Noname manuscript No.
(will be inserted by the editor)

Evaluating the computational performance of the Xilinx Ultrascale+ EG Heterogeneous MPSoC

Jose A. Belloch · Germán León · José M. Badía · Almudena Lindoso · Enrique San Millan

Received: date / Accepted: date

Abstract The emergent technology of Multi-Processor System-on-Chip (MPSoC), which combines heterogeneous computing with the high performance of Field Programmable Gate Arrays (FPGAs) is a very interesting platform for a huge number of applications ranging from medical imaging and augmented reality to high-performance computing in space. In this paper, we focus on the Xilinx Zynq UltraScale+ EG Heterogeneous MPSoC, which is composed of four different processing elements (PE): a dual-core Cortex-R5, a quad-core ARM Cortex-A53, a graphics processing unit (GPU) and a high end FPGA. Proper use of the heterogeneity and the different levels of parallelism of this platform becomes a challenging task. This paper evaluates this platform and each of its PEs to carry out fundamental operations in terms of computational performance. To this end, we evaluate image-based applications and a matrix multiplication kernel. On former, the image-based applications leverage the heterogeneity of the MPSoc and strategically distributes its tasks among both kinds of CPU cores and the FPGA. On the latter, we analyze separately each PE using different matrix multiplication benchmarks in order to assess and compare their performance in terms of MFlops. This kind of operations are being carried out for example in a large number of space-related applications where the MPSoCs are currently gaining momentum. Results stand out the fact that different PEs can collaborate efficiently with the aim of accelerating the computational-demanding tasks of an application. Another important aspect to highlight is that leveraging the parallel OpenBLAS library we achieve up to 12 GFlops with the four Cortex-A53 cores of the platform, which is a considerable performance for this kind of devices.

Jose A. Belloch, Almudena Lindoso and Enrique San Millan
Depto. de Tecnología Electrónica, Universidad Carlos III de Madrid, Spain
E-mail: {jbelloc,alindoso,quique}@ing.uc3m.es

José M. Badía, German Leon
Depto. de Ingeniería y Ciencia de Computadores, Universitat Jaume I de Castellon, Spain.
E-mail: {leon,badia}@uji.es

Keywords heterogeneous computing, parallel computing, multi-processor system-on-chip (MPSoC), Xilinx Ultrascale+.

1 Introduction

Multi-Processor System-on-Chip (MPSoC) devices are gaining momentum inside the category of very-large-scale integrated (VLSI) systems [1]. Modern MPSoCs, such as the Xilinx Ultrascale+ EG device, whose performance is analyzed in this work, combine heterogeneous computing with the high performance of Field Programmable Gate Arrays (FPGAs) [2]. MPSoCs have been traditionally used in networking, automotive, communications, signal processing, and multimedia among other applications [3]. They are currently attracting the attention of the space agencies in order to carry out the compute demanding tasks on board, and efforts are being carried out to improve their radiation tolerance [4–6]. As a matter of fact, NASA is testing commercial Xilinx Zynq SoC devices in the International Space Station (ISS) [7], since hardware of this kind could face the increasing demand of performance per watt in processors used in space [8].

The Xilinx Zynq Ultrascale+ MPSoC platform offers high levels of heterogeneity and paralellism since it is composed by four different Processing Elements (PEs): a dual-core Cortex-R5, a quad-core ARM Cortex-A53, a low end Graphics Processing Unit (GPU) and a high end FPGA. Therefore, this MPSoC offers multiple parallelism levels, although leveraging properly its computational resources becomes a challenging task. In fact, management issues become even more difficult when different programming environments are involved.

Different performance studies have already been carried out in order to assess if different types of processing elements such as the ones included in the Ultrascale+ MPSoC can fit with the increasing needs in the space field. One of the difficulties of the researchers in this field is the access to specific space benchmarks. Researchers can use ESA benchmarks: NIR Hawaii-2RG BM [9] and Next Generation DSP [10]. ESA is already using Next Generation DSP benchmark for performance assessment of processors and computers. Some representative examples of performance studies are briefly mentioned next. For example, in [11] Lentaris et al. evaluate 30 devices for vision-based navigation, including CPUs, GPUs, DSPs and FPGAs in terms of speed, performance per watt and radiation tolerance. In [8] Kosmidis et al. study the applicability of embedded GPUs in space, as they have become an attractive option in terms of performance per watt. Performance comparison between conventional CPUs to embedded ARM processors integrated in SoC devices such as OMAP 4460 are performed in [12], where a relation that involve CPU frequencies, performance and types of CPUs is shown. The study in [13] shows that the throughput of embedded CPUs could be increased tenfold when clock rates of 2.3 GHz are used. From [14], it is derived that the Cortex-A15 processor at 1.2 GHz is 3-5x faster than the Cortex-A9 of Zynq at 667 Mhz. Comparison of GPU

and FPGAs in terms of throughput was carried out by Tippets et al in, [15] where the authors compare the performance of several stereo vision algorithms. GPUs achieve 0.1 -7 Gde/s, whereas FPGA achieve 1-6 Gde/s, where Gde/s is defined as the billions of disparity evaluations per second when comparing the disparities for all pixels in four image data sets. Vision algorithms for space are evaluated in desktops CPU and GPU in [16], where the GPUs achieve one order of magnitude higher performance per watt than the desktop CPUs.

The works presented in [17,18] indicate that heterogeneous platforms in the form of SoC composed by an ARM processor and FPGA are promising candidates for space computing, where the FPGA can be used as a hardware accelerator for intensive computational requirements [19]. All works presented above deal with one or two kinds of processing elements per platform, but none of them explores the possibilities offered by an MPSoC, as an heterogeneous device that combines four kinds of processing elements. Although a deep analysis regarding space issues should be done in the future, we want to present in this work an initial exploration of the performance possibilities of each one of those processing elements, as well as an analysis regarding the flexibility offered by the heterogeneity and parallelism of the UltraScale+ MPSoC. To this end, we have selected two kinds of computations that are present in a large number of applications and that have two clearly different targets:

- A computation intensive application that can be executed on the four PEs of the MPSoC so that, the different performances of the PEs can be assessed and compared. Specifically, we use matrix multiplication, a basic lineal algebra core that arises in multiple applications.
- A complex application, [20], that allows us to evaluate the heterogeneity of the platform, i.e, how the PEs interact with each other in order to carry out tasks collaboratively. To this end we use an application that performs two well-know image processing tasks by combining different PEs of the platform.

The rest of the paper is structured as follows: the following section overviews important concerns to consider when MPSoC is used in the space environment. In Section III, we briefly describe the computational resources of the MPSoC Xilinx Zynq Ultrascale+ together with the software to develop and run applications on this kind of platforms. Next, in Section IV, we introduce the different applications leveraging the PEs of the MPSoC and evaluate their computational performance. Finally, Section V provides some concluding remarks.

2 Radiation tolerance in space environments

Throughput, performance per watt and memory bandwidth are becoming increasingly important for computational platforms in space. However, on this environment we require also to have into account the harmful radiation effects that can affect the spacecraft's electronics [21].

Due to potential radiation-based malfunctioning of electronics in space, commercial circuits require additional testing before they can be used for space applications. COTS (Commercial Off-The-Shelf) have been used by space agencies for decades. Firstly, when they were the only technological choice that can meet requirements, usually dealing with performance. Traditionally, in both space and aeronautics fields the preferred choice for COTS is military or automotive grade parts, when they are available, because they are tested and designed for increased reliability requirements,for instance temperature.

Nowadays, space is not only accessed by space agencies, and the requirements, electronics used and available budget are in constant change. Many companies are launching nanosatellites constellations with limited budget and tight constraints in performance and size, power and weight. Such scenario focuses mainly in COTS. However, mission success, relies also in electronics reliability [22]. NASA guidelines [23] can provide more insight into the risk of using commercial parts and the tests that could be performed to qualify components. The limited budget scenario, also constraints the testing capabilities, radiation qualification tests are quite expensive, and some other approaches are considered instead of traditional component qualification.

Due to the increased interest in commercial circuits in radiation environments, researchers, space agencies and even manufacturers are carrying out radiation tests with commercial components. For example, in the near future, it is expected that the Zynq Ultrascale SOC FPGA will qualify as rad-tolerant and provide more FPGA resources than any other space-grade device (e.g., 10x more logic and 7x more memory compared to the Xilinx Virtex-5QV), as stated in [11]. Regarding the device under study in this work, some studies have been recently carried out to characterize its response to high energy irradiation [24, 25].

3 Exploring the Xilinx Zynq Ultrascale+ MPSoC

Nowadays, high end FPGAs contain not only programmable logic but a variety of components that make possible an outstanding computational capacity. This work is based in the Xilinx Zynq UltraScale+ MPSoC family [26] which is subdivided into three subfamilies intended for different application fields. The CG subfamily is intended for optimized industrial applications, covering among others: IoT, motor control, sensors, etc. The EG subfamily is intended for aerospace and defense applications, covering 5G communications and cloud computing, and the EV subfamily is intended for high definition video applications. All elements of all subfamilies contain a dual-core ARM Cortex R5 [27] and a quad core ARM cortex A53 [28]. The EV and EG subfamilies have also a low end Mali GPU [29] and only subfamily EV adds a H.264/H.265 video codec. In all MPSoC devices from Zynq UltraScale family the system is subdivided in two main parts:

– PS (Processing System), which contains all the microprocessors. On the one hand, the quad-core ARM Cortex-A53 that implements the ARM v8-A 64-

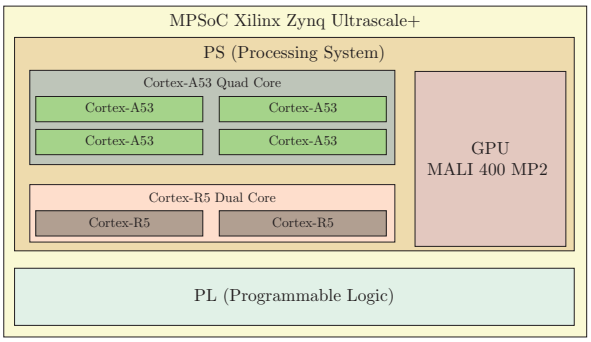

Fig. 1 Functional block diagram of the MPSoC Xilinx Zynq Ultrascale+ EG.

bit instruction set with frequency up to 1.5 GHz. We are accessing to these cores in two different ways : 1) by using the Xilinx Software Development Kit (SDK) [30]; and 2) by using the operating system Petalinux [31] which allows us to customize, build and deploy embedded Linux solutions on Xilinx processing systems. On the other hand, the dual-core ARM Cortex-R5 that belongs to the family of 32-bit RISC ARM v7-R processors can be used in *lockstep* mode (micro-synchronized dual execution), and in *split* mode (each core can work in parallel and executing different tasks). Besides those microprocessors, the PS also contains a Mali 400 MP2 GPU where OpenGL ES 2.0 can be used to perform general purpose computations, as shown in [32–34].

– PL (Programmable Logic), which contains the re-configurable logic. This is the part that can fit different hardware designs according to the needs of the application and the available resources. For this work, we have selected the development board Ultra96 [2] from Avnet which contains a Xilinx Zynq UltraScale+ MPSoC ZU3EG A484 FPGA of the EG subfamily. The selected device contains: 154K System Logic Cells, 141K CLB Flip-Flops, 71K CLB LUTS, 360 DSP slices and block RAM that makes a total of 7.6 MBytes. In order to design and optimize the hardware that is located in the Programmable Logic, we use the tool SDSoC [35], available in the Xilinx Environment.

Inside the device, multiple interconnection options between PL and PS are available, making possible high-speed data transfer suitable for the most demanding applications. Figure 1 shows a general scheme of the Xilinx Zynq UltraScale+ EG subfamily including its main components.

4 Experimental Setup and Evaluation

In order to assess the potential of the computational resources of the Xilinx platform, we evaluate an application that performs two image processing tasks using heterogeneous computing and a basic algebra operation that arises in multiple applications: matrix multiplication.

4.1 Image-based Applications

We have selected an application developed by Xilinx to test the possibilities offered by the heterogeneity of the platform. Specifically, we use the Targeted Reference Design (TRD), which is a video processing application distributed among different Processing Elements of the board [20]. This application demonstrates the possibility of offloading computation intensive tasks from the CPU to the FPGA. We can accelerate the computation and free-up the CPU cores to run other applications.

The application allows the user to choose between two image processing tasks: a 2D-convolution filter and the dense optical flow algorithm [36]. Both tasks are applied to processing the frames from a video source that can also be selected by the user. For example, the video source can be a webcam, a Test Pattern Generator (TPG) or even a file, among others. The application includes a graphic interface, that allows the user to choose if the 2D filter is executed on the Cortex-A53 CPU cores or offloaded to the FPGA in order to compare the performance in terms of CPU usage or frames per second. The resulting processed frames are shown in the screen connected to the board. Figure 2 summarizes the main software and hardware components of the TRD application.

One of the Cortex-R5 cores is devoted to monitor the memory throughput of the application reading the AXI performance monitors included in the PS. To this end, it runs a perfamp-server bare-metal application that uses OpenAMP communication framework to transfer the results to the perfamp-client code run on one of the A53 cores under Linux. Finally, the Mali GPU is used to show in an screen the GUI used to interact with the application and also the video stream resulting of applying the image processing algorithms to its successive frames.

The pure software implementation of the 2D filter is accomplished using OpenCV, while its FPGA-accelerated version and the optical flow algorithm are implemented using the Xilinx xfOpenCV library [37]. Both filters operate on YUYV pixel format, which uses 16 bits per pixel. However, both image processing algorithms are applied only to the 8 bits including the luma (Y) component of the pixels, which is essentially a grayscale image. The 2D filter uses a 3x3 window size to perform the convolution of successive frames of the video source. The optical flow computes the motion vector from two consecutive images by processing them in parallel on the FPGA (see [37, Chap. 6] for more details).

Table 1 shows the performance parameters obtained when the applications run only over the quad-core ARM Cortex-A53 for two different frame sizes; and when the applications run over the quad-core ARM Cortex-A53 in combination with the FPGA. The frame sizes are defined by the available resolutions of the screen used to show the processed video frames. We have also used the FPGA as video source, as it can generate up to 60 frames per second (fps) of the test pattern stressing the PEs of the platform to process them. Results using an A53 core as video source show that it can only generate 15 fps and those can

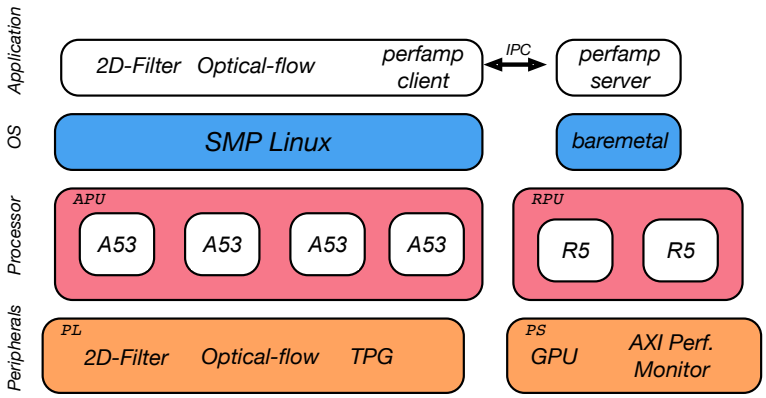

Fig. 2 Main hardware and software components of the image-based application.

be processed without producing much load on another core of the same kind and without having to use the FPGA.

Table 1 Performance parameters when the applications run only over the quad-core ARM Cortex-A53; and when the applications run over the quad-core ARM Cortex-A53 in combination with the FPGA for frame sizes of 1920x1080 and 1280x720. OF and 2D-F refer to the optical flow and the 2D filtering applications, respectively.

Frame Size	1920 x 1080				1280 x 720			
PE	A53		A53 + FPGA		A53		A53 + FPGA	
Application	2D-F	OF	2D-F	OF	2D-F	OF	2D-F	OF
CPU use (%)	100%	—	93%	94%	100%	—	44%	40%
frames per sec. (%)	5	—	60	60	12	—	60	60

As it can be seen, the use of the FPGA greatly accelerates the frames per second that can be managed by the MPSoC up to 60, while when using one ARM Cortex-A53 core, only 5 and 12 frames per second are processed for resolutions of 1920x1080 and 1230x720 pixels, respectively. Using the FPGA not only increases the frames per second that can be processed, but also reduces the load of the CPU core, specially with the lower resolution. Note also that the tackled application did not allow us to run the optical flow on the ARM Cortex-A53 cores. This fact is pointed with a dash at Table 1

These results demonstrate that the MPSoC can be leveraged as a full heterogeneous platform where applications with independent and irregular tasks (in terms of type of operation, loops, etc.) can be distributed among specific-tasks-oriented processing elements.

Many of the applications that can be executed on the Xilinx platform deal with different data types. For example, image processing algorithms usually deal with images stored as integers of different sizes, but in some cases the information is stored as floating point values. To test the effect of the data type

on the performance of the image processing algorithms we have implemented another version of the 2D filter algorithm that performs 3x3 convolutions on static images. Table 2 shows the performance of the algorithm with square images of different sizes stored using different data types. Results show the frames per second that can be processed on one Cortex-A53 core on each case. As expected, the fastest version is the one dealing with bytes as we use only 8 bits per element as opposed to the 32 bits used with the other two data types. The different performance obtained with floating point and 32-bit integers is due to the cost of the basic arithmetic operations used by the algorithm on the Cortex-A53 core.

Table 2 Frames per second processed by the 2D filter algorithm with different data types when applied to square images of size $n \times n$.

n	byte	int	float
32	107,948.2	111,405.2	82,913.9
64	40,000.0	25,641.0	16,666.7
128	15,151.5	6,369.4	4,273.5
256	4,629.6	1,618.1	1,089.3
512	1,404.5	392.8	245.6
1024	371.1	86.0	59.1

4.2 Matrix multiplication: a simple code

The aim of this experiments is to test that we are able to launch the same code in every processing element so that we can compare them and verify the flexibility of the platform. To this end, we have first selected a naive matrix multiplication implementation, known as *ijk*-algorithm, composed of three nested loops.

Figure 3 shows the performance that is achieved by every processing element when they run the *ijk*-algorithm in single precision. Broadly stated, on both CPUs the performance quickly decreases as the matrix size increases and memory-access is a bounding factor. Results show that the ARM Cortex-A53 cores are much more appropriate for this kind of computation, while the ARM Cortex-R5 cores offer the worst performance and should not be used for computation intensive tasks. Intermediate results are obtained by the FPGA, which can only manage sizes of matrices up to 256 because of the limited number of resources offered by the selected device. It is important to note that we have used the same code on every processing element, i.e. it is not a customized version to leverage the different PEs. In the case of the FPGA version of the product we have used the basic implementation provided by the Xilinx SDK. We show the results obtained with the best blocking factor allowed by the resources available on the platform when partitioning the matrices. That is, we

increased the number of blocks as much as possible for each of the matrix sizes to increase the parallelism of the implementation. To obtain the performance with the FPGA and GPU accelerators we have included in the execution time the time required to transfer data to the accelerator device and back.

Figure 3 also includes the results obtained with the multiplication implemented using OpenGL ES 2.0 and run on the Mali GPU. In this case we are using a specific implementation for this kind of processing element that produces results with limited floating point precision [32]. We can see that the performance using the Mali GPU is lower than the one achieved with both types of CPU cores for small matrices. However, as the GPU performance does not decrease with the size of the problem, it overcomes the naive implementation of the multiplication using both kinds of CPU cores with large matrices.

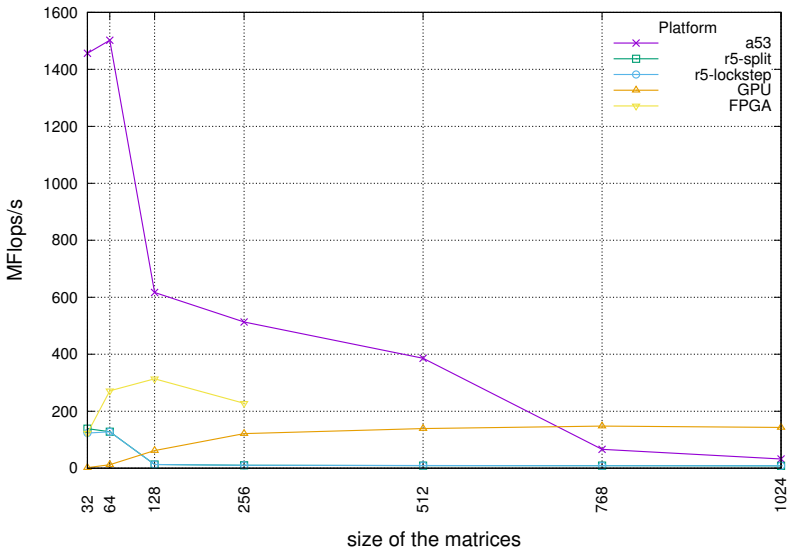

Fig. 3 Performance of the matrix multiplication (naive) for a different sizes of square matrices in the different Processing Elements of the Xilinx Ultrascale+ EG.

We have also tested an optimized version of the matrix multiplication [1] that tries to leverage the memory hierarchy of the processor. To this end it implements a block version of the product and unrolls the innermost loop. Figure 4 compares the performance of running this optimized code on one ARM Cortex-A53 core and one ARM Cortex-R5 core working in lockstep and split modes. This implementation clearly scales better than the naive version shown above, and improves the performance of the ARM Cortex-R5. Despite of this fact, the results confirm the superiority of the A53 architecture

[1] `https://github.com/deuxbot/fast-matrix-multiplication/blob/master/mxm.c`

with respect to the R5 for this kind of computation. On the other hand, the experiments show that the execution mode, lockstep or split, does not affect the performance of the product on the Cortex-R5 cores.

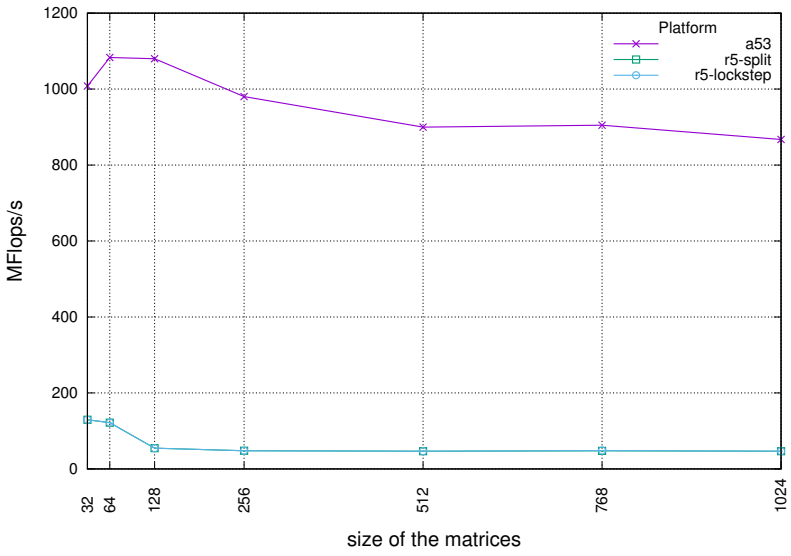

Fig. 4 Performance of an optimized version of the matrix multiplication that compares one core ARM Cortex-A53 and the ARM Cortex-R5 working in lock-step and split modes.

Finally, we have tested the effect of the data type on the performance of the matrix multiplication. We have compared the results of our first version of the matrix multiplication using single precision floating point values and 32-bit integer values on the FPGA component of the board. Table 3 shows the results obtained with matrices of different sizes. The `factor` values shown in the table correspond to the number of blocks used to partition both matrices involved in the multiplication using the `HLS array_partition` pragma. The table shows results obtained with the maximum factor allowed by the resources of the FPGA measured and the millions of arithmetic operations per second achieved. We can see that for the smaller matrices, the best performance is obtained with the floating point version of the multiplication. However, as we increase the size of the matrices, the integer version uses less resources than the floating point version and allows us to use larger partition factors, which results in better performances. Experiments also show that FPGA on the Xilinx UltraScale+ does not have enough resources to execute the multiplication with square matrices of size 512 on one step. It would be necessary to implement a block version of the multiplication dealing with blocks that could fit into the resources offered by the FPGA.

We have also compared the performance of both versions of the matrix multiplication on the two types of CPU cores of the board. The performances

obtained with floating point and 32-bit integer elements are very similar. This is because the simple version of the matrix multiplication is a memory bound algorithm. We have checked that the cost of the basic arithmetic operation is quite different when using floating point and integer elements both in the Cortex-A53 and Cortex-R5 cores. However the cost of the algorithm is clearly dominated by the cost of the memory accesses, which is the same when using floating point and 32-bit integer elements.

To complete our analysis of the effect of the data type, we have executed the OpenGL ES 2 implementation of the matrix multiplication on the Mali-400 GPU using floating point and 32-bit integer elements both signed and unsigned. The performance depends on the cost of the numeric transformations needed to store elements of different data types as texture and framebuffer values on the GPU memory as described in [32] . Our experiments show that the these transformations have a similar cost for the three data types thus producing similar performances for the three versions of the matrix multiplication.

Table 3 Performance of the matrix multiplication on the FPGA using floating point and integer elements with square matrices of size $n \times n$

n	float		int	
	MFlops	factor	MInts	factor
32	122.83	32	36.09	32
64	271.62	32	37.82	64
128	313.58	16	566.78	128
256	227.66	8	900.13	64

4.3 Matrix multiplication with OpenBLAS

OpenBLAS is an open source implementation of the BLAS (*Basic Linear Algebra Subprograms*) [38] which allows to launch parallel executions among all the cores by using OpenMP and is optimized at assembler level for each architecture. We have run the multiplication of two square matrices by using its corresponding invocation to the BLAS routine `sgemm`. The Petalinux operating system installed on the Xilinx platform does not provide the required Fortran compiler or the OpenMP libraries to compile OpenBLAS. Therefore, the library has been cross-compiled in a different platform with the same kind of ARM architecture: `ARMv8.0-A`. Once we have obtained the compiled version of the library, we have copied it to the Xilinx board, where we conducted the following experiments. Specifically, we have used the Jetson TX1 platform [39] and applied the compiling flags `-march=armv8-a -mtune=Cortex-A53` to obtain the library optimized for the kind of core included in the Xilinx board.

Figure 5 shows the performance that is achieved by the quad-core ARM Cortex-A53 of the Xilinx board when running a matrix multiplication varying the number of cores and the size of the square matrices. The sequential version of the code obtains a better performance than the block version previously shown, being highly stable with the size of the matrices around 3.5 GFlops. Using 2 cores we get an almost optimal speedup independently of the size of the problem. The performance increases when we use more than 2 cores, but only with sizes up to 768. We obtain a speedup of 3.5 and a maximum of 12 GFlops using four cores with matrices of size 512. With matrices larger than 768, the performance obtained with 3 or 4 cores is only slightly better than the one obtained with 2 cores, probably due to the shared memory access that bounds the parallelism of the computation on this architecture.

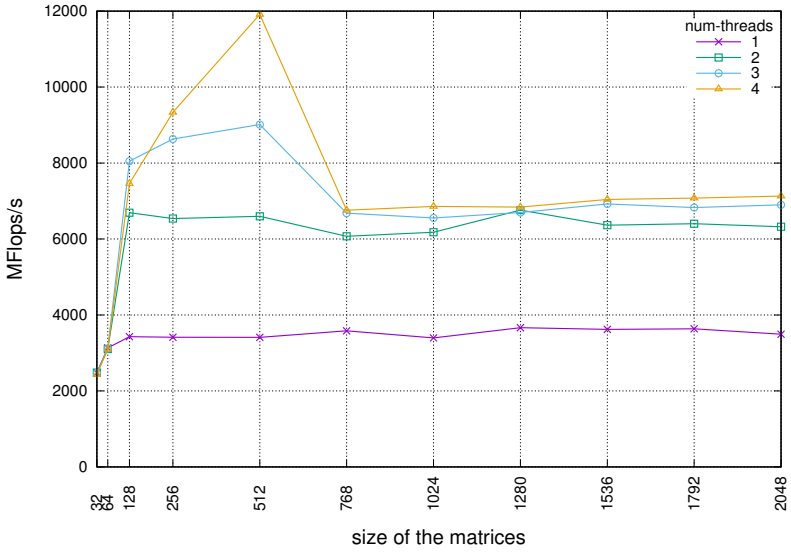

Fig. 5 Performance of the quad-core ARM Cortex-A53 when they run a matrix multiplication varying the number of cores and the size of the square matrices.

5 Conclusions

The use of Multi-processor System-on-Chip (MPSoC) in space environments is gaining momentum because of its flexibility and heterogeneity in terms of computational resources, since it combines different multi-core processors with a GPU and a FPGA. In spite of being used in different tests for space agencies and the fact that there are studies in the literature about these systems, none of them analyzes the computational capabilities of this kind of platforms. This work highlights the potential of combining suitably the computational

resources: ARM Cortex-A53, ARM Cortex-R5 and FPGA in imaged-based applications, where an acceleration of 12x in the number of frames per second is achieved.

Another important result that we must point out is the fact we are able to launch the same code over all the processing elements using different development environments. We have generated applications that can be executed on a Linux OS or baremetal (without OS). To analyze their computational capabilities, we have tested three sequential implementations of the matrix multiplication kernel that demonstrate that the ARM Cortex-A53 gets the best performance in terms of MFlops per second for this kind of computation. It is important to point out that the ARM Cortex-R5 could be used for computing issues if it is necessary. However their poor performance results indicate that it is better to assign them tasks related to control instead of intensive computation. The same behaviour occurs with the GPU included in the selected device (Mali 400 MP2 GPU), that despite of being able to perform computational tasks, is aimed to graphic processing.

Finally we have used the OpenBLAS library, a widespread mathematical framework, in order to evaluate the performance that can be obtained with the quad-core ARM Cortex-A53 when using an optimized parallel code to carry out a matrix multiplication. Results show that the peak performance is obtained for a size of 512 achieving up to 12 Gflops.

At this work, we have explored the computational power of each processing element (CPU, GPU and FPGA) and how they can interact with each other. An evaluation that was not previously tackled from the computational point of view. This work can help designers when choosing resources allocation according to specific computational requirements.

Acknowledgements

This work has been supported by the Spanish Government through TIN2017-82972-R, ESP2015-68245-C4-1-P, the Valencian Regional Government through PROMETEO/2029/109 and the Universitat Jaume I through UJI-B2019-36. We thank Prof. L. Kosmidis and M. M. Trompouki for providing us the OpenGL ES 2.0 code implementation of the matrix multiplication.

References

1. W. Wolf, A. A. Jerraya, and G. Martin, "Multiprocessor system-on-chip (mpsoc) technology," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 10, pp. 1701–1713, Oct 2008.
2. Avnet Inc, "Ultra96-V2 Hardware user guide. Version 1.0," 2019.
3. A. Tumeo, M. Ceriani, G. Palermo *et al.*, "Real-time considerations for rugged embedded systems," in *Rugged Embedded Systems*. Boston: Morgan Kaufmann, 2017, pp. 39 – 56.
4. M. Berg, "NEPP Update of Independent Single Event Upset Field Programmable Gate Array Testing," *NASA NEPP EEE Parts for Small Missions Workshop*, 2014.

5. S. M. Guertin, "Candidate CubeSat Processors," *NASA NEPP EEE Parts for Small Missions Workshop*, 2014.
6. NASA, Mission Design Division Ames Research Center, "Small spacecraft technology state of the art," *NASA/TP-–2015-–216648/REV1*, 2015.
7. NASA, "HyspIRI Final Report," September 2018.
8. L. Kosmidis, J. Lachaize, J. Abella, O. Notebaert, F. J. Cazorla, and D. Steenari, "GPU4S: Embedded GPUs in Space," in *22nd Euromicro Conference on Digital System Design, DSD 2019, Kallithea, Greece, August 28-30, 2019*, 2019, pp. 399–405.
9. European Space Agency, "General Benchmarking and Specific Algorithms: NIR HAWAII-2RG BM," 2012.
10. ——, "Next Generation Space Digital Signal Processor (NGDSP) benchmarks from ESA," 2009.
11. G. Lentaris, K. Maragos, I. Stratakos *et al.*, "High-performance embedded computing in space: Evaluation of platforms for vision-based navigation," *Journal of Aerospace Information Systems*, vol. 15, no. 4, pp. 178–192, 02 2018.
12. G. Mitra, B. Johnston, A. Rendell *et al.*, "Use of SIMD Vector Operations to Accelerate Application Code Performance on Low-Powered ARM and Intel Platforms," in *IEEE 27th International Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW)*, May 2013, pp. 1107–1116.
13. M. F. Cloutier, C. Paradis, and V. M. Weaver, "A Raspberry Pi Cluster Instrumented for Fine-Grained Power Measurement," *Electronics*, vol. 5, no. 4, 2016.
14. B. W. T. Peng, X. Jin, and C. Wang, "An accelerating solution for n-body MOND simulation with FPGA-SoC," *International Journal of Reconfigurable Computing*, pp. 1–11, 2016.
15. B. J. Tippetts, D. Lee, K. D. Lillywhite, and J. K. Archibald, "Review of stereo vision algorithms and their suitability for resource-limited systems," *J. Real-Time Image Processing*, vol. 11, no. 1, pp. 5–25, 2016.
16. B. E. Tweddle, "Computer vision based navigation for spacecraft proximity operations," Master's thesis, Massachusetts Inst. of Technology, Cambridge, MA, 2010.
17. D. Rudolph, C. Wilson, J. Stewart *et al.*, "CSP: A Multifaceted Hybrid Architecture for Space Computing," in *Proceedings of the 28th Annual AIAA/USU Conference on Small Satellites*, 2014.
18. X. Iturbe, D. Keymeulen, E. Ozer *et al.*, "An integrated SoC for science data processing in next-generation space flight instruments avionics," in *2015 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, Oct 2015, pp. 134–141.
19. G. Lentaris, I. Stamoulias, D. Diamantopoulos *et al.*, "Spartan/sextant/compass: Advancing space rover vision via reconfigurable platforms," in *Applied Reconfigurable Computing*, K. Sano, D. Soudris, M. Hübner, and P. C. Diniz, Eds., 2015, pp. 475–486.
20. Xilinx Inc., "Zync UltraScale+ MPSoC Base Targeted Reference Design. User Guide. UG1221. v2018.2," 2018.
21. J. D. Cressler, *Extreme environment electronics*, ser. Industrial Electronics. Hoboken, NJ: CRC Press, 2012.
22. C. M. Fuchs, P. Chou, X. Wen *et al.*, "A Fault-Tolerant MPSoC For CubeSats," in *2019 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, DFT 2019, Noordwijk, Netherlands, October 2-4, 2019*, 2019, pp. 1–6. [Online]. Available: https://doi.org/10.1109/DFT.2019.8875417
23. O. Gonzalez, Y. Chen, R. L. Ladbury *et al.*, "Guidelines for Verification Strategies to Minimize RISK Based on Mission Environment, - Application and -Lifetime (MEAL)," Tech. Rep. NASA/TM-2018-220074 NESC-RP-16-01117, June 2018.
24. D. Lee, M. King, W. Evans *et al.*, "Single-Event Characterization of 16 nm FinFET Xilinx UltraScale+ Devices with Heavy Ion and Neutron Irradiation," in *2018 IEEE Radiation Effects Data Workshop (REDW)*, July 2018, pp. 1–8.
25. M. Glorieux, A. Evans, T. Lange *et al.*, "Single-Event Characterization of Xilinx UltraScale+ ® MPSOC under Standard and Ultra-High Energy Heavy-Ion Irradiation," in *2018 IEEE Radiation Effects Data Workshop (REDW)*, July 2018, pp. 1–5.
26. Xilinx Inc, "Zynq UtraScale+ MPSoC Data Sheet: Overview," *DS891 (v1.7)*, 2018.
27. ARM, "Cortex-R5. Technical Reference Manual. Revision r1p2," 2011.
28. ——, "ARM Cortex-A53 MPCore Processor. Technical Reference Manual. Revision r0p4," 2016.

29. T. Olson, "Mali-400 MP: a scalable GPU for mobile devices," in *Hot3D Session. Proc. International Conference on High Performance Graphics*, 2010.
30. Xilinx Inc., "Embedded System Tools Reference Manual," *UG1043 (v2019.1)*, 2019.
31. Xilinx Inc, "Petalinux Tools Documentation," *UG1144 (v2018.3)*, 2018.
32. M. M. Trompouki and L. Kosmidis, "Towards general purpose computations on low-end mobile GPUs," in *2016 Design, Automation & Test in Europe Conference & Exhibition, DATE 2016, Dresden, Germany, March 14-18, 2016*, 2016, pp. 539–542.
33. ——, "Optimisation opportunities and evaluation for GPGPU applications on low-end mobile GPUs," in *Design, Automation & Test in Europe Conference & Exhibition, DATE 2017, Lausanne, Switzerland, March 27-31, 2017*, 2017, pp. 950–953.
34. ——, "Brook auto: high-level certification-friendly programming for GPU-powered automotive systems," in *Proceedings of the 55th Annual Design Automation Conference, DAC 2018, San Francisco, CA, USA, June 24-29, 2018*, 2018, pp. 100:1–100:6.
35. Xilinx Inc, "SDSoC Environment User Guide," *UG1027 (v2017.4)*, 2018.
36. J. L. Barron, D. J. Fleet, and S. S. Beauchemin, "Performance of optical flow techniques," *International Journal of Computer Vision*, vol. 12, pp. 43–77, 1994.
37. Xilinx Inc., "Xilinx OpenCV User Guide. UG13233. v2019.1," 2019.
38. J. Dongarra, J. D. Croz, S. Hammarling, and R. J. Hanson, "A proposal for an extended set of Fortran basic linear algebra subprograms," *ACM Signum Newsletter*, pp. 2–18, 1985.
39. NVIDIA Corporation, "Jetson TX1 Developer Kit. User Guide," 2016.