HAL open science

On-board Payload Data Processing Combined with the Roofline Model for Hardware/Software Design

Seungah LEE, Emmanuel Casseau, Angeliki Kritikakou, Olivier Sentieys, Ruben Salvador, Julien Galizzi

To cite this version:

Seungah LEE, Emmanuel Casseau, Angeliki Kritikakou, Olivier Sentieys, Ruben Salvador, et al.. On-board Payload Data Processing Combined with the Roofline Model for Hardware/Software Design. AeroConf 2024 - IEEE Aerospace Conference, Mar 2024, Big Sky, Montana, United States. pp.1-12, 10.1109/AERO58975.2024.10521057 . hal-04423185

HAL Id: hal-04423185

https://inria.hal.science/hal-04423185v1

Submitted on 29 Jan 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

On-board Payload Data Processing Combined with the Roofline Model for Hardware/Software Design

Seungah Lee, Emmanuel Casseau, Angeliki Kritikakou, Olivier Sentieys
Univ Rennes, Inria, CNRS, IRISA
France
firstname.surname@irisa.fr

Ruben Salvador
CentraleSupélec, Inria, Univ Rennes, CNRS, IRISA
Rennes, France
ruben.salvador@inria.fr

Julien Galizzi
CNES (French Space Agency)
France
julien.galizzi@cnes.fr

Abstract—**High-performance on-board payload data processing has become more interesting with the development of radiation-hardened multiprocessor System-on-chip (MPSoC). As recent space-qualified MPSoCs include Arm Central Processing Units (CPUs) and Field Programmable Gate Arrays (FPGAs), an efficient design method is required to deal with complex heterogeneous embedded systems. Both data bit-width (data accuracy) and processing performance are important in astronomy, thus the design methodology should concern application-specific Multi-Objective Optimization Problems (MOOPs). This paper proposes to combine the roofline performance model with Design Space Exploration (DSE) of hardware/software designs as a methodology. We use High-Level Synthesis (HLS) for FPGA design to configure different hardware architectures based on C/C++ and pragmas. We develop a benchmark for payload data processing on Arm CPUs and embedded FPGA on a heterogeneous MPSoC by adapting open-source libraries for one of the most commonly used algorithms to provide validated libraries to payload teams. The benchmark takes as constraints the SVOM ECLAIRs payload requirements, and as input data the CCSDS test images, executes applications, and verifies output data. We chose an AMD-Xilinx Zynq UltraScale+ evaluation board and the two-Dimensional Fast Fourier Transform (2-D FFT) as a DSE use case. We designed the benchmark on an Arm Cortex-A53 in bare-metal and an embedded FPGA based on Vitis HLS. The results show a customized roofline model with the hardware/software design. The implemented design has 1.6-55 times faster performance compared to the payload execution time requirement. Based on the proposed roofline model and the DSE results, future payload teams can study the trade-off between execution time and area efficiency to select the most suitable implementation.**

TABLE OF CONTENTS

1. INTRODUCTION 1
2. DSE METHODOLOGY CONSIDERING THE ROOFLINE MODEL 3
3. BENCHMARK FOR HW/SW DESIGN 5
4. THEORETICAL ROOFLINE MODEL FOR HW/SW DESIGNS 6
5. USE-CASE: 2-D FFT HW/SW DESIGN WITH THE COMMON ROOFLINE MODEL 8
6. CONCLUSION 11
ACKNOWLEDGMENTS 11
REFERENCES 11
BIOGRAPHY 14

1. INTRODUCTION

High-performance space-grade data processing platforms have been developed with the rapid growth of System-on-Chip (SoC) technology. Several space agencies and companies have developed new SoCs and On-Board Computers (OBCs) to prepare for future missions. Following this trend, space mission payload teams can consider complex on-board data processing. As a first step, we analyzed on-board data processing platforms, which can be used for spacecraft and payload data processing. Note that, in this context, we define the data processing platform as the hardware device where applications or kernels can be executed, and not as a satellite platform that offers the complete infrastructure for the mission and the support for instruments in space engineering. Table 1 shows the analyzed data processing platforms, which are radiation-hardened or have at least defense-grade product models. Furthermore, we do not focus on CubeSat missions because these missions often use simpler platforms based only on micro-controllers, such as Arm cortex-M or TI MSP430 [1]. Many launched missions have used LEON processors and an RTAX Field-Programmable Gate Array (FPGA) [2]. However, the Arm Cortex series is a newly emerging CPU option. Moreover, recent platforms are characterized as MultiProcessor System-on-Chip (MPSoC), e.g., AMD-Xilinx Versal and NanoXplore NG-Ultra, which integrate an FPGA with CPU-based processors. These recent MPSoC platforms have an advantage in hardware/software (HW/SW) design, whereas they integrate heterogeneous processors, widely used in space

979-8-3503-0462-6/24/$31.00 ©2024 IEEE
2024 IEEE Aerospace Conference - www.aeroconf.org

Table 1. On-board processing platforms for space missions

Manufacturer (Alphabetic order)	Platform	Space-qualified*	Description
AMD-Xilinx	Versal	✓	Adaptive MPSoC: dual-core Arm Cortex-A72, dual-core Arm Cortex-R5F, embedded FPGA, Network-on-Chip (NoC)
	Zynq UltraScale+	N/A	MPSoC: quad-core Arm Cortex-A53, dual-core Arm Cortex-R5F, embedded FPGA, GPU The defense-grade product has potential for proton based missions.
	Zynq 7000	N/A	MPSoC: dual-core Cortex-A9, embedded FPGA The defense-grade product, flight heritage for Cubesat missions
	Kintex UltraScale	✓	FPGA
	Virtex-5, Virtex 4	✓	FPGA
BAE Systems	RAD5545	in process**	SoC: quad-core RAD5500 cores
	RAD750	✓	PowerPC microprocessor
	RH1280B	✓	FPGA
	RH1020B	✓	FPGA
Frontgrade Gaisler	GR740	✓	SoC: Quad-Core LEON4FT SPARC V8 Processor
	GR712RC	✓	Dual-Core LEON3FT SPARC V8 Processor
Microchip	RT PolarFire	in process**	FPGA
	RTG4	✓	FPGA
	RTAX / RTAX-DSP	✓	FPGA
	RTSX-SU	✓	FPGA
NanoXplore	NG-ULTRA	in process**	MPSoC: quad-core Arm Cortex-R52, embedded FPGA
	NG-Large	in process**	MPSoC: single-core Arm Cortex-R5, embedded FPGA
	NG-Medium	✓	FPGA
NASA & Microchip	HPSC	under development	SoC: Multi-core RISC-V

* Microcircuits shall conform to ECSS QPL or MIL-PRF-38535 or QML.
** The results of radiation-tolerance test are currently available.

missions, into the same chip [3, 4]. Besides, the Arm processors have already been used in industry and academia, and there are mature open-source compilers and tools to support their use. In this study, we focus on recently released MPSoC platforms based on the specifications of AMD-Xilinx and NanoXplore products.

Most on-board data processing research has currently focused on spacecraft platforms, such as Attitude and Orbit Control Systems (AOCS) and On-Board Data Handling (OBDH), rather than payload data processing. Payload data processing has been traditionally performed by the data processing pipelines on the ground with the collaboration of the operation center and data processing consortium or data center [5, 6]. However, due to modern hardware platforms such as MPSoCs, it is expected a partial migration of the data processing pipeline on-board. Note that, a comprehensive study dedicated to on-board payload data processing, covering payload-oriented algorithm types, design methodology, and implementation examples, is missing. A thorough and collective understanding of common function blocks of payload data processing can be helpful for payload teams. Therefore, as a second step, we performed a survey on payload data processing in order to identify and rank the key compute-intensive algorithms. To achieve that, we perform interviews with French scientific payload mission teams working on radar, space telescopes, magnetometers, mutual impedance probes, etc. According to our survey, the Fast Fourier Transform (FFT) compression algorithm is used the most frequently by all payload types. The interpolation and filter algorithms, such as Cascaded Integrator–Combo (CIC), Infinite Impulse Response (IIR), and Kalman filter, are ranked next. Moreover, the dimension of the FFT, i.e., the one-dimensional (1-D) FFT or the 2-D FFT, can depend on the type of payloads, i.e., whether they deal with 1-D signal or 2-D image or signal. Especially for the implementation of Synthetic Aperture Radar (SAR) data processing, the dimension of the FFT depends on the implemented algorithm [7]. Currently, for the most launched missions, FFT is implemented in non-optimized 1-D FFT, and the FFT size is small, e.g., 256-point FFT [8]. Otherwise, the implementations are in-house or patented [9], and thus, not many validated open-source codes, dedicated to payload data processing, exist. It is expected that the requirements for 1-D FFT and 2-D FFT with larger FFT size will increase [10, 11] for future missions. Therefore, it is essential to provide validated optimized 2-D FFT benchmarks, which can cover diverse FFT sizes.

We should highlight that the lack of on-board payload data processing studies can be related to the fact that most payload teams consist of scientists who are familiar with software-oriented programming, and not specialized in hardware design, including FPGAs. Consequently, only a few teams with long-time experience have participated in developing on-board payload data processing units, typically called Data Processing Unit (DPU), with the goal of covering most scientific missions. Furthermore, payload teams focus on ground data processing, since there are various astronomy-friendly programming tools and libraries convenient to astronomical data processing, e.g., Python with the Astropy library, MATLAB with AstroPack package, IDL, and Fortran. In addition, it is not easy for payload teams to design on-board processing, because the requirements change depending on the mission type and the subsystem. However, emerging hardware programming tools have appeared with the goal of

enabling users with software-based knowledge to develop hardware easily. One such example is High-Level Synthesis (HLS), which converts C/C++ language-based behavioral descriptions into a traditional Hardware Description Language (HDL), such as Verilog or VHDL. Therefore, we take advantage of HLS and present a Design Space Exploration (DSE) methodology, combined with the roofline performance model, to propose different on-board design options for the most commonly compute-intensive algorithms for payload data processing.

2. DSE METHODOLOGY CONSIDERING THE ROOFLINE MODEL

HLS for hardware design

HLS is a hardware design process that automatically converts high-level behavioral specifications, usually written in C/C++, into RTL design [12]. Several HLS tools exist, with either commercial or academic licenses: AMD-Xilinx Vitis HLS, Siemens Catapult HLS, Bambu HLS, etc. [13]. Fig. 1 shows examples of FPGA design development flows focusing on Catapult HLS and Vitis HLS, which are currently representative commercial HLS tools, compared to traditional HDLs, such as VHDL and Verilog. Catapult HLS and Vitis HLS both support C++-based HLS, and the results of HLS synthesis are integrated into their own implementation tools, such as Siemens Precision Synthesis [14] and AMD-Xilinx Vitis [15]. With the help of these tools, users can implement designs on a target FPGA. The difference between Catapult HLS and Vitis HLS is that Vitis HLS only supports the AMD-Xilinx FPGA-based platforms, whereas Catapult HLS supports multi-vendor platforms, such as NanoXplore NG-Ultra [16].

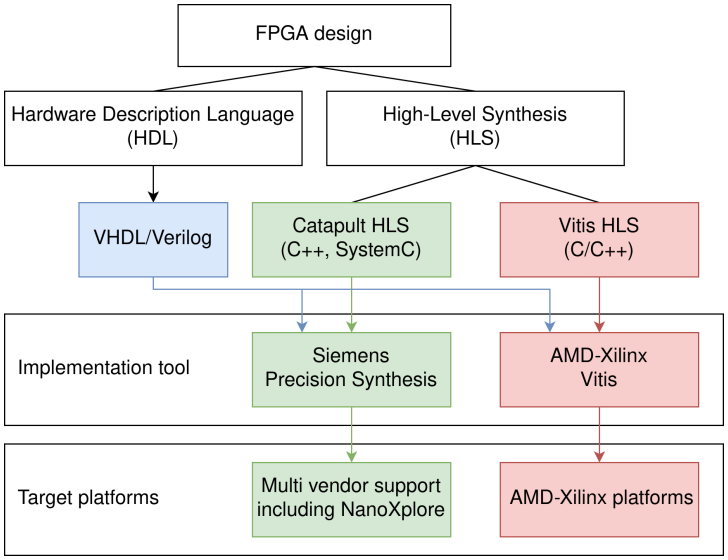

Figure 1. FPGA design development flow

In this work, we use the Vitis HLS tool as an example to explain the HLS tool development flow. The Vitis HLS tool provides four steps for the synthesis and simulation process, as shown in Fig.2: C simulation, C synthesis, C/Register-transfer level (RTL) co-simulation, and RTL synthesis [17]. The essential step of the Vitis HLS tool is to perform C synthesis and RTL export. C synthesis takes input C/C++ code annotated with HLS pragmas and directives and, using the hardware specifications of the platform that are also provided as inputs, and converts it into RTL code, such as VHDL or Verilog. Each pragma and directive affects the hardware architecture design directly. This way, developers can easily generate a number of different designs by modifying their values. As a result, HLS has recently become popular, compared to traditional RTL design, where the designers need to spend a significant amount of time modifying lower-level HDL code each time. This characteristic makes HLS suitable for performing massive DSE easily. Even if C synthesis generates RTL files, the generated RTL files cannot be directly used for AMD-Xilinx platforms, as the RTL export is required in order to make the generated RTL files suitable to AMD tools, such as Vivado (IP file) and Vitis (XO file). Users can integrate an IP or XO file at the system level and combine these files with a software application to create a full HW/SW design. Through RTL synthesis including placement and routing, performed by Vivado, precise results of resource utilization and clock frequency are obtained. Furthermore, Vitis HLS offers two simulation options: C simulation and C/RTL co-simulation. Such simulation results are useful to users for debugging and understanding the expected total latency of a given design. For example, C/RTL co-simulation provides latency in cycles, which is connected to the expected execution time. Both C simulation and C synthesis take HLS code and test-bench codes as inputs.

Roofline model for HW/SW design

The roofline model is a performance model which shows the attainable performance of a given architecture. The roofline model shows whether a design is limited by the computational performance and the memory bandwidth. The memory bandwidth is considered for off-chip memory accesses with DRAM. The classical roofline model was initially invented to target multicore CPU architectures [18]. Recently, the model has been extended to other architecture, such as Graphic Processing Units (GPUs) and FPGAs. However, the adjusted model does not consider the specific characteristics of the architectures [19, 20]. For both CPUs and GPUs, the roofline is fixed for a given architecture and does not depend on the applications/kernels executed on a

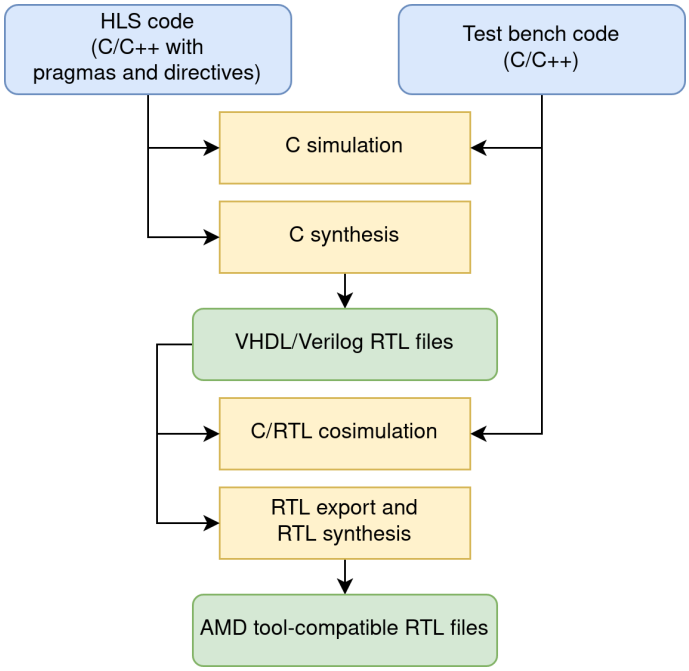

Figure 2. Vitis HLS design development flow [17]

given architecture.

However, the FPGA roofline model has different characteristics compared to the CPU/GPU roofline model. This is because FPGAs have reconfigurable characteristics [21], which allows building custom architectures adapted to the application. Depending on the specific FPGA and the kernel design, the final architecture that implements that kernel, hence the resource utilization, change. As a result, the FPGA roofline is both kernel-specific and FPGA-specific. Moreover, floating-point operations per second (Flop/sec) are typically used as a metric to compare performance for both CPUs and GPUs. Although using Flop/s as a metric is possible for FPGAs, it is probably not the best option. This is because not every FPGA has direct support for floating-point arithmetic operations. One floating-point operation can require several Digital Signal Processing (DSP) blocks in a given FPGA. Using fixed-point operations per second (op/s) can be an alternative metric, assuming the chosen fixed-point data type fits in the DSP block input. For example, the DSP block DSP48E has multiplication inputs with a size of 27×18 bits without direct floating-point support, and thus, the maximum preferred input can be 27 bits and 18 bits for a multiplication [22].

As shown in Table 1, state-of-the-art hardware platforms used in space-grade products combine CPUs and an FPGA in an SoC. To support such platforms, there is a need for a combined roofline model that integrates both CPUs and FPGAs in order to support efficient HW/SW designs. However, this is challenging as CPUs and FPGAs have different characteristics as architectures. To achieve such a common roofline model, it is required to clarify the relevant terminology and constraints. The terminology used in this work is inspired on the FPGA roofline model [20] and used for general description. For instance, the computational ceiling and the Input/Output (I/O) bandwidth ceiling are used to replace the peak-floating performance ceiling and the memory bandwidth ceiling. This change is based on revisions done to the original roofline model through the years to consider that data can come from other memories than the main off-chip DRAM. Moreover, Computational Intensity (CI) is used for the X-axis of the roofline model, as opposed to Arithmetic Intensity (AI) of the original model.

For the calculation of *computational ceiling*, we use both fixed-point operations and floating-point operations. Especially, we use the maximum fixed-point data bit-width that fits in a DSP block to increase data accuracy. For the FPGA design, we use 32 bits for floating-point (FP32) and arbitrary data type with the maximum multiplier input size for fixed-point operation. For the CPU design, we selected FP32 and FP64 for floating-point operations, reflecting that astronomical data processing typically require high precision. In addition, Single Instruction Multiple Data (SIMD) shared with Floating-point Unit (FPU) are available to embedded CPUs, as they process multiple floating-point operations. For the calculation of *I/O bandwidth ceiling*, we consider Dynamic Random Access Memory (DRAM) as external memory for CPUs and FPGA, because the on-chip memory in space-grade platforms is smaller than the expected size of image and signal used as inputs. Note that, recent space-grade DRAMs, such as Double Data Rate Synchronous Dynamic Random-Access Memory (DDR SDRAM), have recently flight heritages [23]. Therefore, the data storage during payload data processing is not required to be limited to Static Random-Access Memory (SRAM), which increases memory performance significantly [24]. Furthermore, the interface between CPUs and FPGA is considered for the FPGA-based hardware accelerator design.

To evaluate the efficiency of designs of applications and kernels, we drew them in the roofline model plane. Each design has CI

as X-axis value and performance (P) as Y-axis value [18, 20]. The CI and P are defined as:

$$CI = \frac{\text{number of operations}}{\text{external memory access in bytes}} \tag{1}$$

$$P = \frac{\text{number of operations}}{\text{execution time}} \tag{2}$$

The number of operations are the total number of arithmetic operations, such as multiplications and additions. However, the denominators of CI and P are calculated differently between a CPU and an FPGA. Section 5 presents in details how the proposed method is applied using an HLS tool for FPGA design and the measurement of execution for CPU design.

3. BENCHMARK FOR HW/SW DESIGN

Regarding the technical aspects, as recent space-qualified heterogeneous MPSoCs include Arm CPU cores and an FPGA, users need to analyze the pros and cons of each architecture to decide an efficient implementation. On the one hand, an FPGA has a significant advantage in terms of processing speed as a hardware accelerator is designed in a custom way for the application. However, there is a high effort associated with the design of application-specific hardware accelerators on FPGAs [25]. Software developers and scientists are not familiar with the knowledge of FPGA architectures. Furthermore, direct floating-point operations are commonly used in the astronomical data processing domains. Not every FPGA supports this kind of operations leading to an exponential increase in the FPGA resource utilization. On the other hand, programming skills for CPUs are more common, but the processing speed is expected to be lower than hardware accelerators on FPGAs. However, developers still need to be aware of the technical constraints of CPUs on embedded space-grade platforms. These constraints do not exist in commercial-grade personal computers, such as x86 architecture. Firstly, Arm CPU cores in embedded systems, such as Arm Cortex-A53 or Cortex-A72 or Cortex-R5, have small on-chip memory represented as L1 and/or L2 cache memory. Therefore, developers should be cautious regarding static and dynamic memory allocation. Furthermore, space applications are often built either on bare-metal environment or on real-time Operating System (OS), such as RTEMS [26]. Embedded Linux OS, such as Yocto, is emerging for space applications, but it is still under analysis [27, 28]. This work focuses on a bare-metal environment.

The non-technical aspects are not negligible when we consider international space missions. For space projects including non-US countries, the International Traffic in Arms Regulations (ITAR) and the Export Administration Regulations (EAR) must be considered for non-open-source software. For the European space missions, following the European Cooperation for Space Standardization (ECSS) documents and systems, ECSS-Q-HB-80-01A shall be considered when existing software, including the management of Software Reuse File (SRF), is used [29]. The software should be validated and verified for space missions, but many public codes under open-source licenses, such as the GPL-3.0 license or the MIT license, do not guarantee warranty or liability. Therefore, developers need a way to validate the execution results and the potential errors.

Considering these technical and non-technical constraints, it is necessary to build a benchmark that can be both easily implemented over Arm CPU cores and an FPGA and comply with space mission requirements. Recently, some works have proposed space benchmarks considering the next-generation processing architectures. For example, ESA and Barcelona Supercomputing Center (BSC) have developed the OBPMark [30] and GPU4S [31] benchmarks targeting GPUs based on the algorithms used in space applications. The benchmarks provide sequential C code and GPU-oriented codes in CUDA, OpenMP, and OpenCL. However, the benchmarks mainly consider GPUs, and thus, the provided C codes are neither directly suitable for embedded Arm CPUs and bare-metal implementation nor to HLS-based synthesis for FPGAs. The benchmarks currently use random-generated input data, whereas the verification is still under development. ESA also provides NIR HAWAII-2RG BM algorithm to benchmark on-board payload processing on multi-core processors [32]. Moreover, micro-benchmarks exist to compare performance and power characteristics of Arm Cortex-72 cores with current space-rated processors [33], such as Dhyrstones, Whetstones, and CoreMark. To the best of our knowledge, open-source embedded C/C++ benchmarks conforming with the above constraints and dedicated to the surveyed space algorithms, are not available. Therefore, we propose a HW/SW space benchmark inspired by OBPMark [30]. The requirements of the proposed benchmark are the following:

Requirement 1. The benchmark shall target a heterogeneous MPSoC, including an embedded FPGA and Arm CPU cores.

Requirement 2. The benchmark shall provide source codes dedicated to scientific payload data processing algorithms.

Requirement 3. The benchmark shall provide input and reference output data suitable for space payloads. Users can compare the reference output data with execution results for meaningful data accuracy analysis.

Requirement 4. The benchmark shall provide open-source codes for HW/SW implementation so users can use the codes for any international scientific missions. However, the commercial usage may be restricted depending on the license type.

Requirement 5. The benchmark shall provide C/C++ codes easily applicable to a bare-metal execution environment in Arm CPU cores, which means codes with low software dependencies or already optimized for Arm architectures.

Requirement 6. The benchmark shall provide HLS codes targeting FPGA design.

Requirement 7. The benchmark shall include the functionality to validate execution results by comparing the provided

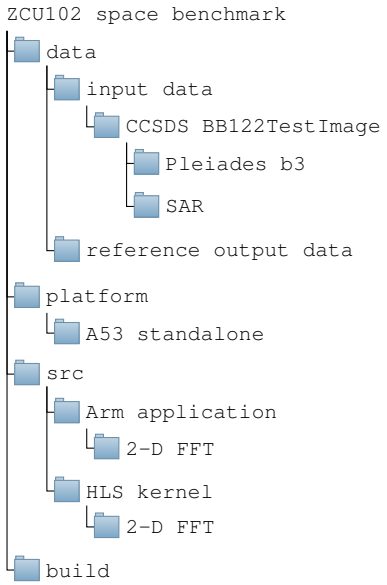

Figure 3. Structure of the payload data processing benchmark targeting the Zynq UltraScale+ platform

reference output data to the execution output data, including data accuracy analysis.

Requirement 8. The benchmark shall provide the functionality to measure the execution time and the amount of external memory access for software implementation on Arm CPU cores.

Based on the above requirements, we developed a benchmark targeting the AMD-Xilinx ZCU102 evaluation board, which embarks the AMD-Xilinx Zynq UltraScale+ heterogeneous MPSoC platform **(Requirement 1)**. The benchmark structure is shown in Figure 3. More specifically, we target Cortex-A53 cores, the embedded FPGA on the platform, and the DDR4 memory connected to Arm CPU cores as external memory.

We selected CCSDS BB122TestImage to cover both 1-D and 2-D data processing [34]. The provided image data include various payload types, image sizes, and data bit-widths, so they are proper candidates for the input data of the benchmark. For the preliminary results, we selected a simulated Pleiades near-infrared image (Pleiades b3), provided by CNES, and ERS-1 SAR image, provided by ESA. The Pleiades image consists of 12 bits/pixel, and the size is 1376×320. The SAR image has 16 bits/pixel with an image size equal to 512×512. The data type is unsigned int16 for all image data. Among the surveyed algorithms, we selected the 1-D FFT and the 2-D FFT as an initial analysis target **(Requirement 2)**. We generated reference output data for the 1-D FFT and the 2-D FFT using Matlab, which is widely used for the data processing pipelines on the ground. The data type of Matlab FFT output is a double-precision floating-point format. The extracted input data are stored as arrays, and generated reference output data are stored in files **(Requirement 3)**. Since FFT libraries frequently require input data in the power of 2, the input and reference output data are provided in cropped data and zero-padded data matching the power of 2 of the original data.

The benchmark contains only open-source library code for both C/C++ codes for Arm Cortex-A53 and Vitis HLS codes for AMD-Xilinx FPGA **(Requirement 4-6)**. Due to page limitations, we present the 2-D FFT case, which is the more complex one. In this 2-D FFT case, we adopted the open-source Ooura FFT library developed by Kyoto University [35] and created a modified version that meets the aforementioned requirements. The library provides different types of 2-D FFT codes and has light dependencies. Therefore, we modified the codes to make them applicable to the Arm CPU cores, especially the parts related to dynamic memory allocation. The main limit of this library is that it requires a power of 2 for the FFT size. Therefore, we used the cropped Consultative Committee for Space Data Systems (CCSDS) image data as input.

We have enhanced the benchmark with additional functionalities. For the software implementation, supplementary functions take the provided input data and validate the execution by comparing the executed output results with the provided reference output data. After comparing each pixel, the benchmark shows the maximum and average value of relative error **(Requirement 7)**. Moreover, the execution time of the Arm CPU is measured via a hardware timer "XilTimer.h" provided by AMD-Xilinx. The measurement of the external memory access is based on the performance counter of the Arm CPU. For the hardware implementation, the synthesis reports from the HLS tool are used to estimate the execution time and memory accesses.

4. THEORETICAL ROOFLINE MODEL FOR HW/SW DESIGNS

The common theoretical roofline model for CPU and FPGA can be obtained by considering the hardware characteristics. We selected the Zynq UltraScale+ evaluation board ZCU102 as a case study. Fig 4 shows the theoretical roofline considering CPU cores and FPGA. The next paragraphs explain the model in detail.

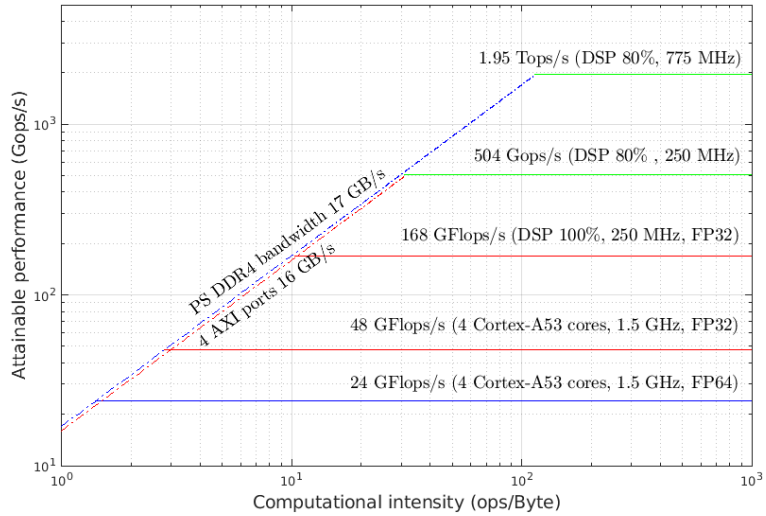

Figure 4. Theoretical roofline of ZCU102 based on Arm Cortex-53 and embedded FPGA using DSP blocks

Zynq UltraScale+ evaluation board

The Zynq UltraScale+ evaluation board ZCU102 includes a quad-core Arm Cortex-A53, a dual-core Cortex-R5 in the Processing System (PS), and an embedded FPGA called Programmable Logic (PL) in AMD-Xilinx terminology. The PL includes 2,520 DSP blocks, 32.1 Mb Block RAM (BRAM), 548K Flip-Flops (FF), and 600K System Logic Cells. For the external memory, we consider the DDR4 memory attached to Arm CPU architectures, called PS DDR. The Advanced eXtensible Interface (AXI) is considered as a connection between the PS and PL.

Theoretical CPU roofline model

The **CPU computational ceiling** is calculated using the number of cores, Floating-point Unit (FPU), and clock frequency. In the case of Armv-8, so-called AArch64, the default compilation target architecture (-march='armv8-a') already includes two feature modifiers: enable floating-point instructions (+fp) and enable Advanced SIMD instructions (+SIMD). The Arm NEON SIMD engine and the Arm Vector Floating-Point (VFP) architecture, VFPv4 for Arm Cortex-A53, are automatically used, unless users purposely remove these feature modifiers. Thus, the theoretical computational ceiling of Armv-8 CPU architecture is:

$$\begin{aligned} C_{A53} = \text{number of cores} \times \frac{\text{vector register bit-width}}{\text{floating-point data bit-width}} \\ \times \text{Max. FLOP per instruction} \times \text{clock frequency} \end{aligned} \tag{3}$$

The Zynq UltraScale+ quad-core Cortex-A53 operates up to 1.5 GHz. As the AArch64 NEON has 128-bit vector registers, four FP32 instructions or two FP64 instructions can be processed in parallel. The AArch64 NEON supports Fused Multiply-Add (FMA), corresponding to two Floating-point operations (FLOP) [36]. Thus, the theoretical FP32 computational ceiling of the quad-core Arm Cortex-A53 on the Zynq UltraScale+ platform is:

$$\begin{aligned} C_{A53,FP32} &= 4 \text{ cores} \times \frac{128 \text{ bits/core}}{32 \text{ bits}} \times 2 \text{ FLOP} \times 1.5 \text{ GHz} \\ &= 48 \text{ GFLOP/sec} \end{aligned} \tag{4}$$

Similarly, the theoretical computational ceiling for FP64 is:

$$\begin{aligned} C_{A53,FP64} &= 4 \text{ cores} \times \frac{128 \text{ bits/core}}{64 \text{ bits}} \times 2 \text{ FLOP} \times 1.5 \text{ GHz} \\ &= 24 \text{ GFLOP/sec} \end{aligned} \tag{5}$$

The **CPU I/O bandwidth ceiling** is calculated using the memory bandwidth. Assuming that all the input data is initially stored in the external DDR memory, the theoretical I/O bandwidth of the DDR memory is:

$$BW = \text{DDR transfer rate} \times \text{DDR transfer size} \tag{6}$$

$$\begin{aligned} BW_{PS_DDR4} &= 2133 \text{ MT/sec} \times 64 \text{ bits/transfer} \\ &= 17.064 \text{ GB/sec} \end{aligned} \tag{7}$$

Theoretical FPGA roofline model

The **FPGA computational ceiling** is calculated using the hardware resource utilization. We focus on DSP blocks among hardware resources, considering the computation-intensive characteristics of the 2-D FFT mentioned in section 5 as a use-case [37]. The theoretical computational ceiling of the PL of the Zynq UltraScale+ platform is:

$$C_{PL} = \frac{\text{available DSP blocks} \times \text{clock frequency}}{\text{required DSP blocks per operation}} \tag{8}$$

The Vitis HLS tool provides synthesis reports that highlight that the recommended maximum DSP block utilization is 80% of the existing DSP blocks on a given platform, mainly due to routing constraints. We follow this guideline to set the available DSP blocks. The used clock frequency is based on the RTL reports. When the data bit-width fits in the DSP block input size, one DSP block is required for an arithmetic operation, i.e.:

$$\begin{aligned} C_{PL,FXP} &= 2016 \text{ DSP blocks} \times 250 \text{ MHz} \times \frac{1 \text{ op}}{1 \text{ DSP block}} \\ &= 504 \text{ Gop/sec} \end{aligned} \tag{9}$$

For the floating-point operations, the required DSP blocks per operation are given by the AMD-Xilinx document resource utilization for floating-point operations [38]. Therefore, we assume the maximum number of required DSPs per operation depending on the data type. For example, if all the operations consist of multiplications and three DSP blocks are required for one FP32 multiplication, the computational ceiling of FP32 for the FPGA is:

$$\begin{aligned} C_{PL,FP32} &= 2016 \text{ DSP blocks} \times 250 \text{ MHz} \times \frac{1 \text{ op}}{3 \text{ DSP blocks}} \\ &= 168 \text{ Gop/sec} \end{aligned} \tag{10}$$

5. USE-CASE: 2-D FFT HW/SW DESIGN WITH THE COMMON ROOFLINE MODEL

CPU design

The 2-D FFT design was executed on the single-core Cortex-A53 in bare-metal, using the proposed benchmark. During the execution, only one Cortex-A53 was active and other cores were deactivated.

The number of operations is obtained using the characteristics of the 2-D FFT algorithm. As the benchmark is based on Ooura library, the split-radix 2-D FFT is used [35]. The library provides two types of 2-D FFT. We chose the faster one, called fftsg2d, which uses the 1-D FFT routines called fftsg. The total number of floating-point operations (FLOP) is the sum of real multiplications and additions. The FLOP for the complex-data 2-D FFT of an $M \times N$ array is given by equation 11, based on the standard Yavne split-radix algorithm, where M and N are a power of 2 [39, 40].

$$\begin{aligned} FLOP = &(4M * log_2 M - 6M + 8) * N \\ &+ (4N * log_2 N - 6N + 8) * M \end{aligned} \tag{11}$$

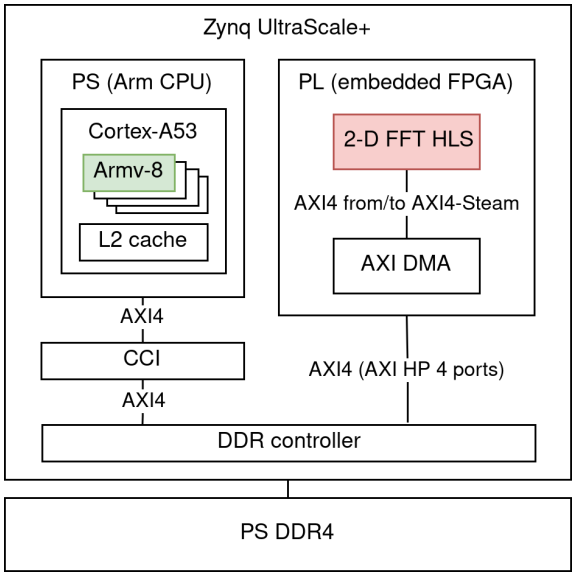

Figure 5. ZCU102 configuration. The CPU and FPGA design of the 2-D FFT benchmark are marked in color.

For the calculation of the CI, the external memory traffic is the sum of Low Level Cache (LLC) misses and write-back traffic [41]. The external memory access (in bytes) is measured using the slave register of the Cache Coherence Interconnect (CCI) directly connected to the Cortex-A53 cores. The slave register is the only data bus connected to the output of the L2 cache of the Cortex-A53 cores [42]. When an L2 cache miss happens on the Cortex-A53, it appears at the CCI, which is connected to all memories, including the DDR memory [43]. We use the CCI performance counter to obtain the number of L2 cache misses by managing the Performance Monitoring Unit (PMU) of the CCI [44]. More specifically, the CCI provides four event counters that can count the number of read and write request handshakes. Also, the CCI uses the L2 cache line as granularity [45].

$$CI_{A53} = \frac{\text{number of operations}}{\text{CCI read\&write requests} \times \text{L2 cache line size}} \tag{12}$$

The execution time is measured using the hardware timer functionality provided by AMD-Xilinx. The timer library "XilTimer.h" provides function and predefined parameters to get the number of timer ticks and the relevant clock frequency.

$$P_{A53} = \frac{\text{number of operations}}{\text{number of timer ticks/clock frequency}} \tag{13}$$

FPGA design based on HLS

The 2-D FFT design is simulated using the Vitis HLS tool to estimate the characteristics of the hardware design. By using the HLS reports, we can calculate the CI and performance of hardware designs. We selected AMD-Xilinx open-source HLS DSP library, named Vitis Libraries, for the FPGA design [46]. The 2-D FFT under study is configured as radix-4 FFT and Super Sample Rate (SSR) FFT scale mode. The library provides several scaling modes. For the fixed-point cases, we chose the scale mode that keeps the output data bit-width the same as the input data bit-width, which fits in the DSP block input data bit-width. Thus, every fixed-point arithmetic operation requires only one DSP block. The total number of operations OP, considering the complex-data 2-D FFT of an $M \times N$ array, is as below [47]. For the FPGA design, the term FLOP is not used in the same way as the CPU design since we use both floating-point and fixed-point operations in our designs.

$$OP = (\frac{17}{2} M log_4 M) * N + (\frac{17}{2} N log_4 N) * M \tag{14}$$

For the calculation of CI for the hardware design, the amount of memory access in bytes is obtained as the multiplication of transferred data per cycle and trip count, which can be found in HLS C synthesis reports [48]. The HLS design uses AXI4-Stream for the interface between PS and PL, which means that AXI Direct Memory Access (DMA) is required for the complete design as shown in Fig. 5. The transferred data through the AXI4-Stream per cycle is called TDATA and is represented in bits, indicated in the hardware interface section in the C synthesis report. The trip count is the number of cycles to execute the total loop iterations. If the HLS design has several loop hierarchies in the C synthesis report, the multiplication of all the trip counts in each hierarchy gives the total trip count. In our design, only input data and output data are interfaced via AXI4-Stream, whereas there is no intermediate data transfer during the 2-D FFT, so all the data processing can be performed relying on BRAMs, hence reducing external memory accesses.

$$CI_{PL} = \frac{\text{number of operations}}{in(\text{TDATA} \times \text{trip count}) + out(\text{TDATA} \times \text{trip count})} \tag{15}$$

For the performance calculation, the estimated execution time of the hardware design is the multiplication of the total latency in cycles and the achieved clock frequency. The clock frequency is obtained via the C synthesis report or RTL synthesis report. We selected the clock frequency of the RTL synthesis report, including placement and routing, as it provides a more realistic estimation. The C/RTL co-simulation reports provide the total latency of the simulated RTL. The C/RTL co-simulation uses a C/C+++ testbench whose main purpose is to verify the behavioral characteristics of the RTL design.

$$P_{PL,AXIS} = \frac{\text{number of operations}}{\text{latency in cycles} \times \text{clock frequency}} \tag{16}$$

Application-specific CPU/FPGA roofline model

The common roofline model targeting HW/SW design is drawn by combining the theoretical CPU roofline model and the application-specific FPGA roofline model. As introduced before, the CPU architecture remains the same regardless of the application. However, an FPGA is a reconfigurable architecture, and the resource utilization of FPGA changes depending on the hardware design. It means that the FPGA roofline is hardware-specific and application-specific. The proposed roofline model for CPU and FPGA and design points for our case study are shown in Fig. 6

We obtained the HW/SW designs based on the proposed benchmark, using different input image data types, FFT sizes, and data bit-widths. As real-time requirement, we set 80 ms as the targeting execution time based on the current SVOM ECLAIRs implementation on the LEON processor. For the input image data, we selected Pleiades b3 and SAR data from the CCSDS

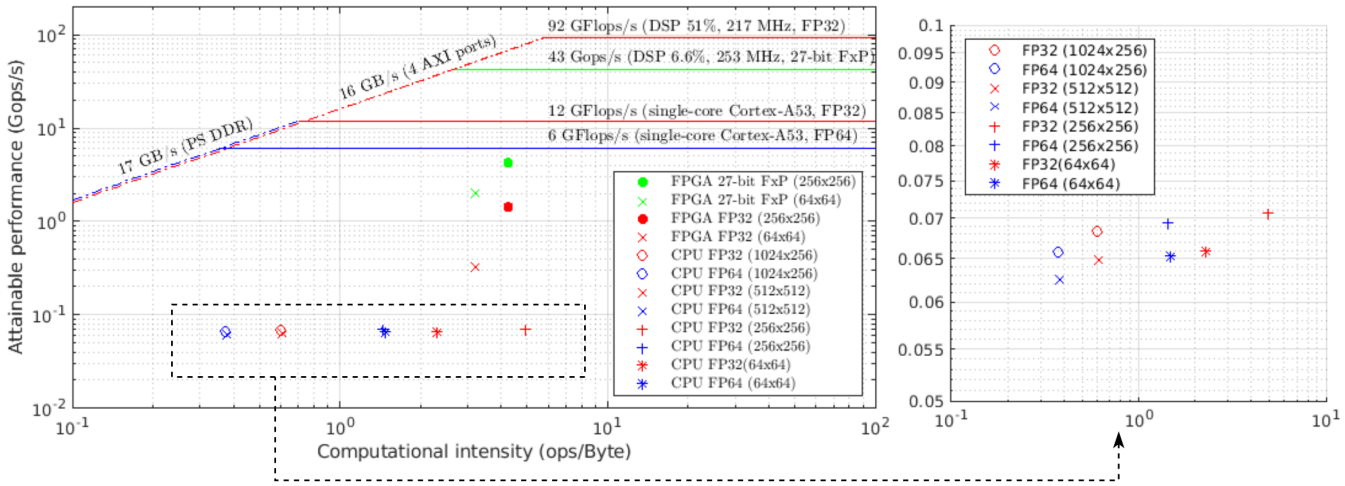

Figure 6. CPU/FPGA roofline model with application and kernel design

BB122 TestImage data set. In the case of the CPU design, there is no specific limit on FFT size, so we use the SAR image (512×512) and the Pleiads b3 image (1376×320). For the Pleiades b3 image, as the number of rows and columns is not a power of 2, as required by the Ooura library (1376×320), we used cropped images: 1024×256, 256×256, and 64×64. We used FP32 and FP64 data types for CPU design because both floating-point types are frequently used in ground data processing, and the target CPU uses FPU and SIMD by default.

For the FPGA design, the AMD-Xilinx DSP library currently has a limitation on the maximum FFT size (256×256). Hence, we selected an FFT size equal to 256×256 and 64×64 using the cropped image of Pleiads b3. The 27-bit fixed-point and FP32 formats are selected for the FPGA design, considering that the target platform's DSP blocks (DSP48E2) do not directly support floating-point arithmetic operations. For example, with DSP48E2, one FP32 multiplication requires 3 DSP blocks, and one FP32 addition requires 2 DSP blocks [38]. If an application does not require floating-point arithmetic or if the output results with the fixed-point format lead to an acceptable error, the resource utilization can be decreased by adopting fixed-point arithmetic.

The CPU roofline model for the use-case (Fig. 4) is composed of the computational ceilings of the FP32 and FP64 cases, and the I/O bandwidth ceiling based on PS DDR (17 GB/s). Then, the FPGA roofline model includes the I/O bandwidth ceiling, considering the AXI (16 GB/s), and the computational ceilings of FP32 and 27-bit fixed-point format cases, considering the generated HLS reports of the 2-D FFT kernels with an FFT size of 256×256. The used parameters, such as latency, clock frequency, and the DSP block utilization, are shown in Table 2.

To compare the results, each HW/SW design point is indicated in the same plane on the roofline model. All FPGA designs have higher performance compared to the CPU designs. Therefore, the generated hardware designs are suitable as accelerators for an MPSoC on space missions. For example, for the FFT size of 256×256, the execution time on FPGA is 1.4 ms for the fixed-point design and 3 ms for the FP32-based design, leading to 25 and 55 times faster execution than the real-time requirement of the SVOM ECLAIRs, respectively. Furthermore, the above hardware accelerators are 15 and 35 times faster than the generated CPU designs, respectively. More precisely, the fixed-point design has higher performance, even if the DSP blocks utilization is significantly lower than the floating-point design. Therefore, using a fixed-point design in terms of performance and area efficiency is more efficient. However, as the data accuracy requirement is different depending on the mission and payload, future payload teams shall decide the specification of the required data bit-width considering the data accuracy requirement.

For the CPU designs, even if the performance is lower than the FPGA designs, they still have advantages in terms of data bit-width, FFT size, and simplicity of implementation. The generated 2-D FFT software designs cover both FP32 and FP64, and it is thus suitable for payloads where high data precision is required. Users can implement bigger FFT sizes, as the library is available for different sizes for input images, as long as the input data size conforms to a power of 2. Moreover, the software implementation is easier and faster than the hardware implementation. Even if the HLS tools ease the complexity of the hardware design process, users still need to have a background in FPGA architectures and HLS-based design flow.

We further analyze the software design results of Table 3 considering the CPU core, the external DDR memory, and the on-chip cache memory. For the 3 CPU design cases with the FFT size of 64×64 (FP32 and FP64) and the FFT size of 256×256 for FP32, we observe that there is no writing back from the Cortex-A53 core. This means that the L2 cache is large enough to contain all the input data initially stored in the DDR memory. Therefore, only L1 and L2 caches are used during the 2-D FFT processing without using the external DDR memory to store intermediate data. Therefore, the performance and computational intensity of the 2-D FFT implementation are mainly affected by the available on-chip memory and the number of required operations.

Even if we obtain highly pipelined hardware designs and optimized CPU designs using caches, the design points are far from

Table 2. HLS synthesis results of 2-D FFT, including placement and routing, based on the no scaling mode of the AMD-Xilinx DSP 2-D FFT library

FFT size	64×64		256×256		Report type
Data type	27-bit FxP	FP32	27-bit FxP	FP32	
TDATA	512 bits				C synthesis
Trip count	512		8192		
Latency (cycles)	23009	44807	346355	676457	Cosimulation
DSP blocks	112 (4.4%)	928 (37%)	168 (6.7)	1276 (51%)	RTL synthesis
BRAM	32 (1.8%)	32 (1.8%)	420 (23%)	528 (29%)	
LUT	39368 (14%)	109870 (40%)	57676 (21%)	157710 (58%)	
FF	61168 (11%)	199467 (36%)	75035 (15%)	262969 (51%)	
Frequency (MHz)	226	227	253	217	

Table 3. 2-D FFT execution results on single-core Arm Cortex-A53

FFT size	Data type	CCI read request	CCI write request	Execution time (ms)
64×64	FP32	1009	0	22.5
	FP64	1564	0	22.7
256×256	FP32	10849	0	48.2
	FP64	28780	8270	49.1
512×512	FP32	262683	143524	242
	FP64	404774	247791	251
1024×256	FP32	265506	145094	230
	FP64	407158	249861	239

the theoretical ceilings. We believe that this behavior is due to non-operational behaviors which generate stalls, which can indicate that there is still room for optimization. More precisely, the matrix transpose between the row-wise 1-D FFT and the column-wise 1-D FFT requires several memory accesses [49]. In future work, we will consider the 2-D FFT implementation based on a non-transpose algorithm to improve performance efficiency.

6. Conclusion

In this work, we identify and analyze current and near-future on-board processing platforms that can be used for on-board payload data processing. Furthermore, we present a payload algorithm survey based on information obtained by existing scientific mission teams. The survey showed that the 2-D FFT is the most widely used algorithm regardless of payload types. Then, we propose a DSE methodology combined with a roofline model for HW/SW designs. We designed a benchmark for the on-board payload data processing dedicated to HW/SW designs. The benchmark includes space-oriented input and reference data and source codes for the CPU design and the HLS-based FPGA design. The code provides time measurement and result validation functionalities to analyze the generated design with regard to performance and data accuracy. We implemented the software on the single-core Arm Cortex-A53 and HLS synthesis for hardware implementation on the embedded FPGA. Based on the generated software and hardware designs, we obtained design points of the CPU/FPGA roofline model. The results show that the designs conform with the real-time execution time requirements.

Acknowledgments

This research was supported by CNES and University of Rennes thanks to PhD funding granted to Seungah Lee.

References

[1] NASA, "State-of-the-Art Small Spacecraft Technology," Tech. Rep. NASA/TP—2022–0018058, 2023.

[2] V. Leon *et al.*, "FPGA & VPU Co-Processing in Space Applications: Development and Testing with DSP/AI Benchmarks," in *2021 28th IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, Nov. 2021, pp. 1–5.

[3] N. Perryman, C. Wilson, and A. George, "Evaluation of Xilinx Versal Architecture for Next-Gen Edge Computing in Space," in *2023 IEEE Aerospace Conference*, Mar. 2023, pp. 1–11, iSSN: 1095-323X.

[4] N. Ibellaatti *et al.*, "HERMES: qualification of High pErformance pRogrammable Microprocessor and dEvelopment of Software ecosystem," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Apr. 2023, pp. 1–5, iSSN: 1558-1101.

[5] H. Xiao *et al.*, "The data center for the Spectrometer and Telescope for Imaging X-rays (STIX) on board Solar Orbiter," *Astronomy & Astrophysics*, vol. 673, p. A142, May 2023. [Online]. Available: https://www.aanda.org/10.1051/0004-6361/202346031

[6] W. Benz *et al.*, "The CHEOPS mission," *Experimental Astronomy*, vol. 51, no. 1, pp. 109–151, Feb. 2021. [Online]. Available: https://doi.org/10.1007/s10686-020-09679-4

[7] H. Cruz, M. Véstias, J. Monteiro, H. Neto, and R. P. Duarte, "A Review of Synthetic-Aperture Radar Image Formation Algorithms and Implementations: A Computational Perspective," *Remote Sensing*, vol. 14, no. 5, p. 1258, Mar. 2022. [Online]. Available: https://www.mdpi.com/2072-4292/14/5/1258

[8] M. Maksimovic *et al.*, "The Solar Orbiter Radio and Plasma Waves (RPW) instrument," *Astronomy & Astrophysics*, vol. 642, p. A12, Oct. 2020. [Online]. Available: https://www.aanda.org/10.1051/0004-6361/201936214

[9] B. H. Dean, "(54) DISCRETE FOURIER TRANSFORMINA COMPLEX VECTOR SPACE," Patent.

[10] S. Wiehle, S. Mandapati, D. Günzel, H. Breit, and U. Balss, "Synthetic Aperture Radar Image Formation and Processing on an MPSoC," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1–14, 2022, conference Name: IEEE Transactions on Geoscience and Remote Sensing.

[11] J. S. Smith, B. H. Dean, and S. Haghani, "Distributed computing architecture for image-based wavefront sensing and 2D FFTs," in *Advanced Software and Control for Astronomy*, H. Lewis and A. Bridger, Eds., vol. 6274. SPIE, 2006, p. 627421, backup Publisher: International Society for Optics and Photonics. [Online]. Available: https://doi.org/10.1117/12.672842

[12] M. Fingeroff, *High-level Synthesis: Blue Book*. Xlibris Corporation, 2010.

[13] R. Nane *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, Oct. 2016, conference Name: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

[14] Simens, "Precision | Advanced FPGA Synthesis & Validation | Siemens Software." [Online]. Available: https://eda.sw.siemens.com/en-US/ic/precision/

[15] AMD, "Vitis Software Platform." [Online]. Available: https://www.xilinx.com/products/design-tools/vitis/vitis-platform.html

[16] P. MERCIER, "Design on NG-ULTRA FPGAs using a HLS-to-Bitstream design flow (TAS)," ESTEC, Mar. 2023. [Online]. Available: https://indico.esa.int/event/439/contributions/7967/

[17] AMD, "Vitis High-Level Synthesis User Guide," Tech. Rep., 2021.

[18] 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, Apr. 2009. [Online]. Available: https://dl.acm.org/doi/10.1145/1498765.1498785

[19] N. Ding and S. Williams, "An Instruction Roofline Model for GPUs," in *2019 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. Denver, CO, USA: IEEE, Nov. 2019, pp. 7–18. [Online]. Available: https://ieeexplore.ieee.org/document/9059264/

[20] B. da Silva Gomes, A. Braeken, E. D'Hollander, and A. Touhafi, "Performance modeling for FPGAs: extending the roofline model with high-level synthesis tools," *INTERNATIONAL JOURNAL OF RECONFIGURABLE COMPUTING*, vol. 2013, pp. 1–10, 2013, publisher: Hindawi Publishing Corporation. [Online]. Available: http://hdl.handle.net/1854/LU-4226966

[21] M. Siracusa *et al.*, "A Comprehensive Methodology to Optimize FPGA Designs via the Roofline Model," *IEEE Transactions on Computers*, vol. 71, no. 8, pp. 1903–1915, Aug. 2022, conference Name: IEEE Transactions on Computers.

[22] E. Calore and S. F. Schifano, "FER: A Benchmark for the Roofline Analysis of FPGA Based HPC Accelerators," *IEEE Access*, vol. 10, pp. 94 220–94 234, 2022, conference Name: IEEE Access.

[23] D. Plus, "Radiation Tolerant SDRAM Stacks | Volatile Memory | Space market." [Online]. Available: https://www.3d-plus.com/products/sdram/

[24] R. Merl *et al.*, "LEON4 Based Radiation-Hardened SpaceVPX System Controller," in *2020 IEEE Aerospace Conference*, Mar. 2020, pp. 1–10, iSSN: 1095-323X.

[25] A. HajiRassouliha, A. J. Taberner, M. P. Nash, and P. M. Nielsen, "Suitability of recent hardware accelerators (DSPs, FPGAs, and GPUs) for computer vision and image processing algorithms," *Signal Processing: Image Communication*, vol. 68, pp. 101–119, Oct. 2018.

[26] K. Moore, J. Sherrill, and J. Mayes, "RTEMS State of the Solar System," 2022.

[27] T. Beck, F. Boniol, J. Ermont, and L. Maillet, "Impact of environment on the execution of a real-time Linux process on a multicore platform," in *11th European Congress on Embedded Real-Time Systems (ERTS 2022)*, Toulouse, France, Jun. 2022. [Online]. Available: https://hal.science/hal-03857305

[28] O. Notebaert, "On-Board Payload Data Processing requirements and technology trends," ESTEC, Feb. 2019. [Online]. Available: https://indico.esa.int/event/225/timetable/

[29] ECSS, "ECSS-Q-HB-80-01A Reuse of existing software," Dec. 2011. [Online]. Available: https://ecss.nl/hbstms/ecss-q-hb-80-01a-reuse-of-existing-software/

[30] ESA, "OBPMark (on-board processing benchmarks)." [Online]. Available: https://obpmark.github.io/

[31] L. Kosmidis *et al.*, "GPU4S: Major Project Outcomes, Lessons Learnt and Way Forward," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Feb. 2021, pp. 1314–1319, iSSN: 1558-1101.

[32] ESA, "NIR HAWAII-2RG Data processing algorithms - Benchmarking software." [Online]. Available: https://essr.esa.int/project/nir-hawaii-2rg-data-processing-algorithms-benchmarking-software

[33] K. Henderson, N. Wiatrek, and P. Saenz, "ARMing the Next Generation of Spaceflight Embedded Platforms Through Processor Reusability," in *2023 IEEE Aerospace Conference*, Mar. 2023, pp. 1–10, iSSN: 1095-323X. [Online]. Available: https://ieeexplore.ieee.org/document/10115627

[34] CCSDS, "SLS-DC BB122TestImage," Apr. 2007. [Online]. Available: https://cwe.ccsds.org/sls/docs/Forms/AllItems.aspx

[35] T. OOURA, "FFT Package 1-dim / 2-dim." [Online]. Available: https://www.kurims.kyoto-u.ac.jp/ ooura/fft.html

[36] Arm, "ARM Cortex-A53 MPCore Processor Advanced SIMD and Floating-point Extension Technical Reference Manual r0p4," Jan. 2016.

[37] D. Diakite, N. Gac, and M. Martelli, "OpenCL FPGA Optimization guided by memory accesses and roofline model analysis applied to tomography acceleration," in *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*, Aug. 2021, pp. 109–114, iSSN: 1946-1488.

[38] Xilinx, "Performance and Resource Utilization for Floating-point v7.1," Tech. Rep., 2022.

[39] S. G. Johnson and M. Frigo, "A Modified Split-Radix FFT With Fewer Arithmetic Operations," *IEEE Transactions on Signal Processing*, vol. 55, no. 1, pp. 111–119, Jan. 2007. [Online]. Available: http://ieeexplore.ieee.org/document/4034175/

[40] R. Yavne, "An economical method for calculating the discrete Fourier transform," in *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, ser. AFIPS '68 (Fall, part I). New York, NY, USA: Association for Computing Machinery, Dec. 1968, pp. 115–125. [Online]. Available: https://dl.acm.org/doi/10.1145/1476589.1476610

[41] 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)*, Mar. 2014, pp. 76–85. [Online]. Available: https://ieeexplore.ieee.org/document/6844463

[42] "Zynq UltraScale+ MPSoC." [Online]. Available: https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html

[43] A. Tyagi and H. Duwe, "Decoupled Real-Time Program Execution State Monitoring," Tech. Rep. FA8750-20-1-0504, Sep. 2021. [Online]. Available: https://apps.dtic.mil/sti/trecms/pdf/AD1148458.pdf

[44] Arm, "ARM CoreLink CCI-400 Cache Coherent Interconnect Technical Reference Manual r1p5," Tech. Rep. ARM DDI 0470K (ID011116).

[45] R. Pujol, H. Tabani, J. Abella, M. Hassan, and F. J. Cazorla, "Empirical Evidence for MPSoCs in Critical Systems: The Case of NXP's T2080 Cache Coherence," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Feb. 2021, pp. 1162–1165, iSSN: 1558-1101. [Online]. Available: https://ieeexplore.ieee.org/document/9474078

[46] Xilinx, "Vitis Libraries," 2023. [Online]. Available: https://docs.xilinx.com/r/en-US/Vitis_Libraries/index.html

[47] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965. [Online]. Available: https://www.ams.org/mcom/1965-19-090/S0025-5718-1965-0178586-1/

[48] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. Monterey California USA: ACM, Feb. 2015, pp. 161–170. [Online]. Available: https://dl.acm.org/doi/10.1145/2684746.2689060

[49] C.-L. Yu, J.-S. Kim, L. Deng, S. Kestur, V. Narayanan, and C. Chakrabarti, "FPGA Architecture for 2D Discrete Fourier Transform Based on 2D Decomposition for Large-sized Data," *Journal of Signal Processing Systems*, vol. 64, no. 1, pp. 109–122, Jul. 2011. [Online]. Available: http://link.springer.com/10.1007/s11265-010-0500-y

BIOGRAPHY

Seungah Lee *is a PhD student at University of Rennes, France, with the collaboration of CNES (French Space Agency) and IRISA Lab./Inria (French Institute for Research in Com- puter Science and Automation). She received her B.S. and M.S degrees in Space Science from Kyung-Hee University, South Korea. She also received her M.S in Aerospace Engineering specialized in embedded systems from ISAE-SUPAERO, France. Her research includes on-board data processing and payload development for space missions.*

Emmanuel Casseau *is a Professor at ENSSAT Graduate Engineering School, University of Rennes, Lannion, France. He is a member of the IRISA/Inria Lab., France. His research interests are in the broader area of computer architecture, where he investigates the design of high-performance and cost-effective embedded systems and reconfigurable-based architecture design. Within this context, he has performed research in high-level synthesis, mapping/scheduling techniques, custom and application-specific hardware architectures targeting multimedia and signal processing applications, reconfigurable architectures and FPGA-based accelerators.*

Olivier Sentieys *is a Professor at the University of Rennes holding the Inria Research Chair on Energy-Efficient Computing Systems. He is leading the Taran team common to Inria and IRISA Laboratory. His research interests are in the area of computer architectures, computer arithmetic, and embedded systems, with a focus on system-level design, energy-efficient hardware accelerators, approximate computing, fault tolerance, and energy harvesting sensor networks.*

Angeliki Kritikakou *is an Associate Professor at Univ. Rennes and IRISA/Inria Rennes research center and holds a position at IUF. She received her Ph.D. in 2013 from Univ. Patras, Greece, in collaboration with IMEC Center, Belgium, and her HDR from Univ. Rennes, France, in 2022. Her research interests include embedded systems, real-time systems, HW/SW co-design, mapping methodologies, design space exploration methodologies, memory management methodologies, low power design and fault tolerance.*

Rubén Salvador *(Member, IEEE) is an Associate Professor at CentraleSupélec and IRISA laboratory, Inria Centre at Rennes University, France. He obtained his PhD (2015) from Univ. Politécnica de Madrid. He was a Visiting Professor (2017) with the IETR/INSA Rennes, France, and a visiting research student (2009) with the Dpt. of Computer Systems, Brno University of Technology, Czech Republic. His research interests span the areas of hardware security, computer architecture, and reconfigurable computing, with a current focus on FPGA and embedded systems security, and hardware and architectural support for security in heterogeneous architectures, hardware accelerators, and emerging computing paradigms.*

Julien Galizzi *is a on-board software engineer expert in CNES (French Space Agency). He is mainly in charge of R&D studies and their deployment on operational space missions in the area of embedded software in order to ease the software development on scientific payloads.*