermore, we determine several guidelines for accelerator design on Xilinx FPGAs by using Xilinx Vitis with regard to the state of the art.
- An in-depth performance evaluation (*Section VII*) of our accelerator prototypes and of state-of-the-art architectures with similar features such as peak performance, thermal design power and lithography technology. The Xilinx Alveo U50 outperforms an Intel i9 9900k CPU in terms of energy efficiency. Moreover, our best accelerator prototype outperforms its single-precision baseline and keeps up in throughput with an AMD RX 550 GPU.

The paper is structured as follows: *Section II* explains the background information regarding radio-astronomical interferometry and imaging. Then, in *Section III* we discuss background information about reduced precision and data types. *Section IV* shows our methodology. In *Section V* we discuss the analysis results of the WSClean imager. *Section VI* reports details on how we design the accelerator architecture. It is evaluated in *Section VII*, where we also present the lessons learned in this work. Finally, we describe the related work in *Section VIII* and we conclude the paper in *Section IX*.

II. RADIO-ASTRONOMICAL IMAGING

A radio telescope detects electromagnetic waves that originate from radio sources in the universe. The signals are used, among other things, to construct a map of the sky containing the positions, intensity, and polarization of the sources. Radio telescopes such as LOFAR [21] and SKA1-Low [22] are comprised of (small) dipole antennas that measure two orthogonal polarizations of the radio sources, while other radio telescopes (such as the VLA [23], MeerKAT [24] and SKA1-Mid [25]) are based on an array of dishes. As shown in *Figure 1*, a station consists of multiple antennas for which the signals for every distinct frequency channel are combined. The signals of a pair of stations (a baseline) are multiplied and integrated (correlation ②) for a short period of time (in the order of seconds), thus producing a single visibility (a 2x2 matrix). The data that the telescope produces (the `visibilities`) is thus a three-dimensional matrix (with indices number of baselines, number of frequency channels, 4). The relation between visibilities and sky brightness is given by a measurement equation, see [26] for complete details.

The visibilities are first calibrated (③) and next used to reconstruct the sky brightness in the observation direction using an imaging step (④) [27].

This work focuses on ④. The imaging step (see *Figure 2*) starts with an empty sky model and it consists of an iterative process: 1) the `inversion` step is used to produce a `dirty image`; 2) one or more bright sources are detected in this image by a deconvolution algorithm such as `CLEAN` (see also Section II-B); 3) a `model image` is created, which contains all of the sources in the sky model; 4) visibilities corresponding to this model image are `predicted`; 5) subtracting the predicted visibilities from the measured (and calibrated) visibilities yields `residual visibilities`. This process

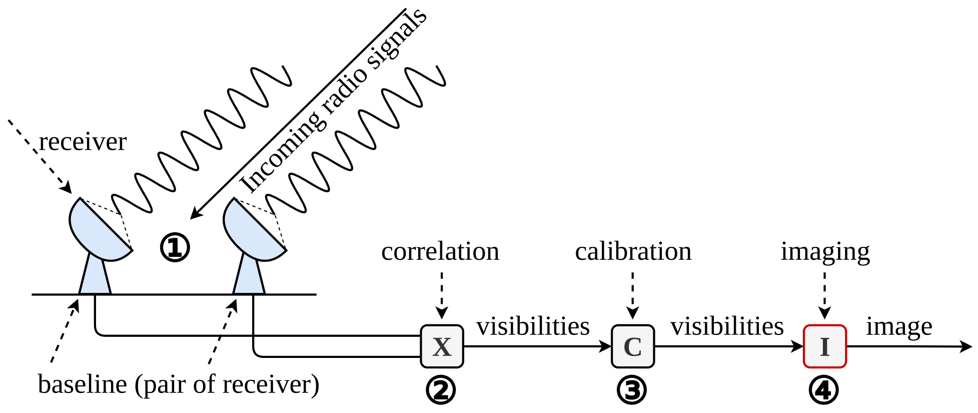


FIGURE 1: Radio astronomy image acquisition: the incoming radio signals are digitized and then correlated and calibrated before the imaging step is executed. We focus on the imaging step, which is highlighted in red.

subtracts strong sources from the measurements, which mask the more interesting weak sources. This step is repeated until the sky model converges. Finally, the sky model is used to create the sky image.

The inversion and prediction steps comprise of 2D FFT and a `gridding` or `degridding` step. The gridding and degridding steps are typically the most compute-intensive image processing steps. To attain high-quality sky images, they need to correct for Direction-Independent Effects (the curvature of the earth, `W-Term` correction) and Direction-Dependent Effects (such as ionospheric effects, `A-Term` correction). The W-Term can be corrected by applying a convolution kernel to every visibility. The required convolution kernel could be huge depending on parameters such as the field-of-view and distance between receivers. A-Term correction requires these convolution kernels to be different for every receiver and change over time according to changes in the Direction-Independent effects. These properties make imaging with correction for W-Terms and A-Terms particularly challenging.

The processing facilities that the SKA consortium is planning to build for the low and mid frequencies consist of large Science Data Processors. Each of them has a peak performance in the order of 6.50 PFLOP/s and a thermal design power in the order of 125 MW [6].

A. IMAGE-DOMAIN GRIDDING

Image-Domaing Gridding (IDG) is a state-of-the-art algorithm for both gridding and degridding [19]. IDG performs both W-correction and A-correction in the image domain, avoiding large convolutions functions. The algorithm performs gridding and degridding using `subgrids`, which represent low-resolution sky images for a subset of visibilities. This approach exposes a lot of parallelism (subgrids can be processed in parallel), which makes it highly efficient on parallel hardware such as GPUs [6]. In IDG, gridding comprises three steps: 1) visibilities are gridded onto subgrids; 2) subgrids are Fourier transformed; 3) subgrids are added to

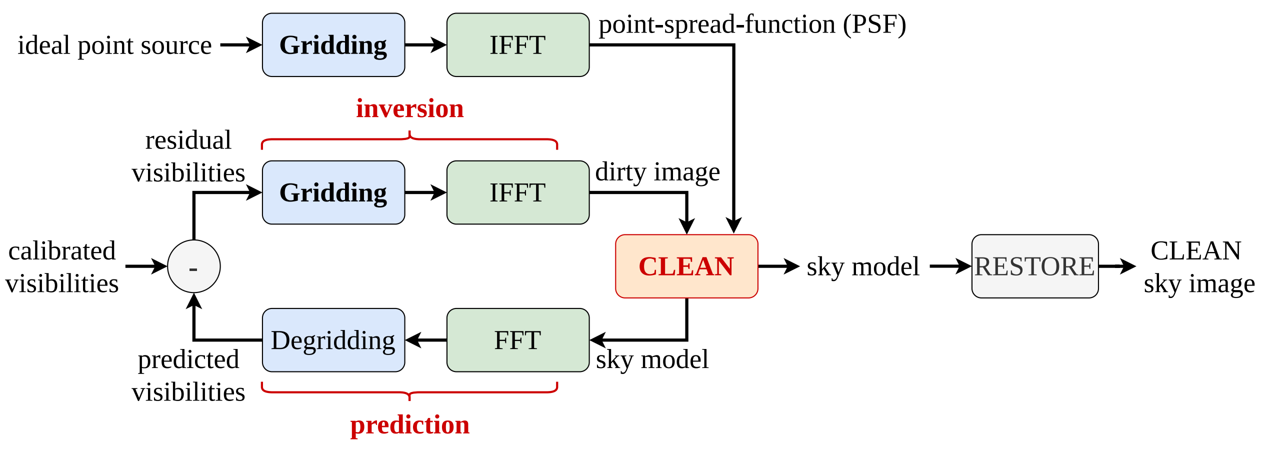


FIGURE 2: High-level schematic of the radio-astronomical imaging step. The three main phases, inversion, prediction, and deconvolution (or CLEAN) are highlighted in red. The critical kernel, the gridding, is included in the inversion step (it is highlighted in bold). We included the point-spread-function (PSF) computation, an inversion step with an all-ones matrix of visibilities used during the CLEAN.

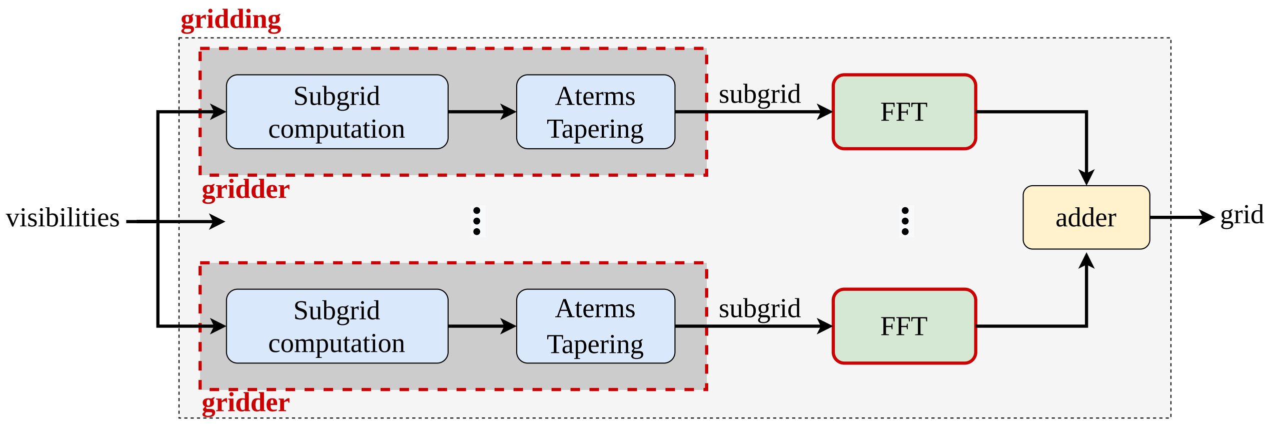


FIGURE 3: Gridding high-level representation. It consists of multiple gridder computations (subgrid computation and tapering) and FFTs that process the input visibilities into subgrids. Then the subgrids are processed by the adder to obtain a grid. The gridder and FFT, red boxes, are the focus of this and the related work [11].

the larger final grid. IDG degridding comprises these steps in reverse order. Refer to [6, 19] for all the details on this algorithm and a formal derivation.

Image-Domain Gridding performs much better [28] than classical gridding/degridding algorithms, such as W-projection [29] or AW-projection [30]. It also employs W-terms to solve artifacts around sources away from the phase center in wide-field imaging. Moreover, IDG image quality is higher than W-projection because IDG, like AW-projection, corrects for DDEs (direction-dependent effects, also called the A-terms), but the computational costs for such DDE corrections are much lower for IDG than for AW-projection [6]. IDG also has higher per-visibility accuracy compared to the other algorithms [19].

B. DECONVOLUTION

The objective of a CLEAN (or deconvolution) algorithm is to detect sky sources by iteratively finding the brightest peaks in a dirty image and fitting a sky model. In *Figure 4* we show an example of a dirty image and the corresponding image after a deconvolution algorithm has been applied. The CLEAN image shows lower noise compared to the dirty image.

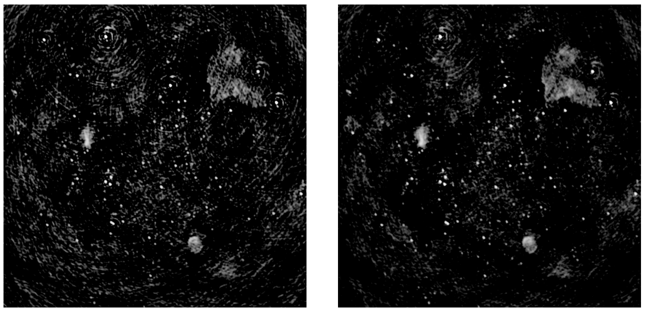


FIGURE 4: Comparison between dirty image (left) and CLEAN image (right) applying Cotton-Schwab algorithm. The CLEAN image has reduced noise, e.g. rotation lines around the image(visible especially at the corners).

Many deconvolution (CLEAN) algorithms have been proposed in the literature. The simplest one is the so-called `Högbom` [31]. After the dirty image is generated from the imaging step (gridding and IFFT), the Högbom CLEAN tries to remove the noise in the image. This is done iteratively, looking for the maximum value in the image. Then, the algorithm subtracts the Point Spread Function (PSF), which is a function dependent on the telescope used. It is computed like the dirty image using as input a Visibility array with values equal to 1, multiplied by a gain factor. After a certain number of iterations or when a certain threshold (e.g., 3σ of the standard deviation) is reached the algorithm stops. This algorithm does not include the prediction step.

The `Clark` CLEAN [32], which is an improvement of the previously described algorithm, adds a feedback loop and tries to remove alias errors. It is possible to distinguish between `major` and `minor iterations` in this case. The minor iterations are represented by the peak search, similar to Högbom (the CLEAN box in *Figure 2*). Then, the model image is Fourier transformed and subtracted from the dirty image. This is the so-called major iteration.

The `Cotton-Schwab` [33] and `Multiscale` [34, 35] are the most employed and modern ones, they have in common the prediction phase, thus including the degridding algorithm. Here the subtraction is done at the visibilities level reducing the pixelation error. More precisely, for these algorithms, a major iteration consists of an entire iteration to transform the data from the visibility domain to the image domain (inversion). The major iterations are executed until a threshold is reached, e.g., until 80% of the flux (which is a power density measure) is removed from the dirty image during the minor iterations.

The Multiscale algorithm operates on a set of residual images obtained by convolving the dirty image with different scale sizes. The peak subtraction step is performed on all the scaled imaged, and only the subtracted components are stored in the CLEAN component table. After being scaled, positioned, and convolved, the final image is obtained by adding the components. It decreases the effects of the pedestal of uncleaned flux and strong sidelobes present in the dirty beam (or Point-Spread Function), which are referred to as "clean bow", around bright resolved structures and has better convergence properties [36].

III. REDUCED PRECISION

Reduced precision is a software and hardware technique employing smaller data types to improve performance. It can be applied at the software level if the architecture supports reduced data types, e.g. half precision in modern GPUs. However, a custom architecture should be designed on FPGA (or ASIC) hardware when a non-supported data type is needed. The main benefits of this technique are the reduced processing elements (PEs) size, which leads to higher throughput, and reduced memory requirements, which increase the effective memory bandwidth. Key factors for reduced precision are data types that are described in *III-A*. In *III-B* we briefly introduce reduced precision and the advent of appealing tools for exploring custom data types in software and hardware.

A. DATA TYPES

Standard architectures, such as CPUs and GPUs, typically support single-precision and double-precision floating-point applications that perform scientific computations. Commonly, single precision is the most widespread data type since the major part of systems supports it. Indeed, even if double precision is supported on most GPUs, they often do not have dedicated units for double-precision computations except for high-end GPUs like Tesla V100 or Ampere A100 [37], thus reducing performance [38]. Radio-astronomical imaging runs precisely enough using single-precision floating-point. For this reason, we focus on data types up to 32 bits. We present in *Figure 5* the most commonly used data types today. In particular, we recognize two main categories: standard data types where the bit length is fixed and custom data types where the data length is defined at design time, compile time, or runtime.

Except for posit and fixed points, the other data types are floating-point representations that can be expressed by *Equation 1*. More precisely, the `exponent` and the `mantissa` bits are responsible for respectively the dynamic range and the precision of the data type. The dynamic range limits the smallest and largest number representable, while the precision is the represented number's resolution (number of digits).

$$(-1)^{sign} * mantissa * 2^{exponent} \tag{1}$$

In *Table 1* we present the mantissa and exponent sizes of the main standard data types. **Single-Precision** [39] (or `binary32` [40]) and **Double-Precision** (or `binary64`) **Floating Point** are commonly supported on GPUs and have been added to the **IEEE 754** standard. With the advent of deep learning applications and their noise tolerance and need for reduced length data types, half precision usually

TABLE 1: Standard floating-point formats.

Name	Exponent	Mantissa
Single-precision	8	23
Half-precision	5	10
NVIDIA-Tensor	8	10
Brain	8	7

offers 2x the performance of single precision in applications that can tolerate the noise introduced by the lower dynamic range and less precision. For the same reason, NVIDIA presents the new `Tensor Float-32` with the release of the NVIDIA A100, the new AI and HPC flagship GPU. This format has the same dynamic range of the **binary32** but reduced precision, which is claimed to be sufficient for most of today's AI applications. **Brain Floating Point** [41] offers a further precision reduction while keeping the same dynamic range offered by single precision. This is especially employed by Intel [42] and Google [43].

Apart from the standard data types mentioned above, other data types are not available in mainstream architecture or do not have a pre-defined number of bits. These data types are described below:

Fixed Point [44]: this format is represented by *Eq. 2*. A real number is represented by two numbers, one for the integer part and one for the fractional part. Compared to floating point, it has a smaller dynamic range since there is no exponent. Still, the hardware implementation is easier since it considers two numbers (integer and fractional), and it is usually employed on custom accelerators, and FPGA [45, 46].

$$(-1)^{sign} * integer.fractional \tag{2}$$

Custom Floating Point [47]: a custom floating-point is super-set of the floating-point above mentioned. It consists of all the possible floating-point format combinations. They are typically used in embedded systems, where the numeric representation is customized for the specific application by selecting specific bit lengths for the mantissa and exponent fields.

Posit [48]: is a numeric representation that has been proposed as a substitute for floating-point data types. Usually, posit has a higher dynamic range than the floating-point with the same bit length (see *Figure 6*). Moreover, posits have a tapered decimal accuracy (see *Equation 3*, where x and y are two numbers with same sign) which means that the decimal accuracy reported in *Figure 6* is roughly symmetrical, and the highest precision is achieved for numbers near 1 (the horizontal axis is reporting the base-2 logarithm of the numbers).

$$decimal\_accuracy = -log_{10}(|log_{10}(x/y)|) \tag{3}$$

This feature differs from floating points that have the constant accuracy across the dynamic range, except for small

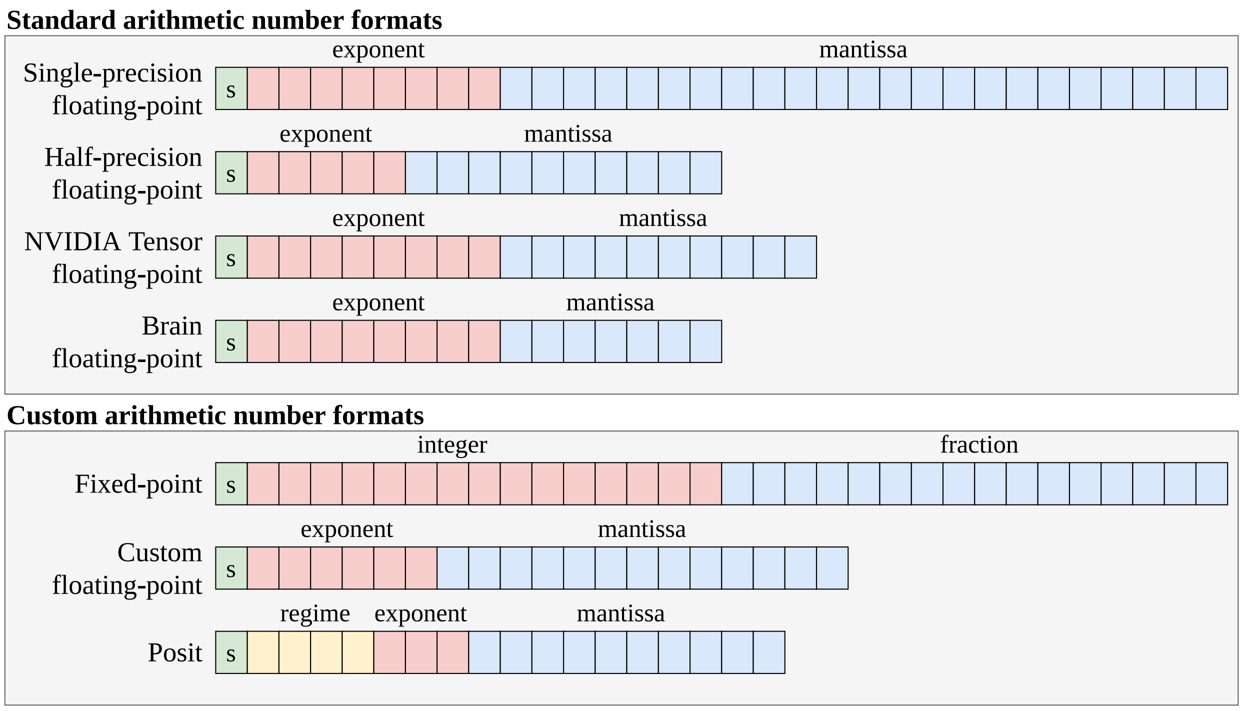

Standard arithmetic number formats
Single-precision floating-point
Half-precision floating-point
NVIDIA Tensor floating-point
Brain floating-point
s
exponent
mantissa
Custom arithmetic number formats
Fixed-point
integer
fraction
Custom floating-point
Posit
regime

FIGURE 5: Data-types overview. Standard arithmetic number formats such as single-precision and half-precision floating-point are data types usually supported by modern CPUs and GPUs. Custom arithmetic number formats comprise some of the main data types employed in research and are often deployed on FPGAs. The custom formats reported are examples, and the number of bits may differ.

numbers (left side), and the accuracy suddenly falls off a cliff (right side) to accommodate all the NaN (not a number values). However, we are not considering posits as a possible datatype candidate since it is usually more expensive compared to floating-point for multiplication and addition operations [49]. For more details refer to [48].

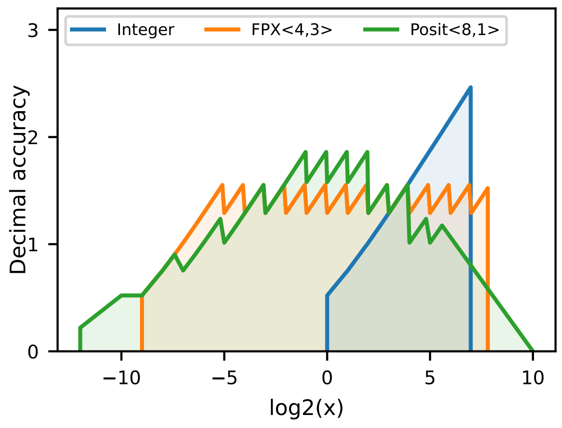


FIGURE 6: The figure is inspired from [48]. Decimal accuracy for 1) 8-bit signed integers; 2) a custom floating points with 4-bit exponent and 3-bit mantissa; 3) 8-bit posits with 1-bit exponent [50, 51]. Posits have a larger dynamic range compared to floating points and integers (very small). Integers have higher decimal accuracy for larger numbers. While posits have a tapered decimal accuracy, floating points have approximately a constant accuracy across the dynamic range.

We employ custom floating point for both analysis and hardware design, which comprises single precision, half precision, NVIDIA Tensor, and brain floating point. Based on the analysis in *Section V* we evaluate the precision requirements of the target application and discuss the excluded data types.

B. REDUCED PRECISION TOOLS

Reduced precision is a branch of approximate computing, which usually consists of either reducing the bit size of standard data types or employing more efficient data types [12]. Recently, there has been the rise of automated/assisted precision tuning tools and emulation libraries to help and improve the selection of custom data types inside applications. However, the major part of the above-mentioned works support only standard data types [52–54] or fixed point [55].

Flegar et al. [47] designed `FloatX`, a C++ template library capable of emulating `custom` floating point, which we employ in our analysis. FloatX also has a reduced execution time overhead compared to the previous library since it employs hardware-supported floating-point types as backend.

Recently, High-Level Synthesis libraries for supporting custom floating-point precision have been researched. These libraries are easy to use and portable compared to RTL approaches [56–58].

DiCecco et al. [56] propose a custom-precision floating-point library (`CPFP` [59]) for High-Level Synthesis, and they evaluate it on a small convolution neural network. While

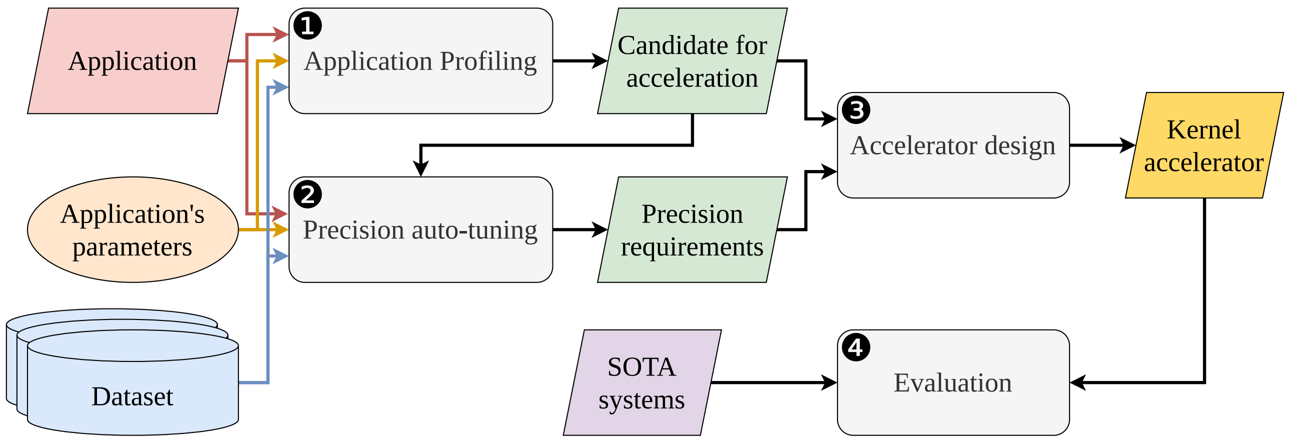


FIGURE 7: High-level overview of the employed methodology. The application's bottleneck is detected by applying application profiling. Then a precision auto-tuning technique defines the precision requirements for the accelerator, which is designed and deployed. Finally, the accelerator performance is assessed with state-of-the-art (SOTA) architectures.

the custom floating-point IP employs fewer resources than single precision, the FPGA design has a lower throughput than the CPU. Thomas [57, 58] proposed a more efficient (see *Appendix A*) templatized floating-point library for high-level-synthesis (`THLS` [60]). This library, which we employ in this work, is also templatized and eventually supports heterogeneous custom floating-point operations. The proposed solution has similar resource consumption when comparing standard data types and notable resource reduction when employing reduced-precision data types. The two approaches mentioned above are easier and portable for FPGA development than employing custom floating-point IPs similar to FloPoCo [61], which need to be used as black-boxes.

IV. METHODOLOGY

We present the methodology employed in *Figure 7*. We first ❶ profile the radio-astronomical imager to determine the most time consuming and thus critical kernels. We perform this analysis on different datasets employing the application's parameter described in *IV-A*. Then, ❷ we evaluate the required minimum precision requirements for the selected kernel using an auto-tuning script based on binary search (see *IV-B*). Through emulation, it identifies the minimum bit sizes for both the exponent and mantissa for custom floating-point data types. Afterwards, ❸ we perform an accelerator design phase to individuate the best design optimization for the selected kernel and the precision requirements(see *IV-C*). Finally, ❹ we assess the accelerator performance with state-of-the-art systems (see *IV-D*).

A. APPLICATION PROFILING

We profile the most widely used and state-of-the-art radio-astronomical imager, WSClean [65], which also includes the state-of-the-art gridding and degridding algorithm (Image-Domain Gridding) [64], by evaluating the execution time breakdown. We report the software version used in this work in *Table 2*.

TABLE 2: Software versions employed. We report the checksum of the commit of the master branch we used for WSClean and IDG. The other packages are WSClean's dependencies.

Software	Version
boost	1.68
OpenBLAS	3.9
python	3.8
wcslib	6.3
cfitsio	3.450
casacore [62]	3.3.0
dysco [63]	1.2
IDG [64]	master 011dfb18
WSClean [65]	master 2680c6a

We employ the `LOFARSCHOOL` dataset, which is usually employed as a test case for practical examples and contains real sky observations [66]. This dataset contains 16 observations and around 30 subbands per observation, available in the LOFAR Long Term Archive (LTA) [67]. We select 14 observations with similar observation parameters such as integration time, observation duration, frequencies, etc. (see *Table 3*), and we select the 10th subbands from every dataset; therefore, all the datasets have the same central frequency.

TABLE 3: Datasets observation parameters.

Name	Description
Central Frequency	120.1172-125.7812 MHz
Channels per subband	4
Channel width	48 828.125 Hz
Declination	50.9410-54.8590
Duration	7199 s
Integration interval	2.002 78 s
Right Ascension	311.2500-318.7500

The WSClean parameters that we use are listed in *Table 4*.

The sky images are plotted using Kstars FITS Viewer [68] setting the following parameters: shadows 0.0080, midtones 0.0625, and highlights 0.6009.

B. PRECISION AUTO-TUNING

To determine the application precision requirements, we employ binary search over the number of mantissa bits like [54], and over the number of exponent bits for custom precision floating-point data, as the execution time overhead (about 5-10x with respect to the optimized single-precision code) of emulation of custom data types in software makes evaluating the entire search space unpractical. As shown in *Figure 8* we manually instrument the application code to support the software emulation of `custom` data types. To emulate custom floating-point we employ a template header C++ libraries: FloatX [69] for floating-point [47]. We do not include fixed-point numbers since they would result in very long fixed-point representations based on the analysis results, which shows that a large dynamic range is required. This binary search algorithm first evaluates the mantissa size and then the exponent size. Since the application usually runs in single precision, the starting mantissa size of 23 bits is divided by two and evaluated. The process consists of first determining the size to evaluate and then updating the headers containing the mantissa and exponent sizes. Then, the application is compiled and run. Finally, the algorithm evaluates the output precision employing the Structural Similarity Index Measure (SSIM) [70] metric.

We select SSIM [70] as the assessment metric since, differently from Peak Signal to Noise Ration (PSNR), we can measure the perceived image quality and thus evaluate how two images are similar. SSIM computation is more complex than PSNR and it is shown in *Equation 4*.

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2\mu_y^2 + C_1)(\sigma_x^2\sigma_y^2 + C_2)} \tag{4}$$

SSIM evaluates different windows images (x and y) using the window average (μ_x and μ_y), the window variance (σ_x and σ_y), the windows covariance (σ_{xy}). The remaining two variables (C_1 and C_2) are employed to avoid instability when $\mu_x^2 + \mu_y^2$ is close to zero. More precisely, these variables are obtained with the following formula $C_{1/2} = (k_{1_2}L_{1/2})^2$, where L is the pixel-values dynamic range k is a constant (usually $k_1 = 0.01$ and $k_2 = 0.03$).

For completeness, we describe the PSNR formula since we report it alongside SSIM. Like Mean Squared Error (MSE), PSNR is straightforward to compute and has specific physical meaning. PSNR computation is shown in *Equation 5*, where MAX_I is the maximum value of the original image and the MSE is calculated with *Equation 6*, where I is the original image and K is its approximate version.

$$PSNR = 20 \cdot log_{10}(\frac{MAX_I}{\sqrt{MSE}}) \tag{5}$$

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2 \tag{6}$$

The auto-tuning algorithm gives as output the mantissa and exponent sizes that should be used to avoid noticeable precision loss. As threshold, we use 0.99 in order to have an approximate image as much similar to the original one [71]. We employ the precision tuning method with the overhead of running the application multiple times to detect the target precision. However, we use binary search because it has lower time complexity ($O(log_2N)$) compared to brute force algorithms such as linear search, which has time complexity in the order of $O(N)$, where N is the search space, which in this case is the sum of the number of mantissa and exponent bits.

C. ACCELERATOR DESIGN

Modern Intel FPGAs such as Intel Arria 10 and Stratix 10 have support for single-precision floating-point operations [9]. Their DSPs can perform Fused-Multiply-Accumulate (FMA), which counts as two floating-point operations. However, when employing fixed-point representation, they usually do not reduce the DSPs usage [72]. Differently, Xilinx FPGAs have smaller DSPs [73], thus causing a larger use of DSPs for floating-point operations, e.g., 3 DSPs for a single multiplication. This property makes Xilinx FPGAs a worse candidate for single-precision application compared to Intel FPGAs, but, at the same time, an interesting candidate for exploring smaller data types that can be mapped on a smaller number of DSPs. Indeed, we target a Xilinx Alveo U50 [74] for deploying our accelerators. The FPGA mentioned above is a small form factor board with a large number of resources, PCIE3 connection, HBM2 memory, and a TDP of 75W. The Alveo U50 is connected to a host system through a PCIE3 X16 connection as shown in *Figure 9*. Modern large FPGAs such as the Alveo U50 are built with multiple Super Logic Regions (SLRs). An SLR is a single FPGA die slice contained in an SSI (Stacked Silicon Interconnect) device [73].

We develop the accelerators into the Xilinx Vitis 2020.2 [75] tool flow, which is shown in *Figure 9*. Xilinx Vitis needs a source code with embedded OpenCL API to run on the host processor to schedule and control the execution of the accelerators. A similar approach is followed for the accelerators code: the source code containing HLS pragmas or optimization directives is compiled and linked by the Xilinx Vitis compiler. Unlike Intel, Xilinx FPGAs expose developers to a deeper level of optimization details, e.g., array partitioning and IP implementation with resource constraints. The kernel compilation step consists mainly of transforming the source code into HDL. At this stage, programmers can detect possible optimization opportunities and or stalls. The linking stage maps the accelerator on the FPGA by employing user configuration directives such as computing units and memory channel connections. We use THLS [60]

TABLE 4: WSClean parameters: the top part of the table shows parameters that exclusively depends on the observation and on the radio-telescope structure; the bottom part reports the CLEAN parameters employed for Cotton Schwab run. These parameters are the most commonly used. However, the CLEAN algorithm heavily depends on the user parameters. More complex CLEAN can be used for extracting the sky image, such as the Multiscale CLEAN.

Parameter	Value	Description (unit)
size	6000 6000	output x and y dimensions (pixels)
scale	5 asec	scale of a pixel (degrees)
use-idg	active	-
auto-threshold	3	CLEAN stop condition (sigma)
niter	50000	number of minor CLEAN iterations
mgain	0.85	gain per major CLEAN iteration
weight	briggs 0	weighting mode and robustness
taper	gaussian 2amin	

for mapping custom floating-point operations on FPGA [57]. Since a well-known limitation in the Xilinx Vitis accelerator development with OpenCL is the missing support of arbitrary precision [75] libraries, which are fundamental for fixed-precision computation, small integers, and custom floating-point (THLS is built on top of that), we employ C++ kernels with HLS pragmas.

D. EVALUATION WITH STATE-OF-THE-ART ARCHITECTURES

Each evaluated architecture needs to be profiled with specific tools. While applications running on CPU can be profiled with a large number of profiling methods, we choose `perf` [76] since it is available in most Linux distributions and is easy to extract information such as floating-point operations count, DRAM memory traffic, and power consumption (it is usually not needed to have root-access). On the other hand, GPUs typically have proprietary tools. Indeed, we use NVIDIA nvprof [77] and AMD CodeXL [78] for profiling flops and DRAM accesses on the GTX 750 and the RX 550. We count the memory requirements and the number of floating-point operations placed for evaluating the FPGA performance. For measuring the power consumption on FPGA and GPUs, we extend libpowersensor [79].

We select CPU and GPU architectures with similar characteristics such as peak performance and power consumption, which are reported in *Table 5*. However, as reported in *Table 6*, we have to employ an NVIDIA GTX 750 instead of an NVIDIA GTX 1050 Ti, which would be the preferred choice given its lithography technology of 14 nm. This is forced by the missing support of power measurement hardware

TABLE 5: Hardware employed for comparison.

Intel i9 9900k	8 cores, 2 threads per core, 4.0 GHz all cores, 16 MB L3 Cache, 64 GB DDR4 3600 MHz
NVIDIA GTX 750	512 CUDA cores, 1.14 GHz, 2 MB L2 Cache, 2 GB GDDR5
AMD RX 550	8 compute units, 1.09 GHz, 512 KB L2 Cache, 4 GB GDDR5
Xilinx Alveo U50	872 K LUTs, 1743 K Registers, 5952 DSPs, 8 GB HBM2

counters on the GTX 1050/1050 Ti [85]. According to [86], we would expect, for the same chip size, an improvement of ~2x in terms of power consumption efficiency. Furthermore, the TDP values reported for the GPUs are the power-cap limits read in the system out-of-the-box. Indeed, these values are reduced compared to the limits advertised: 35 W instead of 50 W for GTX 750 and 38 W instead of 55 W for RX 550. While the peak bandwidths reported in *Table 6* are extracted from the device datasheets, the peak performance is computed by multiplying the device frequency and the number of operations that can be computed in parallel in a cycle for each unit. For instance, the Intel i9 9900k has 8 cores that run at 4 GHz; each core can compute 32 flops (we are considering FMA as two flops) per cycle. These values are verified by using synthetic benchmarks such as clpeak [87].

We apply a similar computation for evaluating the performance of the Alveo U50. As shown in [81] we compute the theoretical performance of the Alveo U50 considering

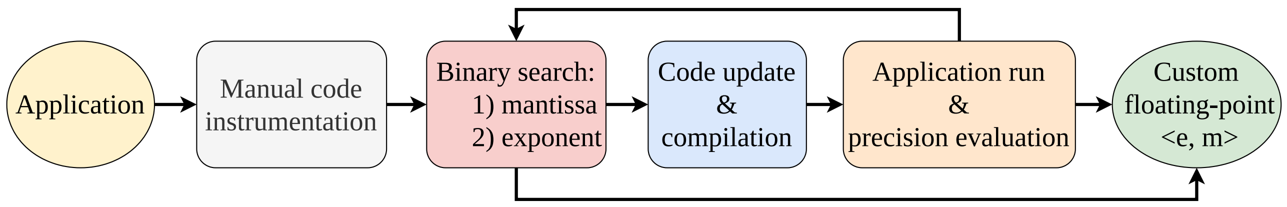

Application
Manual code instrumentation
Binary search: 1) mantissa 2) exponent
Code update & compilation
Application run & precision evaluation
Custom floating-point <e, m>

FIGURE 8: Binary search algorithm.

TABLE 6: Peak performances of the compared architectures. The energy efficiency* is the peak estimate derived by the ratio of the peak performance and the thermal design power (TDP). For the Xilinx Alveo U50, we report the theoretical peak performance by employing all the DSPs for FMA operations at the frequency reported by Xilinx [80] for the adder and multiplier IP (724 MHz) and at the frequency advertised by Xilinx for this FPGA board (300 MHz). We also report the Empirical peak performance obtained by using the FER synthetic benchmark [81].

Architecture	**Peak Performance**	**Bandwidth**	**TDP**	**Energy efficiency***	**Process**
Intel i9 9900k	1.024 TFLOP/s	57.60 GB/s	95 W	10.79 GFLOP/W	14 nm Intel
NVIDIA GTX 1050 Ti	2.138 TFLOP/s	112.1 GB/s	75 W	28.50 GFLOP/W	14 nm Samsung [82]
NVIDIA GTX 750	1.164 TFLOP/s	80.19 GB/s	38 W	30.63 GFLOP/W	28 nm TSMC [83]
AMD RX 550	1.097 TFLOP/s	96.00 GB/s	35 W	31.34 GFLOP/W	14 nm GlobalFoundries [84]
Xilinx Alveo U50	**Peak Performance**	**Bandwidth**	**TDP**	**Energy efficiency***	**Process**
Theoretical (724 MHz)	1.547 TFLOP/s	316 GB/s	75 W	19.77 GFLOP/W	16 nm TSMC
Theoretical (300 MHz)	0.641 TFLOP/s	316 GB/s	75 W	8.55 GFLOP/W	16 nm TSMC
Empirical (292 MHz)	0.535 TFLOP/s	316 GB/s	75 W	6.84 GFLOP/W	16 nm TSMC

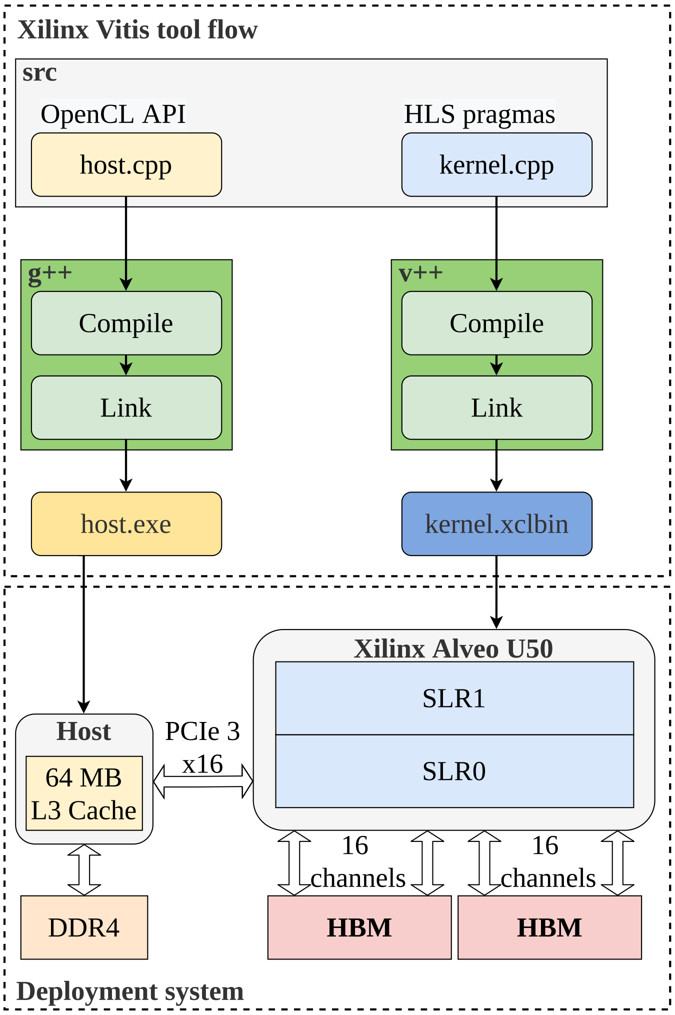


FIGURE 9: Representation of the Xilinx Vitis toolflow (top box) and how it relates to the deployment system (bottom box).

the maximum number of FMA operations (one addition and multiplication [80]) that could be theoretically be placed on this FPGA. Then this number is multiplied by the minimum of the frequency advertised by Xilinx for the single-precision addition and multiplication IP, which is 724 MHz [80]. A theoretical peak of 1.547 TFLOP/s is obtained by multiplying this number by 2 since we consider the FMA 2 flops. Since it is very difficult to achieve such frequencies as reported in [81], we employ the one Xilinx advertised for this FPGA: 300 MHz. This number (0.641 TFLOP/s) is considerably lower with respect to the theoretical one. However, this is still far from what can be really achieved on FPGA. Indeed, [81] shows that usually, a better upper bound is represented by employing 70% of the LUTs or 80% of the DSPs. For completeness, we also compute the peak performance using the FER (FPGA Empirical Roofline model) synthetic benchmark [81], and we obtain a value of 0.535 TFLOP/s.

V. ANALYSIS

To determine the bottleneck in the WSClean imager, we perform a bottleneck analysis (*V-A*). Then, we carry out a data types precision analysis (*V-B*) to understand the precision requirements for the identified bottleneck to be used for the accelerator design.

A. BOTTLENECK ANALYSIS

As shown in *Figure 10*, we first evaluate the execution time breakdown of the overall imaging pipeline for different datasets. The trend in the execution time breakdown is comparable for all datasets. More precisely, the most time-consuming step is inversion. Indeed, inversion needs to be run two times more than the prediction to compute the PSF and the dirty image. Typically, the deconvolution algorithm is the less critical phase.

B. DATA TYPES EXPLORATION

We evaluate custom precision floating-point data types employing software emulation since common architectures such as CPUs and GPUs usually support single- and half-precision floating points.

We notice a fundamental application property: since we are reducing the precision of the gridding kernel, it is sufficient

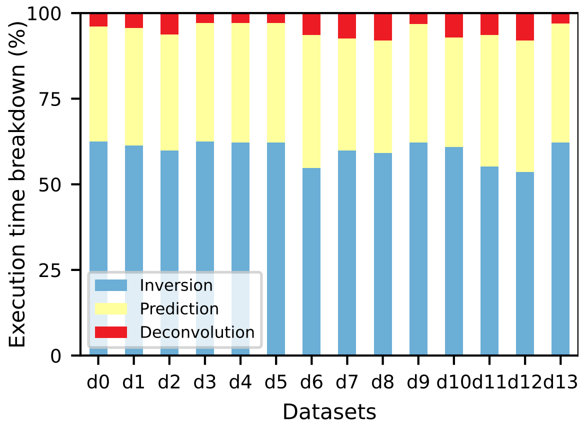


FIGURE 10: Execution time breakdown of WSClean for different datasets.

to compare the dirty images instead of the cleaned image to evaluate the accuracy. The algorithm's intrinsic nature easily explains this: the dirty image is a sky image with added noise, and the CLEAN algorithm extracts the brightest sources at each iteration. Thus, each successive iteration needs an equal or smaller dynamic range. This feature helps evaluate the application requirements faster since we need just run the gridding algorithm once to generate the dirty image, avoiding running the degridding and any CLEAN iterations, thus drastically reducing the analysis time. More precisely, we need to perform the gridding kernel once to generate the dirty image, once for the PSF, and two times during the CLEAN major iterations. Therefore, for this particular case, the analysis is about 4x faster than running the whole WSClean application.

We show in *Figure 11* how the reduced precision affects the radio-astronomical images. Noise effects can be noticed by using a mantissa of only 10 bits. Another important observation regards the relationship between dirty and clean images. In *Figure 11(a)* the images obtained emulating brain floating point contains a large quantity of noise, the same image cleaned (see *Figure 11(b)*) is empty. Therefore we can conclude that it is sufficient to analyze the dirty image to understand the precision requirements for the entire imager paying attention to the error indicators (SSIM).

In *Figure 12* we evaluate the accuracy of dirty images for different data-types for the gridding kernel in terms of SSIM and PSNR compared to the single-precision floating-point version. The precision requirements depend on the datasets employed, but the combination with 11 bits for the mantissa and 6 bits for the exponent can satisfy almost all the selected datasets. Furthermore, data types such as half-precision and fixed precision are not suitable for this field since the dynamic range is too small. Other data types with very small mantissa, such as brain floating point and NVIDIA Tensor Float, are not accurate enough to correctly represent all the faint features in the image. It is necessary to specify that the Tensor Float representation on NVIDIA GPU can only be used to perform warp matrix-to-matrix multiply and accumulate operations. In this work, we consider this data type representation for the whole kernel.

VI. CUSTOM PRECISION ACCELERATOR ARCHITECTURE

Optimizing the gridding algorithm on Xilinx FPGAs requires different steps. We first describe in *VI-A* the high-level structure of the accelerator by explaining the HLS optimization applied. Then, we explain the employed optimization for lookup tables and reduced precision respectively in in *VI-B* and in *VI-C*. Finally, we discuss in *VI-D* all the different methods we explore for placing the accelerator on FPGA through Xilinx Vitis and the design points of our prototypes.

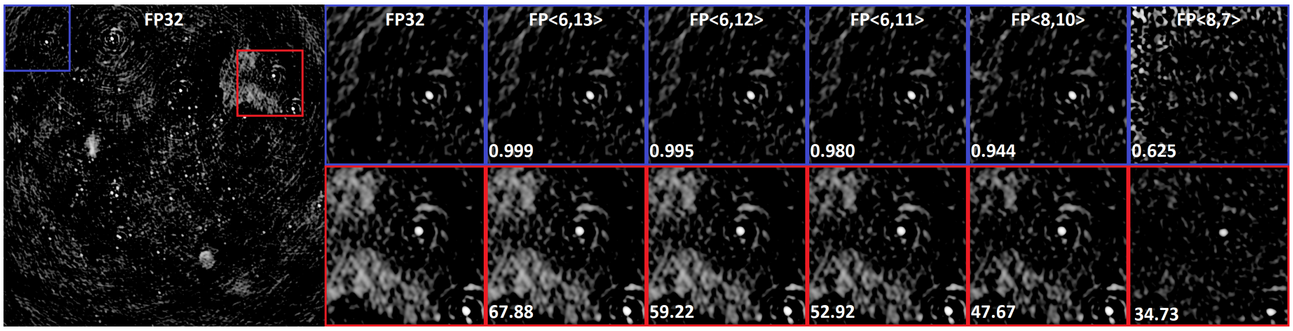
A. GRIDDING ACCELERATOR

We report the HLS optimized pseudocode of the subgrid computation in *Listing 1* and its high-level representation in *Figure 13*. We summarize the main optimization applied to get our highest performance prototypes:

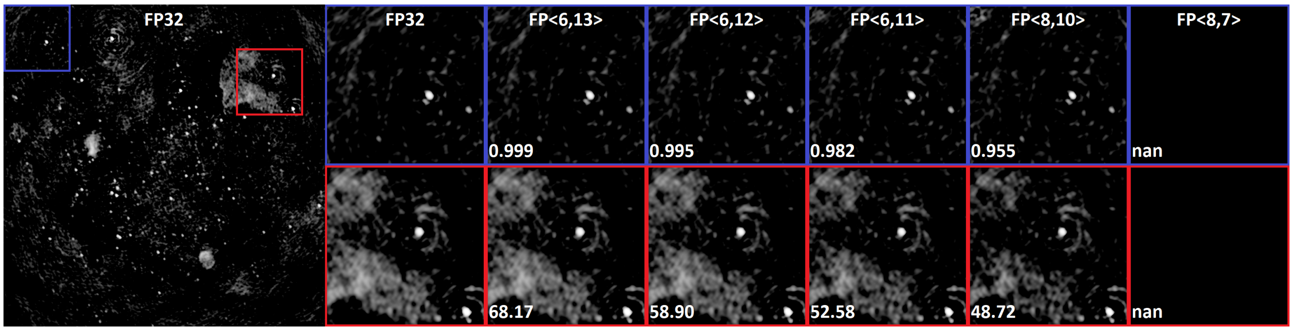
Memory management: We first optimize the data accesses from the HBM memory by using multiple memory channels and widening the AXI (Advanced eXtensible Interface) width to 512 bits for the bus that transfers most of the data. More precisely, we use three channels per compute unit: the first one for the input of the main computation block (subgrid computation), the second one for the input of the post-processing pipeline (Aterms, tapering, reorder, and FFT), and the third one for the output subgrid (see *Figure 13*) The data is moved into local buffers to exploit reuse and improve the memory access latency.

Initiation interval of the subgrid computation: The accelerator described in HLS is implemented as a hardware pipeline where ideally, the pipeline is *stall-free*, and new data is fed into the pipeline every cycle. In this case, the *initiation interval* (II) is equal to 1. In the case of stalls, the II could be larger than 1. Depending on the design, it then takes several cycles for the operations on that data to complete. To achieve II=1, we exchange the loop order (see *Algorithm 1*). The new loop order reduced the Read after Write (RaW) dependencies relative to the subgrid pixel update.

Parallelism: We increase the parallelism with respect to the code mentioned above at different levels. We first increase the parallelism of the subgrid computation by unrolling the channels and the pixels loops. The unrolling is implemented by employing larger local memories (BRAMs) and by unrolling the loops (see lines 5 and 7 in *Algorithm 1*) to increase the number of parallel floating-point units. Moreover, *Figure 14* reports the performance and the resource usage for different unrolling factors for channels and pixels. While the performance increases almost linearly, resources occupancy makes larger unrolling more efficient in terms of resource savings, e.g. especially for DSPs (see *Figure 14*). Indeed, unrolling the loops of a factor 2 do not imply the 2x more utilization of this resource. However, in order to be able to place more units and have a better area usage and placement, the parallelism should not be excessive, e.g., the case 4_8 is using more than 50% of resources, leaving no space



(a) Dirty images.



(b) Clean images.

FIGURE 11: Comparison of (a) dirty images and (b) clean images with different data types precision. The numbers enclosed in the angular brackets represent the exponent and mantissa bits. FP<8,10> and FP<8,7> correspond respectively to NVIDIA Tensor and brain floating point (bfloat). While the dirty image a) is a noise image when using bfloat, the clean image in b) is completely dark. We can observe that the SSIM (and PSNR) does not vary significantly between dirty and clean images except for poor quality images. These values are for the whole image, and the blue and red squares are zoomed image sections. We can notice some visible differences for SSIM lower than 0.99.

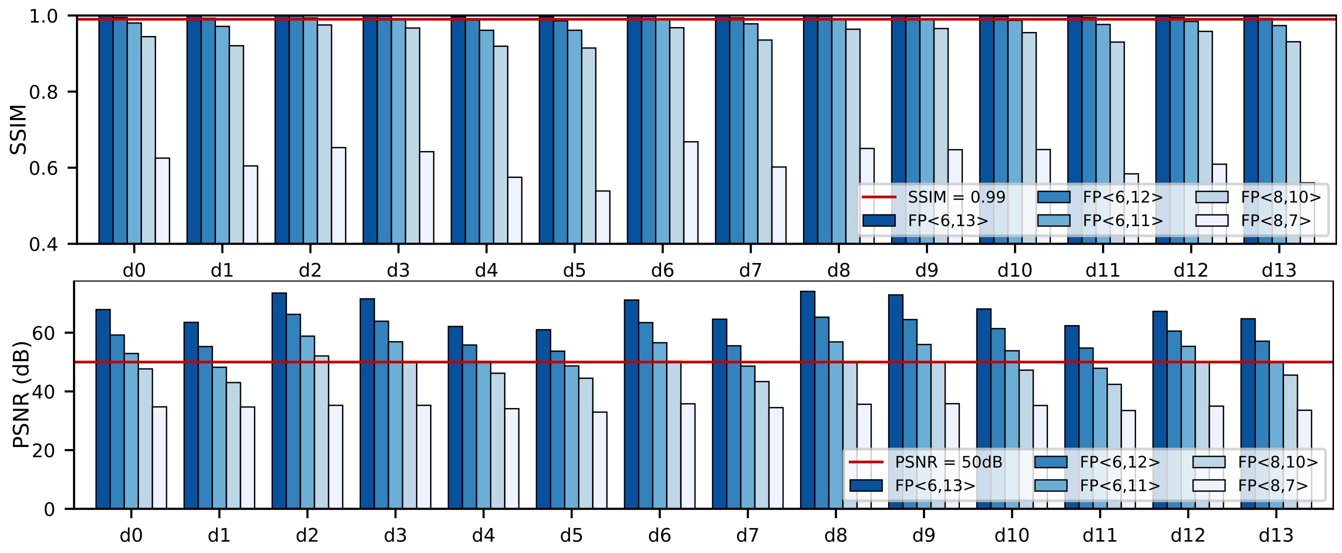


FIGURE 12: Precision accuracy for the selected 14 datasets (d0-d13) for different custom floating-point data types expressed in terms of SSIM and PSNR. The numbers enclosed in the angular brackets represent the exponent and mantissa bits. FP<8,10> and FP<8,7> correspond respectively to NVIDIA Tensor and brain floating point. High-resolution images closely similar to the original usually have an SSIM equal to 0.99 [71]. Given this threshold, reduced precision can produce high-quality images. However, it depends on the datasets. Indeed, in most cases, FP<6,11> or FP<6,12> reach the threshold, while smaller mantissa sizes do not. Such data types are not available on the standard accelerator platform such as GPU, which usually support half, single, double precision. Therefore a custom accelerator is needed. PSNR is just plotted for completeness, and the selected threshold of 50 dB is the maximum value for lossy images, and it serves only as a reference.

for placing multiple units. Another significant observation concerns the relationship between unrolling channels over pixels: unrolling over the channels increases the BRAM usage and reduces the DSPs usage, which is the opposite behavior obtained by unrolling more over the pixels. This happens because unrolling over the pixels introduces more cosine and sine computation, which requires a larger amount of DSPs than other operations such as additions and multiplications. We conclude that the best trade-off is to have similar unrolling factors for channels and pixels to achieve balanced use of DSPs and BRAMs.

Then, we instantiate multiple subgrid computations (see *Figure 13*) that are run in parallel by using the `DATAFLOW` pragma [88]. Finally, we increase the parallelism over the number of instanced kernel units depending on the design time closure difficulty.

Post processing trade-offs: The post-processing computation consists of applying the A-terms, the tapering, and after a pixel reordering an FFT. As mentioned above, the subgrid computation is replicated `N` times, thus making it possible to have just a single post-processing unit that is capable of processing the data from each subgrid computation in a pipelined fashion. This will add a delay given by the execution time of the post-processing computation, but with the advantage of a constant throughput and reduced resource usage. Since the subgrid computation is much more time consuming compared to the post-processing computation, the latter is designed as cheap as possible to have just sufficient performance to balance the subgrid processing, e.g., initiation interval greater than 1 and low parallelism, and save as many resources as possible making the subgrid computation the only responsible for the resource mapping on the FPGA board. For completeness, we report that the post-processing section responsible for applying the Aterms and the tapering has an initiation interval of 2 cycles (internal computation, the data is read at II=1 from the subgrid computation), which can be tuned up to 8 to facilitate the accelerator placing in some instances. We employ this trade off for the single-precision lookup table and custom floating-point design. The section responsible for the FFT has an initiation interval of 3 cycles.

B. COSINE/SINE LOOKUP TABLE AND REDUCED PRECISION

Similarly to [11], we employ a lookup table implementation to perform cosine and sine operations and save resources. Indeed, in Xilinx FPGAs, a `cosisin` operation, which computes the cosine and sine of a given angle, requires 11 DSPs compared with the 3 DSPs needed by the lookup implementation. We further reduce the DSPs usage to zero by not representing the phase in radiants. We move the multiplication used for the phase conversion in the outer loop and apply it when reading the `lmn` input data, which is a common factor when computing the phase offset and index and consequently the phase. The lookup table implementation consists in saving into BRAMs for pre-computed values for sine and cosine in the range of $[0; \frac{\pi}{4}]$. Then these values are used to compute the sine and cosine by employing the symmetry properties of trigonometric functions. Then, we have to tune the performance of the post-processing pipeline to let the accelerator achieve better frequencies (about 10 MHz higher).

C. REDUCED PRECISION

The reduced-precision accelerators are obtained by employing the Templatized Floating-Point HLS library [60] for replacing the floating-point operations. Even if we select a global minimum precision for the gridding kernel (homogenous custom floating-point operations), we decide to employ this library instead of the CPFP [59] since it is the most resource-efficient (see *Appendix A*).

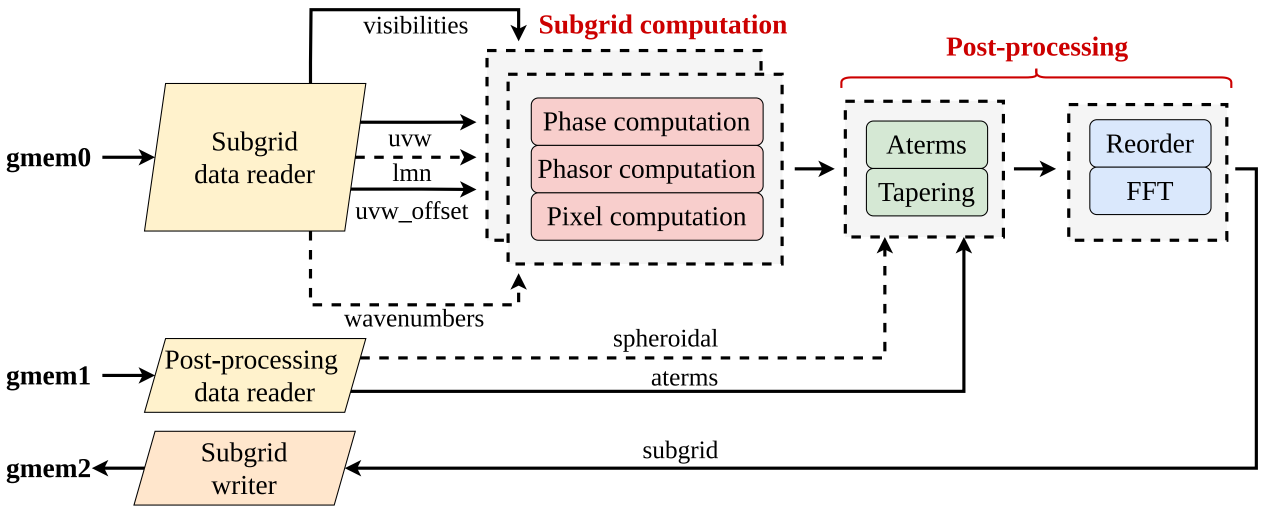
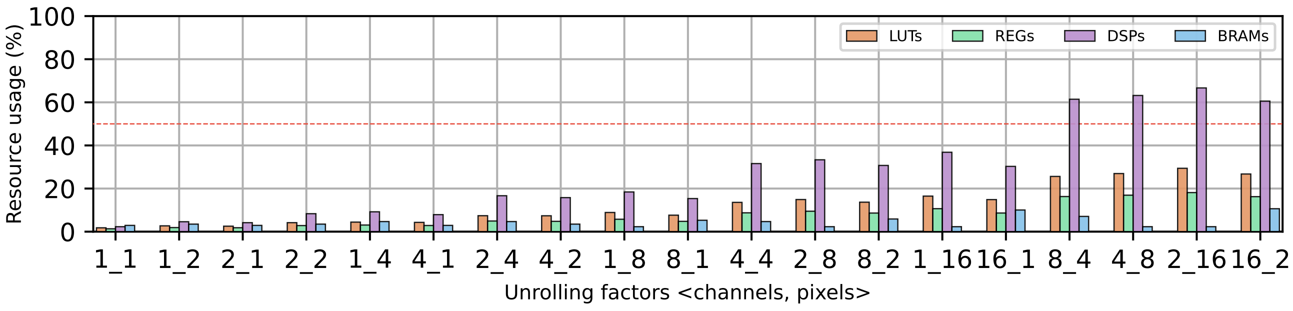


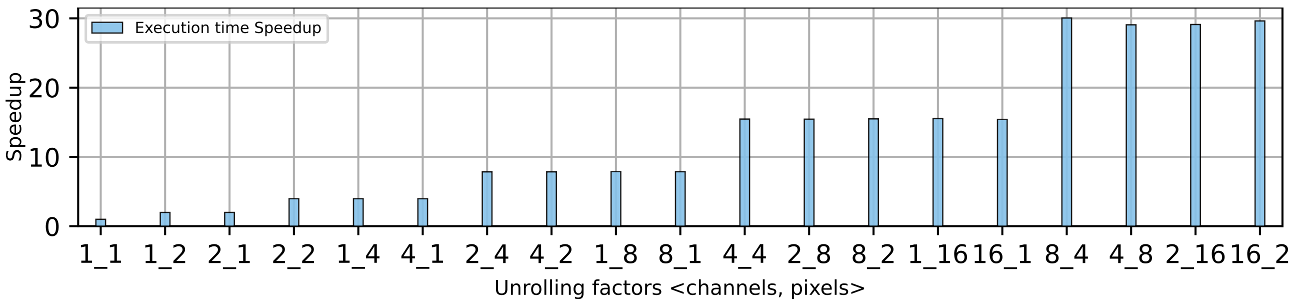
FIGURE 13: High-level scheme of one gridding compute unit: the data are read and written in parallel from different memory channels (gmem); the main part, subgrid computation, is replicated N times; finally, post-processing HW performs the Aterms, the tapering, and an FFT. Vectorized memory accesses are shown in black, while the scalar accesses are dashed.

Algorithm 1: Subgrid computation HLS pseudocode.

```
Input: visibilities, wavenumbers, uvw, uvw_offset, lmn
Result: subgrids
subgrids ⟵ 0;
for s in subgrids_per_cu do
    for t in timesteps do
        for c in channels do
            #pragma unroll factor = UNROLL_CHANNELS
            for p in pixels do
                #pragma unroll factor = UNROLL_PIXELS
                complex<float> pixel[pol]
                float lmn [3] ⟵ lmn[p]
                float phase_offset ⟵ compute_phase_offset(uvw_offsets, lmn)
                float phase_index ⟵ compute_phase_index(uvw, lmn)
                float phase ⟵ compute_phase(phase_index, phase_offset, wavenumbers)
                float phasor [2] ⟵ cosisin(phase)
                for pol in polarizations do
                    #pragma unroll
                    complex<float> pixel[pol] += visibilities[t][c][pol] * phasor
                end
            end
        end
    end
end
```



(a) Resource usage.



(b) Speedup.

FIGURE 14: Resource usage and speedup of the subgrid computation (not including the post-processing pipeline since it is not a critical computation) for different channels and pixels unrolling factors. The resources values refer to the kernel only, therefore without considering the overhead used by Xilinx Vitis, and are normalized by the maximum number of resources available in the device. The execution time is normalized by the largest values, which is the subgrid computation without unrolling (<1,1>).

In order to have a portable accelerator, we use the same host-kernel interfaces as for the single-precision floating-point accelerators. Then, we add some conversion steps to the input (from single precision to custom precision) and to the output (from custom precision to single precision) of the subgrid computation. This design choice is supported by the fact that the application is purely compute bound, and the conversion does not affect the performance significantly. Since sine and cosine are not available in the library mentioned above, we adapt the lookup implementation used for single-precision by adding a custom floating-point round to integer method that is used to determine the index of the lookup table. As mentioned above, we have to tune the performance of the post-processing pipeline to achieve the accelerator placement at a reasonable frequency (greater than 250 MHz), which means getting a significant speedup. Indeed, when using more resources, the achieved frequency becomes considerably low, e.g., 200 MHz, thus not being beneficial to place multiple units.

D. DEVICE-SPECIFIC CONSIDERATIONS

After optimizing the high-level synthesis code, one of the main tasks to get the best performance from an FPGA device is the accelerator placement. While it is possible to let the Xilinx Vitis tool flow map the accelerator automatically on the device, it is more efficient in terms of achieved frequency and, consequently performance to fully customize the accelerator mapping. We report the most challenging tasks we face during the FPGA placement:

Super Logic Regions: *Figure 15* shows an example of two accelerators with two compute units each placed on the Xilinx Alveo U50. The Xilinx Alveo U50 consists of two Super Logic Regions (SLRs), and it is recommended to map an accelerator on a single SLR. Crossing two SLRs can cause a critical path even if the SLRs are connected with special registers that try to mitigate this problem. During the placement of our accelerators, we find out that each SLR is divided into two subregions [73], and in the middle of them, there is a region consisting of units responsible for managing the input clock. Traversing the two regions as shown in *Figure 15* can introduce critical paths that will negatively affect the maximum clock frequency of the accelerator. In our most resource-demanding design, the single-precision lookup table and the reduced-precision ones, we have to instantiate multiple units at different levels to meet timing requirements: 1) one accelerator/compute unit per SLR to avoid SLR crossing, and 2) multiple subgrid computation in each compute units to avoid clock region crossing.

Static region overhead: When mapping application by using HLS a static region (blue area in *Figure 15*) is flashed on the FPGA for supporting (accelerated) applications by using HLS (see *Table 7*). This static region has the task of managing interconnections such as AXI. It also makes the programmers able to only place the accelerator in the dynamic region, which results, as depicted by *Figure 15*, in two asymmetric SLR sub-regions, thus being important to place, for a large design, more compute units on the left side than on the right side. This observation leads to placing multiple compute elements, whenever possible, in the same accelerator to help the tool find a better hardware placement.

TABLE 7: Xilinx Alveo U50 resources: total indicates the overall number of resources of the FPGA, while the dynamic region reports the available resources for accelerator deployment using Xilinx Vitis HLS, which is also reported in percentage.

Resources types	Total	Dynamic region	Available (%)
LUTs	872 K	731 K	83.83%
REGs	1743 K	1462 K	83.88%
DSPs	5952	5340	89.72%
BRAMs	1344	1128	83.93%
URAMs	640	608	95.00%

HBM memory channels: HBM2 memory can be employed efficiently for both memory-bound and compute-bound applications. In the first case, the large memory bandwidth will improve the application runtime by speeding up the memory transfers. In the second case, which coincides with our case, the multiple channels allow instantiating multiple accelerators that access independent memory spaces. Related to the HBM channels, there is another key issue: the Alveo U50 HBM memory channels are connected directly to the SLR0 [89], which means that additional logic is needed to transfer the data to the SLR1. When placing multiple accelerators and/or large sub-units that access the memory, connecting the AXI interfaces to distant memory channels is beneficial to avoid a long critical path between memory and processing. We carefully select the placement of HBM memory channels to avoid critical paths caused by area congestion, e.g., between two employed HBM channels, we decide to leave at least three channels unused.

More precisely, the Alveo U50 has 32 HBM2 channels, of which only 28 are usable due to power budget limitations (the peak bandwidth of 316 GB/s can be achieved by employing 24 channels). As shown in *Fig. 3* we use 3 HBM channels per compute unit. Therefore, we employ 6 channels in the lookup and reduced precision implementation because we instantiate two compute units. This choice is determined by the mentioned above congestion area issues. Indeed, as shown in *Fig. 16* the application is never memory bound by any of the considered architectures. The bandwidth of a single HBM channel (13.2 GB/s) would be sufficient to satisfy the bandwidth application requirements (the arithmetic intensity value shown in *Fig. 16*), but it would lead to difficult routable designs.

Vivado strategies [90]: Xilinx Vitis [75] is built on top of Xilinx Vivado, which is responsible for placing and routing the design on the FPGA board. Differently from the CPUs and GPUs programming model, the user can fully customize placement and routing strategies. The choice is between predefined implementation strategies or fully cus-

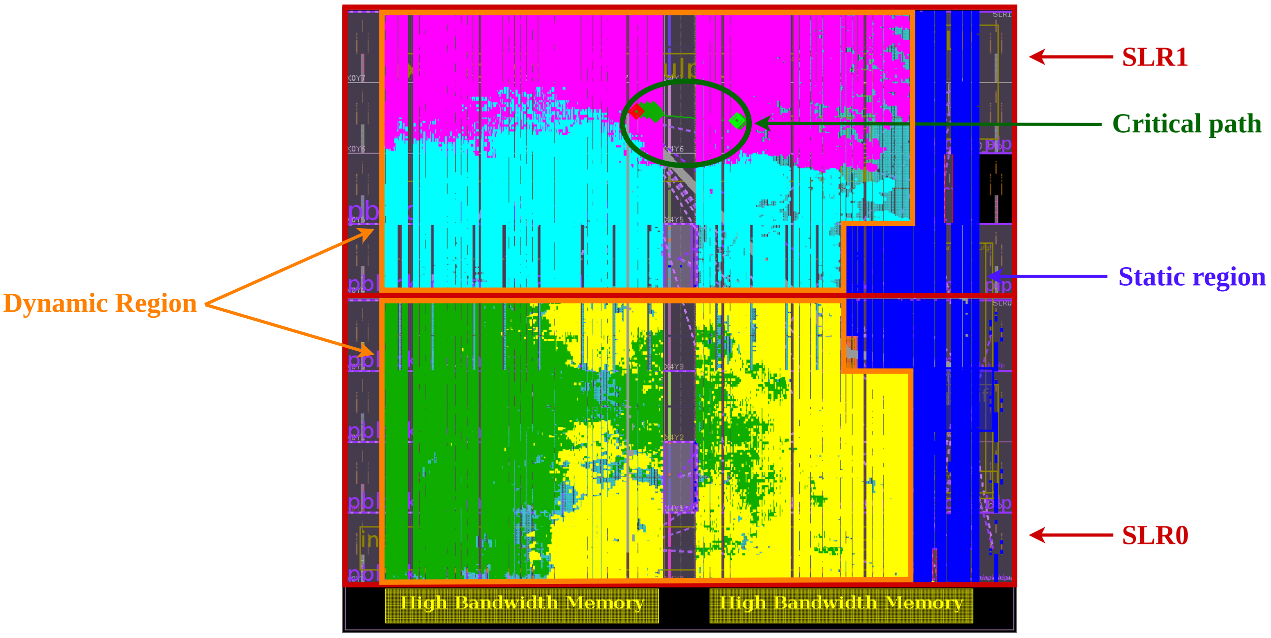


FIGURE 15: Example of an accelerator placement on Xilinx Alveo U50. In blue is reported the static region for deploying accelerators with Xilinx Vitis. The dynamic regions, which are the areas where the accelerators (yellow, purple, green, light blue) are mapped, are highlighted in orange. The FPGA is divided into multiple Super Logic Regions, FPGA die slices that compose large FPGA boards. Critical paths can usually occur when crossing multiple SLRs. However, critical paths can occur in the same SLR when connecting physically distant components and can be due to traversing the clock region like in the highlighted case.

tomizable strategies that reduce power consumption, area usage, improved performance, re-timing, etc. However, the main strategies are accessible by the user simply changing the Vitis Compiler optimization flags. We notice that both approaches lead to similar HW results. Indeed, when the frequency of the default Vivado strategy (-O0) is not able to meet the power constraints, we employ the PowerOpt strategy (-O1), e.g., single-precision lookup table, or when it is not able to meet the timing constraint, we use the ExtraTimingOpt (-O3), e.g., to achieve higher frequencies in the reduced-precision accelerators.

Pblock placement: This is a technique that can be applied after the design is implemented for guiding Xilinx Vivado towards a better design placement. With this option, the user can visually select from the GUI where to place certain units by creating a physical constrained region, the so-called pblock. This usually helps in cases of a desired higher frequency. However, we did not notice a significant improvement in our accelerators since they were already close to the highest advertised frequency.

Frequency overclock: Although Xilinx advertises 300 MHz as the maximum frequency for the HLS kernels, in reality, it is possible to achieve higher frequencies. Indeed, in [91] the authors match the same HBM frequency of 450 MHz for small accelerators. However, this is not achievable for larger accelerators that usually reach frequency in the range of 200-300 MHz [81, 92]. We manage to increase the frequency of the baseline single-precision floating-point accelerator since the resource usage is not so close to the maximum, and it is able to reach the standard frequency (300 MHz) by just employing the Vivado Default implementation strategy.

E. DESIGN POINTS

Here we report the main design points of our accelerators:

Single-precision: Our baseline accelerator in single-precision (`FP32`) is obtained by simply using 2 subgrid computations with unrolling factor <4,4> and a single post-processing pipeline. In this case, due to the modest use of LUTs and FFs, it is not necessary to place the computation over different SLRs or apply any particular strategy. We are also able to achieve higher frequencies (`FP32_OC` in *Table 8*). Unfortunately, it is impossible to use more resources on this FPGA without violating timings (slow clocks that make the accelerator inefficient) or power constraints (high power budget).

Single-precision lookup-tables: The cosine/sine lookup table implementation can significantly reduce the number of DSPs employed. We notice only a small, but still significant reduction of DSPs since the main computation consists of FMA operations. We manage to improve the performance by placing 50% more computations. This is achieved by using an unroll factor of <2,4> and three subgrid computation units. This accelerator is then instantiated in each SLR, thus having two post-processing pipelines. In this case, to be able to place the accelerator, we have to reduce the number of DSPs by employing the `config_op` option during the HLS compilation to implement all the floating-point additions and multiplication with LUTs. To achieve better frequency, we employ the ExtraTiming_Opt strategy and reduced the post-

TABLE 8: Resource utilization for the highest performance gridding accelerators. `FP32_OC` is the single-precision floating-point prototype at higher frequency and `FP32_LT` is the single-precision floating-point prototypes the lookup implementation.

Version	LUTs	FFs	DSPs	BRAMs	Frequency
FP32	434 k (49.88%)	604 k (34.72%)	4114 (69.12%)	454 (33.74%)	300 MHz
FP32_OC	435 k (49.99%)	640 k (36.78%)	4114 (69.12%)	454 (33.74%)	346 MHz
FP32_LT	649 k (74.58%)	614 k (35.29%)	3142 (52.71%)	1045 (77.75%)	296 MHz
FPX_6_11	642 k (74.81%)	754 k (43.33%)	1956 (32.81%)	818 (60.86%)	300 MHz
FPX_6_12	656 k (75.39%)	767 k (44.10%)	1956 (32.81%)	818 (60.86%)	300 MHz

processing computation's performance. We notice through analysis, a lookup table of 2048 is sufficient to keep the SSIM close to 1. The reported accelerator `FP32_LT` uses a lookup table with 2048 entries.

Reduced-precision: we manage to place 100% more computation with reduced precision compared to the single-precision baseline accelerator and 50% more than the single-precision lookup table implementation. This accelerator consists of 2 subgrid computations with unroll factors <4,4> placed in each SLR. To achieve better frequency, we employ the ExtraTiming_Opt strategy. For the reduced-precision prototypes, we use a lookup table with 2048 entries. In order to achieve the 300 MHz frequency, we employ the reduced performance post-processing computation. Higher frequencies do not meet timing constraints and the power budget (or TDP).

VII. EVALUATION AND DISCUSSION

In *VII-A* we report the area usage of the proposed accelerators, and we assess our accelerators performance in *VII-B* by employing the roofline model [94] and by measuring the throughput and the energy efficiency. Then, we highlight significant lessons learned during this work in *VII-C*.

A. AREA USAGE

In *Table 8* is presented the area usage of the highest performance accelerators here designed. More precisely, we report as `FP32` the baseline single-precision floating-point accelerator, which can reach the advertised frequency of 300 MHz. Moreover, we show the same design with higher frequency (346 MHz) as `FP32_OC`. Resource usage does not vary significantly. Compared to [11] our DSP usage is significantly lower due to the mentioned above issues regarding the static region and timing closure.

The reported single-precision lookup-table (`FP32_LT`) accelerator has a higher overall resource usage because we manage to place more units compared to `FP32`. DSPs usage is lower since the lookup-table sine and cosine computation uses fewer DSPs with regards to the baseline design, and we implement addition and subtraction without DSPs. The achieved frequency of 295 MHz is close to the advertised one, which is difficult to achieve due to power-budget constraints.

The reduced-precision accelerators (`FPX_6_11` and `FPX_6_12`) consume more resources than the baseline, but we can place two times more compute units. Furthermore, this design uses fewer resources than the baseline and consumes less power than the design with a single-precision lookup table. As already mentioned `FPX_6_11` consumes slightly fewer resources (LUTs and FFs) than `FPX_6_12` because of the smaller mantissa.

B. PERFORMANCE

We assess the performance of our accelerators against CPU and GPUs with similar peak performance and manufacturing technology. We first evaluate the performance achieved by each platform with the roofline model [94, 95] in *Figure 16*. The roofline model shows the performance obtained (TFLOP/s) and the analyzed kernel's arithmetic intensity (FLOP/Byte). The roofline defined in this way is not a measure of throughput but an indicator of how the application can be optimized for a certain architecture. Indeed, we report that only the NVIDIA GTX 750 can reach almost peak performance. This is because cosine and sine operations are offloaded to special units and indeed do not create bottlenecks. Differently, the AMD RX 550 does not have these special units, and the cosine and sine operations run at a quarter speed [96] compared to single-precision floating-point operations. We further notice that each cosine and sine function introduces three floating-point operations. Indeed, in *Figure 16* the RX 550 performance takes into account the largest number of instructions. We similarly reported the performance of the proposed accelerators showing how they are performing better than CPU. In particular, our best reduced-precision design is close to the AMD GPU in terms of TFLOP/s. Compared to [11] our baseline design has lower performance due to not having single-precision floating-point DSPs support.

In *Figure 17* we evaluate the throughput and the energy efficiency of the accelerators. More precisely, in *Figure 17(a)* we report the throughput in terms of Mega Visibilities per second (Mvis). *Figure 17(a)* evaluates the performance of our accelerators. All our designs have higher throughput compared to the i9 9900k up to 2.12x. Our best reduced-precision design is close to the AMD GPU performance. Moreover, the reduced-precision prototype is 1.84x faster than the single-precision baseline. However, the GTX 750 is the faster architecture due to their special function units mentioned above, and our baseline architecture is outperformed by the Intel counterpart proposed in [11] because of

the single-precision DSPs support of Intel FPGAs.
As shown in *Figure 17(b) and 17(c)* our accelerators outperform up to 3.46x in terms of energy-efficient the CPU. The single-precision lookup table implementation reaches 88.97% of the empirical peak performance. Our best reduced-precision design is 2.03x more energy efficient than the single-precision baseline design. However, it is 78.77% and 63.29% less energy-efficient than AMD and NVIDIA GPUs. We also observe that the AMD RX 550 is more energy-efficient than the NVIDIA GTX 750. This is mainly motivated by the different lithography technology. Scaling the performance of the NVIDIA GTX 750 chip to a 14 nm [86] the power consumption would be ~2x reduced, thus being more efficient than the AMD counterpart.

Summarizing the key observations are:

1) Our single-precision floating-point accelerators are more energy-efficient compared to CPUs with similar technology because of the better hardware utilization.
2) Reduced precision improves the accelerator's overall performance being much faster than CPU and achieving comparable performance compared to AMD GPUs thanks to the higher density of operations that we placed on the FPGA.
3) NVIDIA GPUs reach the highest percentage of peak performance exploited (~88%) compared to the other architectures, especially against AMD GPUs (~73%) thanks to the special units for sine and cosine. Overall, GPUs are the more energy-efficient architecture (see *Figure 16* and *17*) compared to CPUs and FPGAs with similar features due to their energy-efficient architecture (see *Table 6*).

C. LESSONS LEARNED

Radio-astronomical imaging applications usually employ single-precision or double-precision floating-point data types. We evaluate the use of reduced-precision data types for the gridding kernel and make the following observations:

Reduced precision applicability in radio-astronomical imaging: Different from artificial intelligence applications, where it is possible to highly reduce the data size, e.g. 1 or 8 bits [7, 100], radio-astronomical imaging needs higher precision for reconstructing sky images. Indeed, from our analysis, we observe that reduced precision can be applied in the state-of-the-art radio-astronomical imager. However, compared to AI tasks, the required precision is higher to avoid image artifacts. Tensor-Float floating-point numbers have a sufficiently high dynamic range for radio-astronomical imaging kernels, but the number of bits is too low to accurately represent the application values, such as visibilities and subgrids. Moreover, this format can only be used for specific

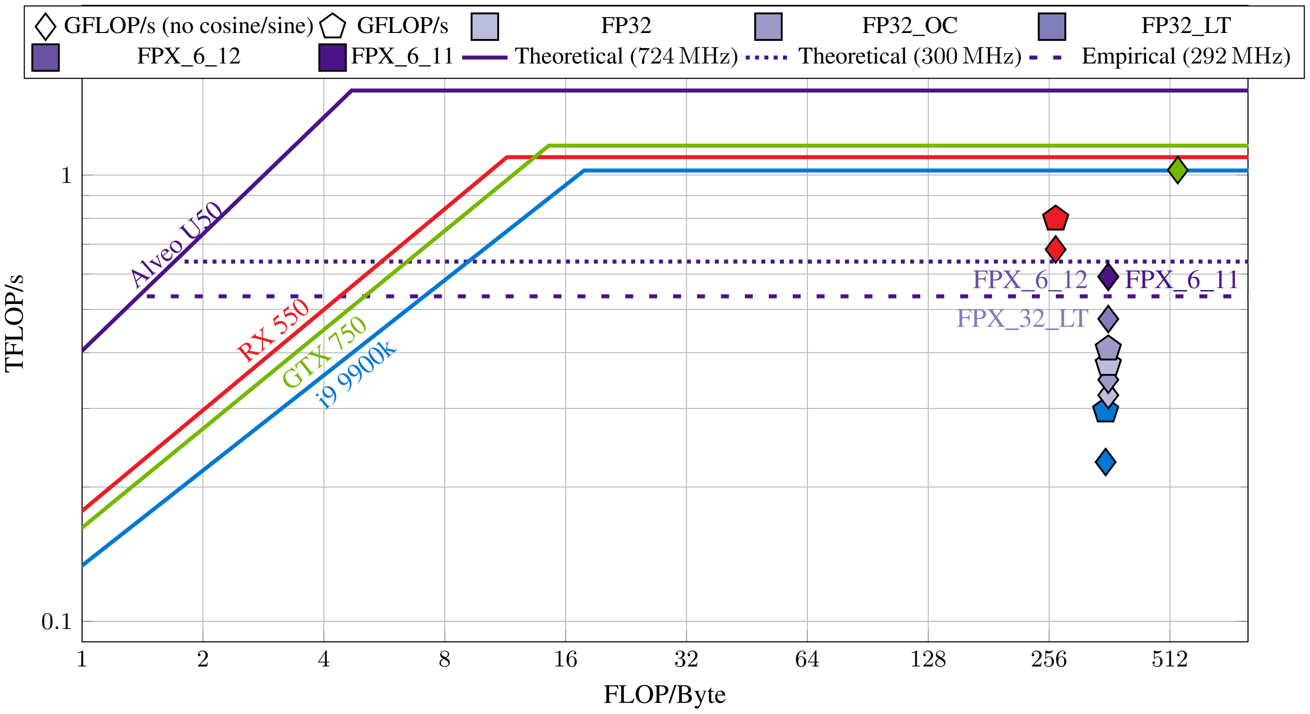
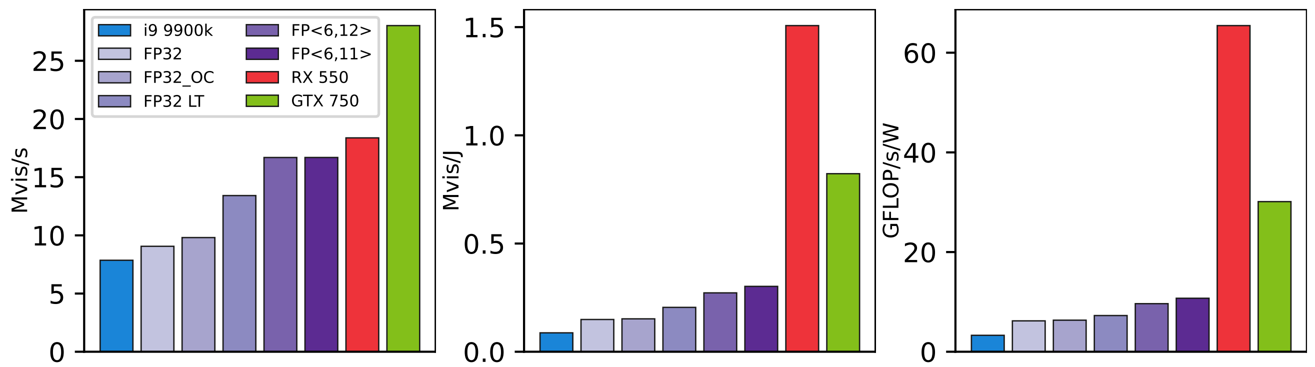


FIGURE 16: Roofline model of the gridding kernel on different architectures. The pentagon shape represented the TFLOP/s achieved by each platform measured using performance counters. The diamond shape shows the TFLOP/s achieved without including sine and cosine since specific architectures such as the AMD RX 550 uses multiple floating-point instructions to compute sine and cosine compared to the NVIDIA GTX 750 that has special function units for transcendental math operations [93]. Indeed, we show only one point for the GTX 750 and the lookup table accelerators (including the custom floating-point prototypes) since the sine and cosine operations are not executed as floating-point operations. Thus the two points have the same value. We report different horizontal roofs for the Xilinx Alveo U50 based on the discussion regarding peak performance in IV-D.



(a) Performance in terms of Mvis per second. (b) Performance in terms of Mvis per Joule (c) Energy efficiency in terms of GFLOP/s per Watt

FIGURE 17: Performance and energy efficiency evaluation on different architectures.

warp matrix-to-matrix multiply and accumulate operations.

Benefits of applying reduced-precision in radio-astronomical imaging: Reduced precision is a well-known technique for improving performance and energy efficiency [12]. It is a technique that can be applied to compute-bound applications to reduce the compute unit size and memory-bound applications to decrease memory bandwidth requirements. We observe a significant improvement in GFLOP/s and energy efficiency, respectively 81.96% and 84.71% compared to the standard single-precision accelerator and 25.75% and 33.02% compared to the lookup-table implementation.

FPGAs vs CPUs and GPUs: The state of the art confirms that FPGAs and GPUs are more energy-efficient than CPUs except in rare cases [101]. The major part of past works shows that FPGAs are more energy-efficient than GPUs. However, these works compare GPUs with higher performance and power consumption. In this work, we show, as presented in [11], that for this particular application domain, GPUs with similar peak performance, thermal design power and manufacturing process are faster and have better energy efficiency. GPUs are also easier to program compared to FPGAs, which require hours of compilation to generate the bitstream [102]. However, FPGAs have the flexibility to synthesize custom data types on hardware for assessing performance improvement against standard data types. In this work, we first evaluate the applicability of reduced precision for radio-astronomical imaging. Then, we design a reduced precision accelerator for radio-astronomical imaging on FPGAs reporting similar performance to GPUs without special sine/cosine units and improved performance and energy efficiency compared to its baseline.

Xilinx Alveo U50: We make the following observations about the Xilinx Alveo U50 FPGA:

- The static region consumes many resources (~17% LUTs and ~13% DSPs) reducing the maximum performance attainable with HLS accelerators.
- The power budget and timing closure constraints, which can be improved with Vivado strategies and overclocking, make it impossible to fit more compute units (see *Table 8*) in the designs reaching a maximum of 69% of DSPs (77% counting the static-region effect).
- Large accelerators are difficult to place as discussed in VI-D due to SLR placement. However, as previously

TABLE 9: Radio-astronomy related work.

Work	Date	Application	Platform	Optimization
Offringa [20, 35]	2014-2017	W-Stacking, CLEAN (Högbom, Cotton-Schwab, Multiscale)	CPU	Optimized full imager (WSClean)
Veenboer [6, 97]	2017-2020	Image-Domain Gridding	CPU/GPU	code optimization (added to WSClean)
Grel [98]	2018	Högbom CLEAN	FPGA	Custom accelerator of Högbom CLEAN formulated as a Compressive Sensing problem
Veenboer [11]	2019	Image-Domain Gridding	FPGA	custom accelerator
Seznec [15]	2019	Generic deconvolution	GPU	half-precision deconvolution
Hou [99]	2020	W-Projection	FPGA	custom accelerator
Corda [4]	2020	Large 2D FFT	FPGA	NMC acceleration
This work	2021	WSClean (Image-Domain Gridding)	FPGA	Reduced precision analysis and acceleration

discussed Xilinx FPGAs are appealing candidates due to their DSPs structure (see *IV-C*).

- A positive feature in HBM-based FPGAs is the possibility to map AXI interfaces to a larger set of memory channels, alleviating congestion issues arising from small numbers of memory channels.

VIII. RELATED WORK

Related work on radio-astronomical imaging acceleration on modern computing systems is described in *VIII-A*; moreover, we report related work on accelerating domain-specific-application by using Xilinx Vitis in *VIII-B*.

A. RADIO-ASTRONOMY ACCELERATION

Over the past couple of years, hardware technologies have significantly improved, and a fair amount of research has been done on optimizing radio-astronomy algorithms to satisfy the Square Kilometre Array requirements [103].

The current state-of-the-art full imager, WSClean, is proposed by Offringa et al. [20]. It consists of an entire radio-astronomical imaging pipeline, including W-stacking, which is an extension of the previous gridding and degridding algorithm, the so-called W-projection, and different CLEAN algorithms such as Högbom, Cotton-Schwab and Multiscale CLEAN [35].

Veenboer et al. [6, 97] optimize the Image-Domain Gridding [19], the current state-of-the-art fast algorithm for gridding and degridding for radio-astronomical imaging. They show how GPUs could reach almost peak performance and deliver better execution time and energy efficiency than CPUs. Image-Domain Gridding is now part of the WS-Clean imager. In [11] the authors also accelerate the gridding and degridding kernels on FPGA using a high-level-synthesis methodology based on OpenCL. Their FPGA implementation outperforms CPUs, but the GPU one is more energy efficient. Our single-precision accelerator baseline is outperformed by the one presented in [11] due to the single-precision floating-point DSPs support of Intel Arria FPGA [9]. However, we employ Xilinx FPGAs to assess the performance of a reduced precision accelerator for radio-astronomical imaging.

Hou et al. [99] implement an optimized prototype for the degridding algorithm on FPGA, outperforming both CPU and GPU by respectively 2.74x and 2.03x in terms of energy-delay product (EDP). However, they employ an outdated version of the degridding algorithm called W-projection, which does not reach the performance of IDG and does not include DDE corrections [28].

Corda et al. [4] focus on estimating the benefit of near-memory computing for huge images in IDG. They show that FFT is a critical bottleneck, which can be alleviated using architectures exploiting High-Bandwidth Memory. More precisely, they demonstrate how an FPGA design could reach a similar performance compared to a GPU with smaller memory and less memory bandwidth.

Seznec et al. [15] propose a simple deconvolution GPU implementation using half-precision data type. While half-precision floating-points speed up the algorithm significantly, this data type brings output degradation based on the dataset used. Furthermore, they focus on small images (2048x2048 pixels), while radio astronomical images are typically bigger, e.g., 5000x5000, 16000x16000 and larger [6].

Grel et al. [98] formulate the Hogbom Clean as an Iterative Hard Thresholding (IHT), a compressive sensing technique, to reduce the data dimensions. While this work provides compelling insights, they optimize the most simple Clean algorithm and use very low resoluted images (256x256 pixels).

To the best of our knowledge, our work is the first demonstration in proposing a custom floating-point analysis and architecture for radio-astronomical imaging by focusing on the main bottleneck (gridding kernel).

B. XILINX VITIS

Recent high-level synthesis work on Xilinx FPGAs is based on Xilinx's new programming tool-flow: Xilinx Vitis. Brown et al. [91] show the steps to take when optimizing an application for Xilinx FPGAs by employing Xilinx Vitis. Although some observations in their work have been helpful, they focus on very small FPGA designs (using about 2% of the available resources), while we try to put as many resources to good use as possible. Indeed, for a small accelerator, it is possible to reach a frequency of 450 MHz, while in a realistic use case, it turns out to be in a range of 200-300 MHz.

Calore et al. [81] benchmark the Alveo U250 with the FPGA Empirical Roofline model (FER), showing that there is a considerable difference between theoretical and attainable performance on FPGA, which usually is not so high in different architectures such as CPUs and GPUs. In our work, we employ the FER benchmark to determine the single-precision horizontal roof for the Alveo U50.

Nguyen et al. [102] evaluate FPGAs from different vendors with GPUs showing that modern GPUs are easier to program and better energy efficient except for rare cases such as fixed-point precision.

Choi et al. [104] benchmark HBM-based FPGAs from different vendors to compare the memory performance. They present insightful observations regarding how the HBM memory channels must be mapped on Xilinx Vitis, e.g. Alveo FPGAs usually has 30 channels available over a total of 32. Related to this work, we notice that it is crucial to carefully select the HBM channel when implementing the accelerator to avoid critical paths that can significantly reduce the achieved frequency.

IX. CONCLUSION

We present the first reduced-precision custom floating-point analysis and accelerator for radio-astronomical imaging. More precisely, we evaluate the bottleneck of WSClean, the state-of-the-art radio-astronomical imager. Then, we analyze for the first time the impact of reduced precision on radio-astronomical imaging by employing custom floating-point for the main kernel, the so-called gridding, from the state-of-the-art Image-Domain Gridding algorithm. We demonstrate that reduced precision could be applied to radio-astronomical imaging, but data types must be selected carefully based on the dataset to avoid precision loss. Indeed, our analysis excludes standard low-precision data types supported in modern GPUs and highlights insightful observations regarding how the analysis can be evaluated in a shorter time. We port the gridding algorithm (from Image-Domain Gridding) on Xilinx FPGAs in single-precision floating-point to serve as a baseline. Moreover, we map the first reduced precision prototype of gridding kernel on the same hardware by employing custom floating-point data types. We find that the reduced precision accelerator is up to 1.84x faster and 2.03x more energy-efficient than its single-precision version. Compared to the CPU, our best accelerator prototype is 2.12x faster and 3.46x more energy-efficient than the CPU. However, it is 9.16% and 40.44% slower and 78.77% and 63.29% less energy-efficient than AMD and NVIDIA GPUs.Thus we corroborate with the earlier observation that GPUs are the most energy-efficient architecture for radio-astronomical imaging, and custom precision support in the next generation of GPUs could further improve the performance/watt for radio astronomy imaging. Even though FPGA design support improved in the past years, FPGAs would need improved lithography technology [105], smarter HLS compilers, and faster placement tools to be competitive against GPU in this application domain.

This work paves the way for future research in this specific application domain. Indeed, it would be interesting to explore reduced precision at runtime, e.g., by reconfiguring the accelerators based on dataset requirements and/or user parameters, to further improve performance. Fine-grained reduced precision, which consists of differentiating the data types for different instruction or kernel sections, would be a viable option for higher performance and energy efficiency in future works.

ACKNOWLEDGMENT

We would like to thank Remi Robert from Ericsson Research for reviewing the first draft.

REFERENCES

[1] “Square Kilometre Array Phase 1 Central Signal Processor Software: An Overview,” in *2018 2nd URSI Atlantic Radio Science Meeting (AT-RASC)*, 2018, pp. 1–1.

[2] R. V. van Nieuwpoort and J. W. Romein, “Correlating Radio Astronomy Signals with Many-Core Hardware,” *International Journal of Parallel Programming*, vol. 39, no. 1, p. 88–114, Jun 2010. [Online]. Available: http://dx.doi.org/10.1007/s10766-010-0144-3

[3] R. Bolton, P. Broekema, T. Cornwell, G. van Diepen, C. Holli, M. Johnston-Holli *et al.*, “Parametric models of SDP compute requirements,” *SKA Office, Manchester, UK, Tech. Rep. SKA-TEL-SDP-0000040*, vol. 21, 2016.

[4] S. Corda, B. Veenboer, A. J. Awan, A. Kumar, R. Jordans, and H. Corporaal, “Near Memory Acceleration on High Resolution Radio Astronomy Imaging,” in *2020 9th Mediterranean Conference on Embedded Computing (MECO)*, 2020, pp. 1–6.

[5] R. Jongerius, S. Wijnholds, R. Nijboer, and H. Corporaal, “An End-to-End Computing Model for the Square Kilometre Array,” *Computer*, vol. 47, no. 9, pp. 48–54, 2014.

[6] B. Veenboer and J. Romein, “Radio-astronomical imaging on graphics processors,” *Astronomy and Computing*, vol. 32, p. 100386, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2213133720300408

[7] C. Wu, M. Wang, X. Chu, K. Wang, and L. He, “Low Precision Floating-point Arithmetic for High Performance FPGA-based CNN Acceleration,” 2020.

[8] J. G. Pandey, A. Karmakar, C. Shekhar, and S. Gurunarayanan, “An FPGA-based fixed-point architecture for binary logarithmic computation,” in *2013 IEEE Second International Conference on Image Information Processing (ICIIP-2013)*, 2013, pp. 383–388.

[9] Intel, “Enabling High-Performance Floating-Point Designs,” https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01267-fpgas-enable-high-performance-floating-point.pdf, 2020.

[10] R. Nane, V.-M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis *et al.*, “A survey and evaluation of FPGA high-level synthesis tools,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 10, pp. 1591–1604, 2015.

[11] B. Veenboer and J. W. Romein, “Radio-astronomical imaging: Fpgas vs gpus,” in *Euro-Par 2019: Parallel Processing*, R. Yahyapour, Ed. Cham: Springer International Publishing, 2019, pp. 509–521.

[12] S. Cherubin and G. Agosta, “Tools for Reduced Precision Computation: A Survey,” *ACM Comput. Surv.*, vol. 53, no. 2, Apr. 2020. [Online]. Available: https://doi.org/10.1145/3381039

[13] Z. Carmichael, H. F. Langroudi, C. Khazanov, J. Lillie, J. L. Gustafson, and D. Kudithipudi, “Performance-Efficiency Trade-off of Low-Precision Numerical Formats in Deep Neural Networks,” in *Proceedings of the Conference for Next Generation Arithmetic 2019*, ser. CoNGA’19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3316279.3316282

[14] M. M. Thomas, K. Vaidyanathan, G. Liktor, and A. G. Forbes, “A Reduced-Precision Network for Image Reconstruction,” *ACM Trans. Graph.*, vol. 39, no. 6, Nov. 2020. [Online]. Available: https://doi.org/10.1145/3414685.3417786

[15] M. Seznec, N. Gac, A. Ferrari, and F. Orieux, “A Study on Convolution using Half-Precision Floating-Point Numbers on GPU for Radio Astronomy Deconvolution,” in *2018 IEEE International Workshop on Signal Processing Systems (SiPS)*, 2018, pp. 170–175.

[16] J. W. Romein, “The Tensor-Core Correlator,” in *Astronomy and Astrophysics)*, 2021.

[17] N. Doucet, H. Ltaief, D. Gratadour, and D. Keyes, “Mixed-Precision Tomographic Reconstructor Computations on Hardware Accelerators,” in *2019 IEEE/ACM 9th Workshop on Irregular Applications: Architectures and Algorithms (IA3)*, 2019, pp. 31–38.

[18] “UltraScale Architecture DSP Slice,” https://www.xilinx.com/support/documentation/user_guides/ug579-ultrascale-dsp.pdf.

[19] S. van der Tol, B. Veenboer, and A. R. Offringa, “Image Domain Gridding: a fast method for convolutional resampling of visibilities,” *Astronomy & Astrophysics*, vol. 616, p. A27, Aug 2018. [Online]. Available: http://dx.doi.org/10.1051/0004-6361/201832858

[20] A. R. Offringa, B. McKinley, N. Hurley-Walker, F. H. Briggs, R. B. Wayth, D. L. Kaplan *et al.*, “WSClean: an implementation of a fast, generic wide-field imager for radio astronomy,” *Monthly Notices of the Royal Astronomical Society*, vol. 444, no. 1, p. 606–619, Aug 2014. [Online]. Available: http://dx.doi.org/10.1093/mnras/stu1368

[21] van Haarlem, M. P., Wise, M. W., Gunst, A. W., Heald, G., McKean, J. P., Hessels, J. W. T. *et al.*, “LOFAR: The LOw-Frequency ARray,” *A&A*, vol. 556, p. A2, 2013. [Online]. Available: https://doi.org/10.1051/0004-6361/201220873

[22] M. G. Labate, R. Braun, P. Dewdney, M. Waterson, and J. Wagg, “SKA1-LOW: Design and scientific objectives,” in *2017 XXXIInd General Assembly and Scientific Symposium of the International Union of Radio Science (URSI GASS)*, 2017, pp. 1–4.

[23] R. J. Selina, E. J. Murphy, M. McKinnon, A. Beasley, B. Butler, C. Carilli *et al.*, “The Next Generation Very Large Array: A Technical Overview,” 2018.

[24] J. L. Jonas, “MeerKAT,” in *2019 URSI Asia-Pacific Radio Science Conference (AP-RASC)*, 2019, pp. 1–1.

[25] S. W. Schediwy, D. R. Gozzard, S. Stobie, C. Gravestock, T. Pulbrook, R. Whitaker *et al.*, “Phase Synchronization for the Mid-Frequency Square

Kilometre Array Telescope," in *2018 2nd URSI Atlantic Radio Science Meeting (AT-RASC)*, 2018, pp. 1–1.

[26] Smirnov, O. M., "Revisiting the radio interferometer measurement equation - i. a full-sky jones formalism," *A&A*, vol. 527, p. A106, 2011. [Online]. Available: https://doi.org/10.1051/0004-6361/201016082

[27] U. Rau, S. Bhatnagar, M. A. Voronkov, and T. J. Cornwell, "Advances in Calibration and Imaging Techniques in Radio Interferometry," *Proceedings of the IEEE*, vol. 97, no. 8, pp. 1472–1481, 2009.

[28] A. R. Offringa, F. Mertens, S. van der Tol, B. Veenboer, B. K. Gehlot, L. V. E. Koopmans, and M. Mevius, "Precision requirements for interferometric gridding in the analysis of a 21 cm power spectrum," *Astronomy & Astrophysics*, vol. 631, p. A12, Oct 2019. [Online]. Available: http://dx.doi.org/10.1051/0004-6361/201935722

[29] T. J. Cornwell, K. Golap, and S. Bhatnagar, "The Noncoplanar Baselines Effect in Radio Interferometry: The W-Projection Algorithm," *IEEE Journal of Selected Topics in Signal Processing*, vol. 2, no. 5, pp. 647–657, 2008.

[30] S. Bhatnagar, T. J. Cornwell, K. Golap, and J. M. Uson, "Correcting direction-dependent gains in the deconvolution of radio interferometric images," *Astronomy & Astrophysics*, vol. 487, no. 1, p. 419–429, Jun 2008. [Online]. Available: http://dx.doi.org/10.1051/0004-6361:20079284

[31] J. A. Högbom, "Aperture Synthesis with a Non-Regular Distribution of Interferometer Baselines," *Astronomy & Astrophysics, Supplement*, vol. 15, p. 417, Jun. 1974.

[32] B. G. Clark, "An efficient implementation of the algorithm 'CLEAN'," *Astronomy & Astrophysics*, vol. 89, no. 3, p. 377, Sep. 1980.

[33] F. Schwab, "Relaxing the isoplanatism assumption in self-calibration; applications to low-frequency radio interferometry," *The Astronomical Journal*, vol. 89, pp. 1076–1081, 1984.

[34] T. J. Cornwell, "Multiscale CLEAN Deconvolution of Radio Synthesis Images," *IEEE Journal of Selected Topics in Signal Processing*, vol. 2, no. 5, p. 793–801, Oct 2008. [Online]. Available: http://dx.doi.org/10.1109/JSTSP.2008.2006388

[35] A. R. Offringa and O. Smirnov, "An optimized algorithm for multiscale wideband deconvolution of radio astronomical images," *Monthly Notices of the Royal Astronomical Society*, vol. 471, no. 1, p. 301–316, Jun 2017. [Online]. Available: http://dx.doi.org/10.1093/mnras/stx1547

[36] J. W. Rich, W. J. G. de Blok, T. J. Cornwell, E. Brinks, F. Walter, I. Bagetakos, and R. C. Kennicutt, "Multi-Scale CLEAN: A comparison of its performance against classical CLEAN in galaxies using THINGS," *The Astronomical Journal*, vol. 136, no. 6, p. 2897–2920, Nov 2008. [Online]. Available: http://dx.doi.org/10.1088/0004-6256/136/6/2897

[37] J. Choquette, W. Gandhi, O. Giroux, N. Stam, and R. Krashinsky, "NVIDIA A100 Tensor Core GPU: Performance and Innovation," *IEEE Micro*, vol. 41, no. 2, pp. 29–35, 2021.

[38] L. Polok and P. Smrz, "Increasing Double Precision Throughput on NVIDIA Maxwell GPUs," in *Proceedings of the 24th High Performance Computing Symposium*, ser. HPC '16. San Diego, CA, USA: Society for Computer Simulation International, 2016. [Online]. Available: https://doi.org/10.22360/SpringSim.2016.HPC.032

[39] I. C. Society, "IEEE Standard for Binary Floating-Point Arithmetic," *ANSI/IEEE Std 754-1985*, pp. 1–20, 1985.

[40] ——, "IEEE Standard for Binary Floating-Point Arithmetic (revision 2008)," *ANSI/IEEE Std 754-1985*, pp. 1–20, 2008.

[41] G. Tagliavini, S. Mach, D. Rossi, A. Marongiu, and L. Benin, "A transprecision floating-point platform for ultra-low power computing," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2018, pp. 1051–1056.

[42] A. V. Cueva, "Code Sample: Intel® Deep Learning Boost New Deep Learning Instruction bfloat16 - Intrinsic Functions," https://software.intel.com/content/www/us/en/develop/articles/intel-deep-learning-boost-new-instruction-bfloat16.html?wapkw=bfloat16, 2020.

[43] S. Wang and P. Kanwar, "BFloat16: The secret to high performance on Cloud TPUs," https://cloud.google.com/blog/products/ai-machine-learning/bfloat16-the-secret-to-high-performance-on-cloud-tpus, 2020.

[44] Seehyun Kim, Ki-Il Kum, and Wonyong Sung, "Fixed-point optimization utility for C and C++ based digital signal processing programs," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 45, no. 11, pp. 1455–1464, 1998.

[45] M. Nguyen, N. Serafin, and J. C. Hoe, "Partial Reconfiguration for Design Optimization," in *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*, 2020, pp. 328–334.

[46] L. Qiao, G. Luo, W. Zhang, and M. Jiang, "FPGA Acceleration of Ray-Based Iterative Algorithm for 3D Low-Dose CT Reconstruction," in *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*, 2020, pp. 98–102.

[47] G. Flegar, F. Scheidegger, V. Novaković, G. Mariani, A. E. Tom´ s, A. C. I. Malossi, and E. S. Quintana-Ortí, "FloatX: A C++ Library for Customized Floating-Point Arithmetic," *ACM Transactions on Mathematical Software (TOMS)*, vol. 45, no. 4, Dec. 2019. [Online]. Available: https://doi.org/10.1145/3368086

[48] Gustafson and Yonemoto, "Beating Floating Point at Its Own Game: Posit Arithmetic," *Supercomput. Front. Innov.: Int. J.*, vol. 4, no. 2, p. 71–86, Jun. 2017. [Online]. Available: https://doi.org/10.14529/jsfi170206

[49] L. Forget, Y. Uguen, and F. De Dinechin, "Hardware cost evaluation of the posit number system," in *Compas'2019 - Conférence d'informatique en Parallélisme, Architecture et Système*, Anglet, France, Jun. 2019, pp. 1–7. [Online]. Available: https://hal.inria.fr/hal-02131982

[50] E. T. L. Omtzigt, P. Gottschling, M. Seligman, and W. Zorn, "Universal Numbers Library: design and implementation of a high-performance reproducible number systems library," *arXiv:2012.11011*, 2020.

[51] ——, "Universal: a header-only C++ template library for universal number arithmetic," https://github.com/stillwater-sc/universal.

[52] W.-F. Chiang, M. Baranowski, I. Briggs, A. Solovyev, G. Gopalakrishnan, and Z. Rakamarić, "Rigorous Floating-Point Mixed-Precision Tuning," *SIGPLAN Not.*, vol. 52, no. 1, p. 300–315, Jan. 2017. [Online]. Available: https://doi.org/10.1145/3093333.3009846

[53] M. O. Lam and J. K. Hollingsworth, "Fine-grained floating-point precision analysis," *The International Journal of High Performance Computing Applications*, vol. 32, pp. 231 – 245, 2018.

[54] A. Borghesi, G. Tagliavini, M. Lombardi, L. Benini, and M. Milano, "Combining Learning and Optimization for Transprecision Computing," in *Proceedings of the 17th ACM International Conference on Computing Frontiers*, ser. CF '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 10–18. [Online]. Available: https://doi.org/10.1145/3387902.3392615

[55] S. Cherubin, D. Cattaneo, M. Chiari, A. D. Bello, and G. Agosta, "TAFFO: Tuning Assistant for Floating to Fixed Point Optimization," *IEEE Embedded Systems Letters*, vol. 12, no. 1, pp. 5–8, 2020.

[56] R. DiCecco, L. Sun, and P. Chow, "FPGA-based training of convolutional neural networks with a reduced precision floating-point library," in *2017 International Conference on Field Programmable Technology (ICFPT)*, 2017, pp. 239–242.

[57] D. B. Thomas, "Templatised Soft Floating-Point for High-Level Synthesis," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2019, pp. 227–235.

[58] ——, "Compile-Time Generation of Custom-Precision Floating-Point IP using HLS Tools," in *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*, 2019, pp. 192–193.

[59] "HLS Custom-Precision Floating-Point Library," https://github.com/dicecco1/fpga_cpfp.

[60] D. B. Thomas, "Templatised Soft Floating-Point for High-Level Synthesis," https://github.com/template-hls/template-hls-float.

[61] F. de Dinechin, "Reflections on 10 years of FloPoCo," in *26th IEEE Symposium of Computer Arithmetic (ARITH-26)*, Jun. 2019.

[62] "Casacore," https://github.com/casacore/casacore.

[63] "Dysco," https://github.com/aroffringa/dysco.

[64] "Image Domain Gridding," https://git.astron.nl/RD/idg.

[65] "WSClean," https://gitlab.com/aroffringa/wsclean.

[66] ASTRON. [Online]. Available: https://support.astron.nl/LOFARImagingCookbook/tutorial.html

[67] "LOFAR Long Term Archive," https://lta.lofar.eu/.

[68] "Kstars," https://docs.kde.org/trunk5/en/extragear-edu/kstars/tool-fitsviewer.html.

[69] "FloatX," https://github.com/oprecomp/FloatX.

[70] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[71] O. S. Faragallah, H. El-Hoseny, W. El-Shafai, W. A. El-Rahman, H. S. El-Sayed, E.-S. M. El-Rabaie *et al.*, "A Comprehensive Survey Analysis for Present Solutions of Medical Image Fusion and Future Directions," *IEEE Access*, vol. 9, pp. 11 358–11 371, 2021.

[72] Intel, "Intel Stratix 10 Variable Precision DSP Blocks Overview ," https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01267-fpgas-enable-high-performance-floating-point.pdf.

[73] Xilinx, "Large FPGA Methodology Guide," https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/ug872_largefpga.pdf.

[74] ——, "Xilinx Alveo U50," https://www.xilinx.com/products/boards-and-kits/alveo/u50.html.

[75] ——, "Vitis Unified Software Development Platform 2020.2 Documentation," https://www.xilinx.com/html_docs/xilinx2020_2/vitis_doc/devckernels.html.

[76] B. Gregg, "Performance counters," *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2018. [Online]. Available: http://www.brendangregg.com/perf.html

[77] NVIDIA, "CUDA toolkit documentation - nvprof," https://docs.nvidia.com/cuda/profiler-users-guide/index.html.

[78] AMD, "AMD CodeXL," https://github.com/GPUOpen-Archive/CodeXL.

[79] "PowerSensor Library," https://gitlab.com/astron-misc/libpowersensor.

[80] Xilinx, "Performance and Resource Utilization for Floating-point v7.1." [Online]. Available: https://china.xilinx.com/html_docs/ip_docs/pru_files/floating-point.html

[81] E. Calore and S. F. Schifano, "Performance assessment of fpgas as hpc accelerators using the fpga empirical roofline," in *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*, 2021, pp. 83–90.

[82] Samsung. [Online]. Available: https://www.samsung.com/us/sas/Business/Foundry14nm

[83] TSMC. [Online]. Available: https://www.tsmc.com/english

[84] GlobalFoundries. [Online]. Available: https://gf.com/

[85] "Nvidia 387.12 breaks power reading in nvidia-smi," https://forums.developer.nvidia.com/t/nvidia-387-12-breaks-power-reading-in-nvidia-smi/53946/21.

[86] S. Sarangi and B. Baas, "DeepScaleTool: A Tool for the Accurate Estimation of Technology Scaling in the Deep-Submicron Era," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5.

[87] "clpeak," https://github.com/krrishnarraj/clpeak.

[88] "HLS Pragmas," https://www.xilinx.com/html_docs/xilinx2021_1/vitis_doc/hls_pragmas.html#sxx1504034358866.

[89] Y. Hu, Y. Du, E. Ustun, and Z. Zhang, "GraphLily: Accelerating Graph Linear Algebra on HBM-Equipped FPGAs," *Power (Watts)*, vol. 268, no. 182, p. 49.

[90] G. Daughtry, "Top 5 Timing Closure Techniques," https://www.xilinx.com/publications/prod_mktg/club_vivado/presentation-2015/paris/Xilinx-TimingClosure.pdf.

[91] N. Brown, "Weighing Up the New Kid on the Block: Impressions of using Vitis for HPC Software Development," in *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*, 2020, pp. 335–340.

[92] N. Zhang, X. Chen, and N. Kapre, "Rapidlayout: Fast hard block placement of fpga-optimized systolic arrays using evolutionary algorithms," in *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*, 2020, pp. 145–152.

[93] A. Li, S. L. Song, M. Wijtvliet, A. Kumar, and H. Corporaal, "SFU-Driven Transparent Approximation Acceleration on GPUs," in *Proceedings of the 2016 International Conference on Supercomputing*, ser. ICS '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2925426.2926255

[94] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[95] G. Ofenbeck, R. Steinmann, V. Caparros, D. G. Spampinato, and M. Püschel, "Applying the roofline model," in *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2014, pp. 76–85.

[96] GPUOpen, "AMD RDNA™ 2 Performance Guide - GPUOpen."

[97] B. Veenboer, M. Petschow, and J. W. Romein, "Image-Domain Gridding on Graphics Processors," in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2017, pp. 545–554.

[98] N. M. Gürel, K. Kara, A. Stojanov, T. Smith, T. Lemmin, D. Alistarh *et al.*, "Compressive Sensing Using Iterative Hard Thresholding With Low Precision Data Representation: Theory and Applications," *IEEE Transactions on Signal Processing*, vol. 68, pp. 4268–4282, 2020.

[99] J. Hou, Y. Zhu, S. Du, S. Song, and Y. Song, "FPGA-Based Scale-Out Prototyping of Degridding Algorithm for Accelerating Square Kilometre Array Telescope Data Processing," *IEEE Access*, vol. 8, pp. 15 586–15 597, 2020.

[100] P. Colangelo, N. Nasiri, E. Nurvitadhi, A. Mishra, M. Margala, and K. Nealis, "Exploration of Low Numeric Precision Deep Learning Inference Using Intel® FPGAs," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2018, pp. 73–80.

[101] S. Daghaghi, N. Meisburger, M. Zhao, Y. Wu, S. Gobriel, C. Tai, and A. Shrivastava, "Accelerating SLIDE Deep Learning on Modern CPUs: Vectorization, Quantizations, Memory Optimizations, and More," 2021.

[102] T. Nguyen, S. Williams, M. Siracusa, C. MacLean, D. Doerfler, and N. J. Wright, "The performance and energy efficiency potential of fpgas in scientific computing," in *2020 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, 2020, pp. 8–19.

[103] E. Vermij, L. Fiorin, R. Jongerius, C. Hagleitner, and K. Bertels, "Challenges in exascale radio astronomy: Can the SKA ride the technology wave?" *The International Journal of High Performance Computing Applications*, vol. 29, no. 1, pp. 37–50, 2015. [Online]. Available: https://doi.org/10.1177/1094342014549059

[104] Y.-k. Choi, Y. Chi, W. Qiao, N. Samardzic, and J. Cong, "HBM Connect: High-Performance HLS Interconnect for FPGA HBM," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 116–126. [Online]. Available: https://doi.org/10.1145/3431920.3439301

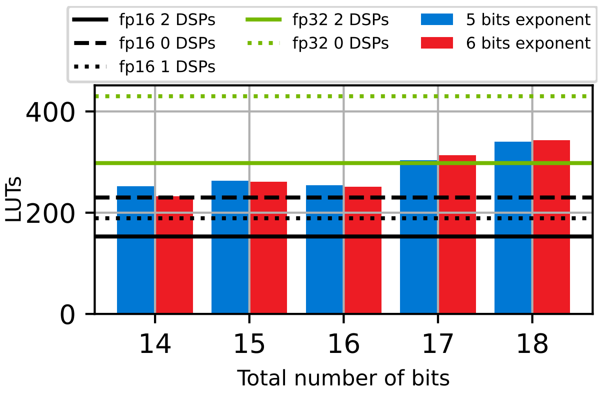
[105] B. Gaide, D. Gaitonde, C. Ravishankar, and T. Bauer, "Xilinx Adaptive Compute Acceleration Platform: Versal<sup>TM</sup> Architecture," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 84–93. [Online]. Available: https://doi.org/10.1145/3289602.3293906

[106] "Xilinx's Arbitrary Precision Data Types Library," https://www.xilinx.com/html_docs/xilinx2020_2/vitis_doc/ogi1585574074269.html.

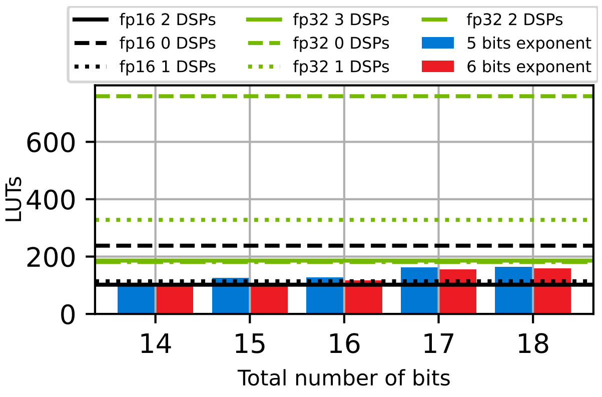
APPENDIX A HLS CUSTOM FLOATING-POINT LIBRARY COMPARISON

The main custom floating-point libraries for High-Level-Synthesis are the custom-precision floating-point (CPFP) [59], and the templatised soft floating-point for high-level synthesis (THLS) [60]. Both the libraries are based on the Xilinx arbitrary precision library [106]. While CPFP support homogeneous custom floating-point, THLS enables heterogeneous custom floating-point, which may be helpful for fine-grained mixed-precision or compute operations at higher precision and then truncate the result. Since we do not not need heterogeneous operators for this work the CPFP library would be sufficient. However,we evaluate the resource usage of the two libraries for the adder (see *Figure 18*) and multiplier operator (see *Figure 19*) to motivate our choice of THLS over CPFP.

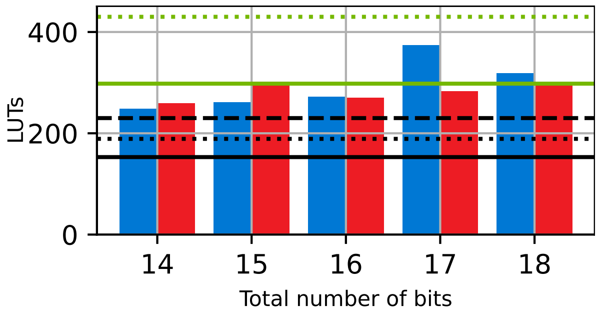
Both the operators use less resources compared to single-precision floating-point for the presented precision, including the one employed in our highest performing accelerator (6 bits exponent and 11 bits mantissa, with a total of 18 bits). Unlike the adder operator, the multiplier operator, considering the same precision and number of DSPs (1), has similar resource usage (less in the case of THLS) than half-precision. Overall THLS has reduced LUTs, e.g. *Figure 19(b)* vs *Figure 19(a)*, and FFs, e.g. *Figure 19(d)* vs *Figure 19(c)*, usage compared to CPFP. This analysis thus motivates the usage of THLS over CPFP.



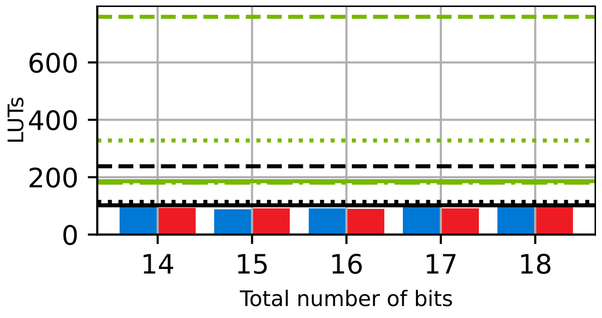
(a) LUTs usage CPFP adder.



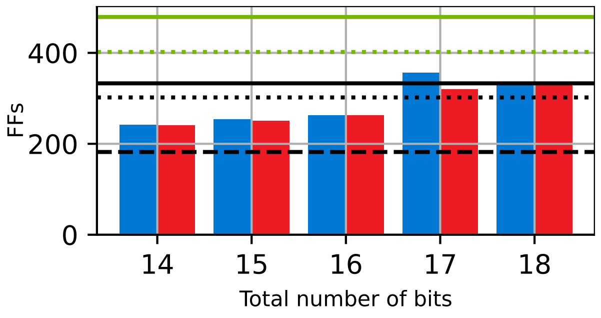
(a) LUTs usage CPFP multiplier.



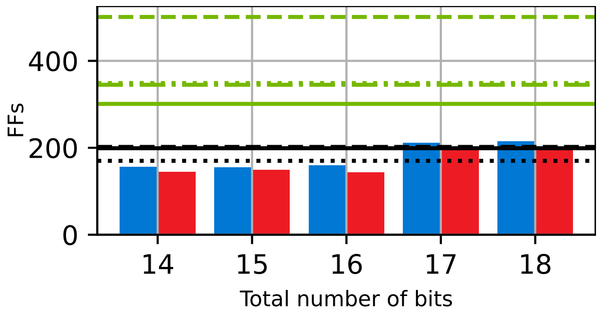
(b) LUTs usage THLS adder.



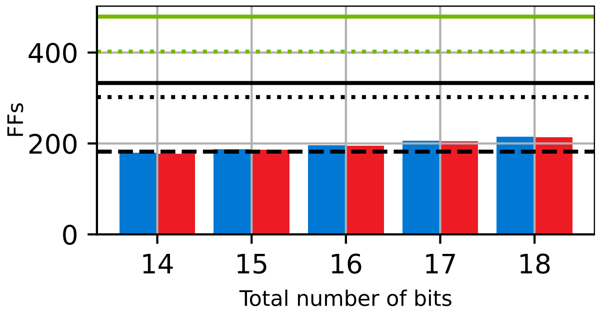
(b) LUTs usage THLS multiplier.



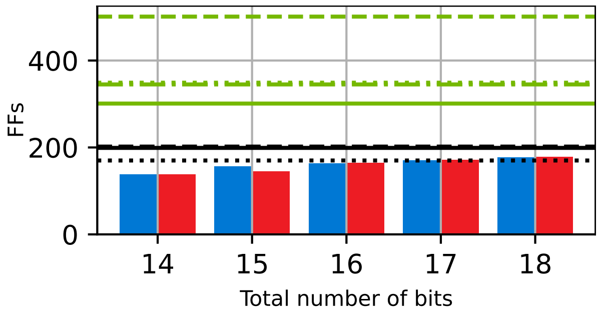
(c) FFs usage CPFP adder.



(c) FFs usage CPFP multiplier.



(d) FFs usage THLS adder.



(d) FFs usage THLS multiplier.

FIGURE 18: Custom floating-point adder (0 DSPs) resource usage for the CPFP and THLS libraries.

FIGURE 19: Custom floating-point multiplier (1 DSPs) resource usage for the CPFP and THLS libraries.

STEFANO CORDA is a PhD candidate in the Electronic Systems group at the Department of Electrical Engineering of Eindhoven University of Technology. He is also a guest PhD at the Chair of Processor Design of Dresden Technical University. He received his BSc in Electrical and Electronic Engineering in 2015 and his MSc in Electronic Engineering in 2017 from the University of Cagliari. He started to pursue his PhD in October 2017. He visited IBM Zürich and Dresden Technical University during his PhD as a guest researcher. His research interests include high-performance computing, FPGA design, and application profiling.

BRAM VEENBOER is researcher at the HPC group of the Innovation & Systems department of The Netherlands Institute for Radio Astronomy (ASTRON). He received his Ph.D. in computer science at the Vrije Universiteit in Amsterdam, on radio-astronomical imaging using accelerator hardware. His current interests are on high-performance imaging for the Low Frequency Array (LOFAR) and the upcoming Square Kilometre Array (SKA), the development of novel algorithms (for imaging, but also for time-domain radio-astronomy), and the development of a real-time processing pipeline for all-sky imaging.

AHSAN JAVED AWAN is a Senior Researcher of Future Computing Platforms at Ericsson Research. Prior to joining Ericsson in 2018, he worked at Imperial College London, IBM Research – Tokyo in Japan, and Barcelona Supercomputing Center in Spain. Awan holds an Erasmus Mundus joint Ph.D. from KTH Royal Institute of Technology in Stockholm, Sweden and the Universitat Politècnica de Catalunya in Barcelona, Spain, for his work on performance characterization and optimization of in-memory data analytics on a scale-up server. He also holds Erasmus Mundus Joint Masters Degree from TU Kaiserslautern, Germany, University of Southampton, UK and NTNU Norway. He has written 20+ scientific papers (including two best paper awards), filed 14 patent applications and served as TPC member of international conferences

JOHN W. ROMEIN John Romein is researcher at ASTRON, where he leads several projects on HPC research for radio-astronomical applications. His main interest is accelerated computing, using GPUs, FPGAs, and other accelerators. John received his Ph.D. in computer science at the Vrije Universiteit in Amsterdam, on distributed game-tree search.

ROEL JORDANS is an Assistant Professor in the Electronic Systems group of the Electrical Engineering Department at Eindhoven University of Technology (TU/e) and R&D manager at the Radboud RadioLab of the Radboud University Nijmegen. His research interests include developing hardware architectures for reliable, high-performance systems and compilation techniques for, and design automation applied to, application-specific processors.

AKASH KUMAR (SM'13) received the joint Ph.D. degree in electrical engineering and embedded systems from the Eindhoven University of Technology, Eindhoven, The Netherlands, and the National University of Singapore (NUS), Singapore, in 2009. From 2009 to 2015, he was with NUS. He is currently a Professor with Technische Universität Dresden, Dresden, Germany, where he is directing the Chair for Processor Design. His current research interests include the design, analysis, and resource management of low-power and fault-tolerant embedded multiprocessor systems.

ALBERT-JAN BOONSTRA received the MSc degree in applied physics at Groningen University (NL) in 1987, and the PhD degree at Delft University of Technology (NL) in 2005. He started his career at SRON Groningen where he conducted cryogenic optical and infrared tests on the SWS spectrometer of the infrared astronomical satellite ISO. In 1992 he joined ASTRON, the Netherlands Institute for Radio Astronomy in Dwingeloo. He coordinated the upgrade of the Westerbork Synthesis Radio Telescope (WSRT), after which he investigated temporal and spatial filtering techniques for suppressing radio interference at the WSRT. From 2012-2017 he led the Dome project at ASTRON, aimed at developing exascale technologies for the Square Kilometre Array. Albert-Jan has led several groups and projects at ASTRON, and is currently Programme Manager Technical Research at the Innovation and Systems Department of ASTRON. His scientific interests lie in the area of signal processing, scientific computing, and artificial intelligence.

HENK CORPORAAL is Professor in Embedded System Architectures at the Eindhoven University of Technology (TU/e) in The Netherlands. He has gained a MSc in Theoretical Physics from the University of Groningen, and a PhD in Electrical Engineering, in the area of Computer Architecture, from Delft University of Technology. Corporaal has co-authored over 500 journal and conference papers. Furthermore he invented a new class of VLIW architectures, the Transport Triggered Architectures, which is used in several commercial products, and by many research groups. He also initatiated and leads the Dutch NWO perspectief program on Efficient Deep Learning; in this program many research institutes and over 30 companies participate. His research is on low power multiprocessor, heterogenous processing architectures, their programmability, and the predictable design of soft- and hard real-time systems. This includes research and design of embedded system architectures, including CGRAs, SIMD, VLIW and GPUs, on accelerators, the exploitation of all kinds of parallelism, fault-tolerance, approximate computing, architectures for machine and deep learning, optimizations and mapping of deep learning networks, and the (semi-)automated mapping of applications to these architectures. For further details see corporaal.org.