Evaluating the Computational Capabilities of Embedded Multicore and GPU Platforms for On-Board Image Processing

Ivan Rodriguez-Ferrandez*†‡, Leonidas Kosmidis*†, Matina Maria Trompouki*†,
David Steenari‡, Francisco J. Cazorla†

*Universitat Politècnica de Catalunya (UPC), Barcelona, Spain
†Barcelona Supercomputing Center (BSC), Barcelona, Spain
‡European Space Agency (ESA), Noordwijk, The Netherlands

***Abstract*—On-board Image Processing is one of the most computationally intensive tasks performed in space payload processing. With the constant increase in the resolution of optical sensors, new powerful architectures are needed for their processing. In this work, we evaluate the capabilities of several state-of-the-art embedded COTS and radiation tolerant multicore and GPU featuring platforms, with an open source on-board image processing application we have developed and parallelised. Our results show that both embedded multicores and especially GPUs are very effective for such tasks. In fact, GPUs become even more efficient when sensor sizes are increasing, making them ideal candidates for future space missions.**

***Index Terms*—multicore, GPU, on-board image processing**

I. Introduction

Data obtained with space optical instruments need to be processed on-board to compensate for several issues either due to sensor deficiencies or the space radiation environment. This type of space processing is among the most computationally intensive ones for payload processing. For this reason, frequently such processing is not possible on conventional space processors and therefore is implemented in FPGAs. However, FPGA development requires significant expertise which is not always available. Moreover, significant effort is required and has a long development lifecycle, which can increase significantly the mission cost and delay its schedule.

Furthermore, there is a constant trend for increasing sensors resolutions which has a superlinear effect in the computational power required for their processing. This calls for new, more powerful processing devices to be used for on-board processing. Embedded multicores and graphics processing units (GPUs) are potential candidates for future on-board processing due to their increased computational power with low power consumption. In addition, several GPUs are designed to comply with the requirements of the automotive market, which has several reliability similarities with the space domain. More importantly, these parallel platforms are much easier to program in software compared to the hardware development of FPGAs and there is a much larger pool of experienced software developers with their programming models.

An earlier study [1] has shown that embedded GPUs are a good fit for on-board image processing tasks, by porting ESA's Euclid NIR (Near Infrared) image processing algorithm. However, this image processing algorithm is tightly connected to the Euclid optical instrument and not representative of more on-board image processing pipelines. Moreover, this algorithm which has been used in many ESA's studies as a benchmarking use case, does not come with real imaging data, but contains a random input generator utility. Since evaluations from unrelated parties may use different random inputs, and the application performance is heavily dependent on the particular input, result reproducibility and performance comparison of different devices and implementations is complicated. Last but not least, the Euclid NIR is not open source, since its intellectual property is owned by ESA, and therefore is only available to entities from ESA member states for ESA projects.

For this reason, in this work we use a generic, open source implementation of an on-board image processing algorithm developed in the ESA-funded GPU4S project [2], which is part of ESA's OBPMark benchmark suite [3] [4].

We have parallelised the #1.1 image processing benchmark from OBPMark using OpenMP, CUDA and OpenCL and we are evaluating several embedded multicore and GPU platforms which have been identified as potential candidates for future on-board processing in the GPU4S project [2], including NVIDIA Orin, the latest NVIDIA embedded platform for automotive. In addition, we include in the comparison other multicore and GPU platforms for the space domain, including devices with flight heritage. Our results show that although multicore processing is very effective for on-board image processing, GPUs are an ideal architecture for this task achieving very high processing rates, which increase with the image size.

II. On-Board Image Processing SW

The application software is based on the typical on-board image processing function of image calibration and correction. Unlike Euclid NIR [1], it is more general and representative of the on-board processing tasks necessary for panchromatic imaging instruments in scientific remote sensing applications, such as deep-space telescopes with long exposure times. Frames from several acquisitions are pre-processed individ-

 https://dx.doi.org/10.23919/EDHPC59100.2023.10395928

ually, then co-registered and summed to create the final image output.

A. High Level Application Overview

The stages image offset correction, bad pixel correction, radiation scrubbing, gain correction, spatial binning and temporal binning are performed for each frame in series. The output of each processing step is used by the subsequent one in a pipelined fashion, as shown in Figure 1.

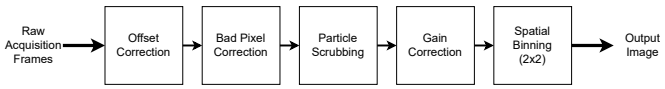

Fig. 1. General Pipeline

However, each stage uses a different number of frames for its computation. For example, radiation scrubbing (Section II-A3) is performed over 5 frames as shown in Figure 2. For this reason, we need to maintain a buffer of 5 frames.

This divides the pipeline in two distinct parts, which we refer to as first and second part respectively. The first part of the pipeline consists of all of the operations before the buffer and the second part contains the computations after the buffer.

Frames $t-2$ and $t-1$ of the buffer are introduced in the buffer in order to start the processing. These two frames represent frames that the second part has already finished processing. Frames 0 and 1 in the buffer are frames that only the first part has been completed processing. Figure 2 shows the state of the pipeline for the first frame.

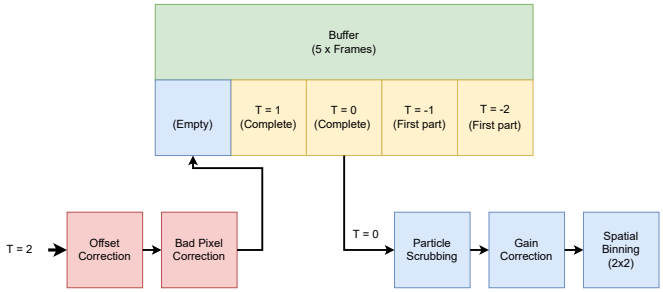

Fig. 2. Initial state of the pipeline for the first frame.

When Frame 2 enters the pipeline, it goes through the first part of the pipeline and when it finishes that computation, the result is stored in the buffer. Then, Frame 0 can be sent to the second part of the pipeline, because its previous two frames ($t-1$ & $t-2$) and its next two frames are ready ($t+1$ & $t+2$). When the second part finishes for the frame 0, the result is stored in the buffer and the frame $t-2$ is removed to create an empty slot.

1) Image Offset Correction: This processing step is used to reduce image noise from the sensor, which is present due to long exposure times. In order to compensate for this effect, a *dark* frame is acquired with closed shutter, creating a configurable offset value, which is subtracted for each frame from each pixel.

The computation is carried out as a 16-bit unsigned saturated subtraction, however it only results in positive values, i.e. the result is not wrapped-around below zero but it is clamped at 0.

2) Bad Pixel Correction: For each frame a correction is made for pixels marked as bad (deficient) in the optical sensor. The state of each pixel (good/bad) is defined in a configurable look-up table (M). In case the pixel is marked as bad, it is replaced based on the values of its neighbouring pixels.

The neighbourhood replacement function $F(I_{x,y})$ depends on whether or not the pixel is on the edge of the image or not, i.e. by the values of x and y, and if the neighbouring pixels are also bad pixels or not.

The average of all good pixels in the 3x3 neighbourhood are calculated and used as the replacement value. If the pixel is on the left or right edge of the frame, the neighbourhood is 2x3 in size. If the pixel is on the top or bottom edges of the frame, the neighbourhood is 3x2 in size. If the pixel is in any of the four corners of the frame, the neighbourhood size is 2x2 pixels.

The calculation is performed using unsigned 32-bit variables and the division is calculated as an unsigned 32-bit division. The output result is typecasted to an unsigned 16-bit variable.

3) Radiation Scrubbing: For each frame, each pixel is screened for radiation upsets, and if detected, scrubbed. Each pixel is compared to a constant multiplied with the average of its temporal neighbours in the +/- 2 non-scrubbed frames.

Then for each pixel, the value is compared to the average (multiplied by a constant, 2 in our case), and if the current pixel value is larger, the pixel is scrubbed.

The processing is performed using a 32-bit computation. When a pixel is replaced, the replacement value is typecasted to an unsigned 16-bit variable.

Note that for the first two and the last two frames in each data sets where previous ($t-2$, $t-1$) and following ($t+1$, $t+2$) frames are not included in the data set for the current image. Instead, auxiliary "Scrub frames" are used. All 12 frames (input frames and additional scrub frames) can be pre-stored in internal frame buffers for simplicity.

4) Gain Correction: For each frame a configurable reference gain value is multiplied with the value of each pixel.

The offset value is unique per pixel position and is stored as an integer value map with the same size and bit-depth as the raw frames.

The computation is carried out as a 32-bit unsigned multiplication, followed by a right shift.

Gain values range from 0.950 to 1.050, or 0x799A to 0x8666 as UQ1.15 numbers. It is important to note that 0x8000 is equivalent to 1.0 in UQ1.15. Using this method during the computation results in loss of one bit of precision.

5) Spatial Binning: For each frame, blocks of 2x2 pixels are summed to generate a new frame with size equivalent to one quarter of the original frame.

Note that the output image has a smaller spatial size than the input image. However, the number of bits per pixel increase due to the summation. The summation is carried out as unsigned 32-bit additions and the output data has the same type.

6) Temporal Binning: For each set of frames, an image is generated by summing the values (pixel by pixel) of each pixel in the frame, for 8 frames.

The summation is carried out with unsigned 32-bit additions and the output data is unsigned 32-bit.

B. Input and Verification Data

The input and verification data of this application have been taken from ESA's Herschel mission [5]. However, as mentioned earlier the processing pipeline is not identical to that mission, but is representative of the image processing pipelines of several optical instruments.

The base input data of this benchmark come from the Herschel mission [5]. In particular the image observation ID is 1342218967, and target name OrionA-N-1. The image is acquired with the SPIRE Instrument and is of image level 3. The various image sizes are produced from the same image, with up-scaling for the 4096 image size.

The sizes have been selected based on the ones used in existing missions ($1K \times 1K$, $2K \times 2K$) as well as for sizes which are likely to be used in future missions ($4K \times 4K$). Each application configuration uses 8 frames with 16bit per pixel, as well as 4 additional scrub frames.

III. Evaluation

A. Software Parallelisation

The application is parallelised using OpenMP for the evaluation of the multicore CPUs and in CUDA and OpenCL for GPUs. Our implementations are provided in the official OBPMark repository [4].

All implementations produce identical results to the sequential reference implementation of the application which is written in C.

All tests are using the pre-defined sizes of 1024 by 1024, 2048 by 2048 and 4096 by 4096 pixels of 16 bit per pixel. All of the devices are tested with these sizes. In the OCE platform, the 4K size was not possible to be executed due its limited RAM capacity.

B. Experimental Setup

As target platforms we have selected a number of embedded COTS multicore and GPU platforms which have been identified as good candidates for future space processing in the GPU4S ESA funded project [2]. These include AMD's V1605B which is used in Unibap's iX10-100, NVIDIA's TX2 and Xavier, the first automotive grade embedded platform of NVIDIA. Moreover, we include the latest automotive grade NVIDIA GPU platform, the NVIDIA Orin, which is the first platform to include the automotive ARM Cortex-A78AE CPU.

In addition to the GPU featuring boards, we have included also some additional multicore devices which are considered good candidates for future on-board processing. The Zynq Ultrascale+ (ZCU102) from AMD/Xilinx is an FPGA SoC, which includes 4 A53 hard cores from ARM, which we evaluate. Moreover, we use NXP's Layerscape LS1046A and LX2160A, both of which are targeting automotive, avionics and aerospace applications.

Also from the space domain, we included two radiation tolerant devices and a radiation hardened device used in different missions. First we have the Unibap iX5-100 (predecessor of the iX10-100) featuring a AMD x86 System. Second, we evaluated the OCE HISAOR-03A, which has a 4-core ARM A9 processor and a Vivante VIP8000 GPU-like Neural network accelerator which is programmable using OpenCL. Finally, we include Frontgrade Gaisler's GR740, which is a quad-core platform based on the SPARC v8 LEON4 fault tolerant CPU.

Table I provides a summary of the hardware capabilities of each board. The NVIDIA devices include predefined power modes which limit the TDP (thermal design power) of the board. According to the work performed in the GPU4S project [2], COTS devices to be used for on-board processing shall limited to a 15W TDP at most, for thermal dissipation reasons, since no active cooling is possible in space for small satellites. However, since the TDP of the NXP boards exceed that limit, we include also a 30W power mode for the Orin platform, for a fair comparison.

For each power mode we specify how many processing elements (CPUs and GPU shader cores) are enabled. For the AMD platform we used a custom made power profile which is similar to the 15W of the NVIDIA Xavier. For the rest of the platforms, we have used the manufacturer provided TDP. Also for some of the NVIDIA devices, we provide an alternative power mode. These power profiles have been created using the NVIDIA power estimator tool [6]. This allow us to create custom modes (which we mark as *Cus* (Custom) in the Figures) that benefit each of the processing systems (CPU or GPU) and while maintaining the same power budget with predefined modes (marked as *Def* (Default) in the Figures). In this way we can improve further the performance of the devices. It is worth noting that for all devices we use the full power consumption of the board.

Finally, all experiments have been performed under Linux and the end-to-end application time has been measured. For all of the presented results we measure the full execution time, which in the case of the GPU implementations includes the memory copies between the host and device. With that execution time, we compute the processing throughput of the device. This performance metric, which is measured in MPixels/s, includes the processing of the pixels of the 8 frames.

TABLE I
HARDWARE DETAILS OF THE EVALUATED EMBEDDED COTS MULTICORE AND GPU PLATFORMS

Board	Number of Cores	CPU Frequency (MHz)	Memory Capacity	Memory Frequency (MHz)	Number of GPU Multiprocessors	Frequency of the Multiprocessors	Power Consumption (W)
NVIDIA TX2	4 ARM A57 & 2 Denver Cores (4 A57 Enabled and 2 Denver Disabled)	1200	8 GB (Shared between CPU and GPU)	1331	2 SMs Pascal (Streaming Multiprocessors)	850	7.5
NVIDIA TX2	4 ARM A57 & 2 Denver Cores (4 A57 Enabled and 2 Denver Disabled)	2000	8 GB (Shared between CPU and GPU)	1600	2 SMs Pascal	1120	15
NVIDIA Xavier IND	4 ARM v8.2 Carmel Cores (4 Enabled and 4 Disabled)	1200	16 GB (Shared between CPU and GPU)	1600	8 SMs Volta	670	20
NVIDIA Xavier IND	8 ARM v8.2 Carmel Cores	1200	16 GB (Shared between CPU and GPU)	1600	8 SMs Volta	900	40
NVIDIA Xavier NX	4 ARM v8.2 Carmel Cores (4 Enabled and 2 Disabled)	1400	8 GB (Shared between CPU and GPU)	1600	6 SMs Volta	1100	15
NVIDIA Xavier NX	6 ARM v8.2 Carmel Cores	1400	16 GB (Shared between CPU and GPU)	1866	6 SMs Volta	1100	20
NVIDIA Xavier NX CPU Optimised	6 ARM v8.2 Carmel Cores	1900	8 GB (Shared between CPU and GPU)	1866	6 SMs Volta	306	15
NVIDIA Xavier NX GPU Optimised	2 ARM v8.2 Carmel Cores (2 Enabled and 4 Disabled)	1900	8 GB (Shared between CPU and GPU)	1600	6 SMs Volta	1100	15
NVIDIA Xavier NX CPU Optimised	6 ARM v8.2 Carmel Cores	1900	8 GB (Shared between CPU and GPU)	1866	6 SMs Volta	900	20
NVIDIA Xavier NX GPU Optimised	2 ARM v8.2 Carmel Cores (2 Enabled and 4 Disabled)	1900	8 GB (Shared between CPU and GPU)	1866	6 SMs Volta	1100	20
NVIDIA Orin AGX	4 ARM v8.2 Cortex A78AE Cores (4 Enabled and 8 Disabled)	1113	32 GB (Shared between CPU and GPU)	2133	14 SMs Ampere (6 Enabled and 8 Disabled)	420	15
NVIDIA Orin AGX	8 ARM v8.2 Cortex A78AE Cores (8 Enabled and 4 Disabled)	1728	32 GB (Shared between CPU and GPU)	3200	14 SMs Ampere (8 Enabled and 6 Disabled)	624	30
NVIDIA Orin AGX CPU Optimised	10 ARM v8.2 Cortex A78AE Cores (10 Enabled and 2 Disabled)	1888	32 GB (Shared between CPU and GPU)	2133	14 SMs Ampere (6 Enabled and 8 Disabled)	408	15
NVIDIA Orin AGX GPU Optimised	2 ARM v8.2 Cortex A78AE Cores (2 Enabled and 10 Disabled)	960	32 GB (Shared between CPU and GPU)	2133	14 SMs Ampere (6 Enabled and 8 Disabled)	918	15
NVIDIA Orin AGX CPU Optimised	12 ARM v8.2 Cortex A78AE Cores	2200	32 GB (Shared between CPU and GPU)	3200	14 SMs Ampere (8 Enabled and 6 Disabled)	1300	30
NVIDIA Orin AGX GPU Optimised	2 ARM v8.2 Cortex A78AE Cores (2 Enabled and 10 Disabled)	1888	32 GB (Shared between CPU and GPU)	3200	14 SMs Ampere (14 Enabled and 2 Disabled)	1020	30
AMD V1605B	2 x86 Ryzen Cores (with hyperthreading disabled)	1600	16 GB (Shared between CPU and GPU)	1067	8 CUs Vega (Compute Units)	1100	∼12.5
AMD V1605B	4 x86 Ryzen Cores (with hyperthreading disabled)	1600	16 GB (Shared between CPU and GPU)	1067	8 CUs Vega	1100	∼15
Xilinx ZCU102	4 ARM A53 Cores and 2 ARM R5 Disabled	1500	4 GB	1200	NA	NA	∼22.8
NXP LS1046A	4 ARM A72 Cores	1800	8 GB	1050	NA	NA	∼19
NXP LX2160A	16 ARM A72 Cores	2200	8 GB	1600	NA	NA	∼30
Unibap iX5-100	4 x86 Jaguar AMD Cores	1200	2 GB (Shared between CPU and GPU)	N/A	2 CUs Radeon	351	∼15
OCE HISAOR-03A	4 ARM A9 Cores	1000	1 GB (Shared between CPU and GPU)	1333	8 PPU VIP8000 (Parallel Processing Unit)	1000	∼5
Frontgrade Gaisler GR740	4 LEON4FT Cores Fault Tolerant SPARC V8	250	512 MB	100	NA	NA	1.5

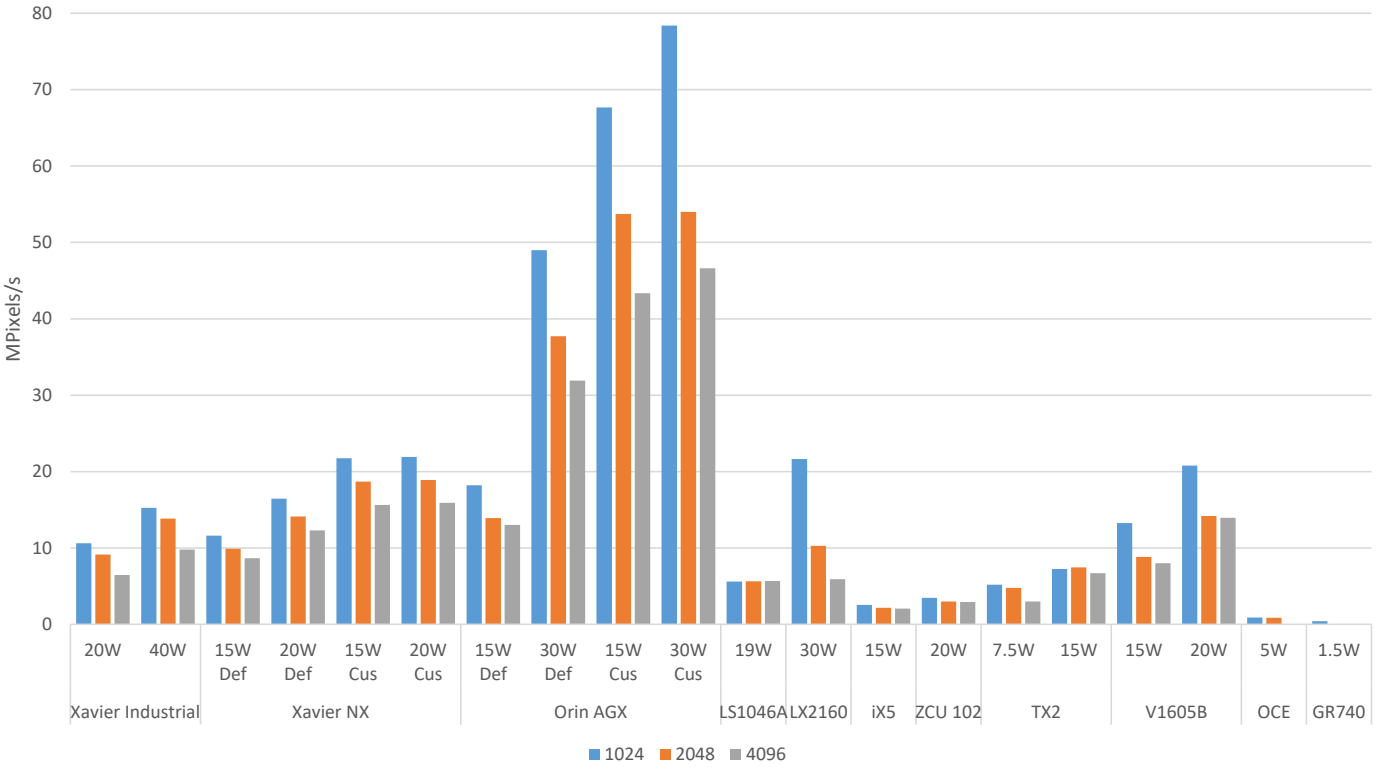

Fig. 3. Multicore Performance

C. Multicore Performance

Figure 3 shows the processing capability of the multicores. We notice that for all configurations, the processing capability of the CPU is dropping when the size of the image is increased. Likely this is because smaller image sizes fit in the CPU L2 shared cache. This effect is more pronounced in the NVIDIA Orin, which achieves the highest processing rate with 78.3 MPixels/s for the $1K \times 1K$ image size at the 30W custom power mode which is optimised for CPU performance. This is 60% higher than the performance obtained with the default power mode.

As it is expected, the higher the performance mode in a given platform, the higher the achieved performance. The reason for this is that a higher performance mode uses either more CPUs or the same number of CPUs with a higher frequency. However, the performance increase is not linear. For example in the NVIDIA TX2 the performance is 3 times higher when the TDP is doubled. Note that in the NVIDIA Xavier NX, the 20W performance mode uses two more CPUs with the same frequency compared to the 15W mode and the increase is moderate. On the other hand, when comparing the default and the custom power modes we can observe that in the case of the 15W, the performance of the custom mode is double under the same TDP. Moreover, it is interesting to note that the performance for the 15W and 20W custom modes is very similar. This is due to the fact that their difference is only in the frequency of the GPU, which is not used in the multicore execution.

NVIDIA Orin is a great example of the effects of adjusting the power distribution using the custom power modes. The 15W mode of the Orin provides almost twice of the performance of the NX using the default mode. If we compare the custom modes, this effect is even more confirmed. If we compute the MPixel/s per core we can see that with the NX we achieve 3.6 MPixels/s per core while with the Orin we manage to achieve 6.7 MPixels/s per core (both in the 1K size). This demonstrates the performance improvement between the two generation of cores, when they are given the maximum possible configuration with the same power envelop.

Similarly, when the power budget of the Orin is doubled, using twice as many CPUs and with a higher clock frequency, the achieved performance is 3 times higher.

The Zynq Ultrascale+ is the platform with one of the lowest multi core performance only behind the three radiation tolerant and hardened devices, the Unibap iX5-100, OCE and GR740. This result is really interesting, since this platform is an FPGA SoC, which can be used to implement custom processing solutions, which however are not evaluated in our work. Moreover, it is faster than the x86-based iX5 system, which uses a very old AMD CPU.

The NXP LS1046A provides lower performance compared with the TX2 in the 15W power mode, while in the 7.5W mode the performance is similar. In contrast, in the other NXP board, the LX2160, we can observe that with its increased number of cores (16) results in the second best performance for 1K processing behind Orin, and very similar performance with the AMD V1605B. However, this high performance only happens

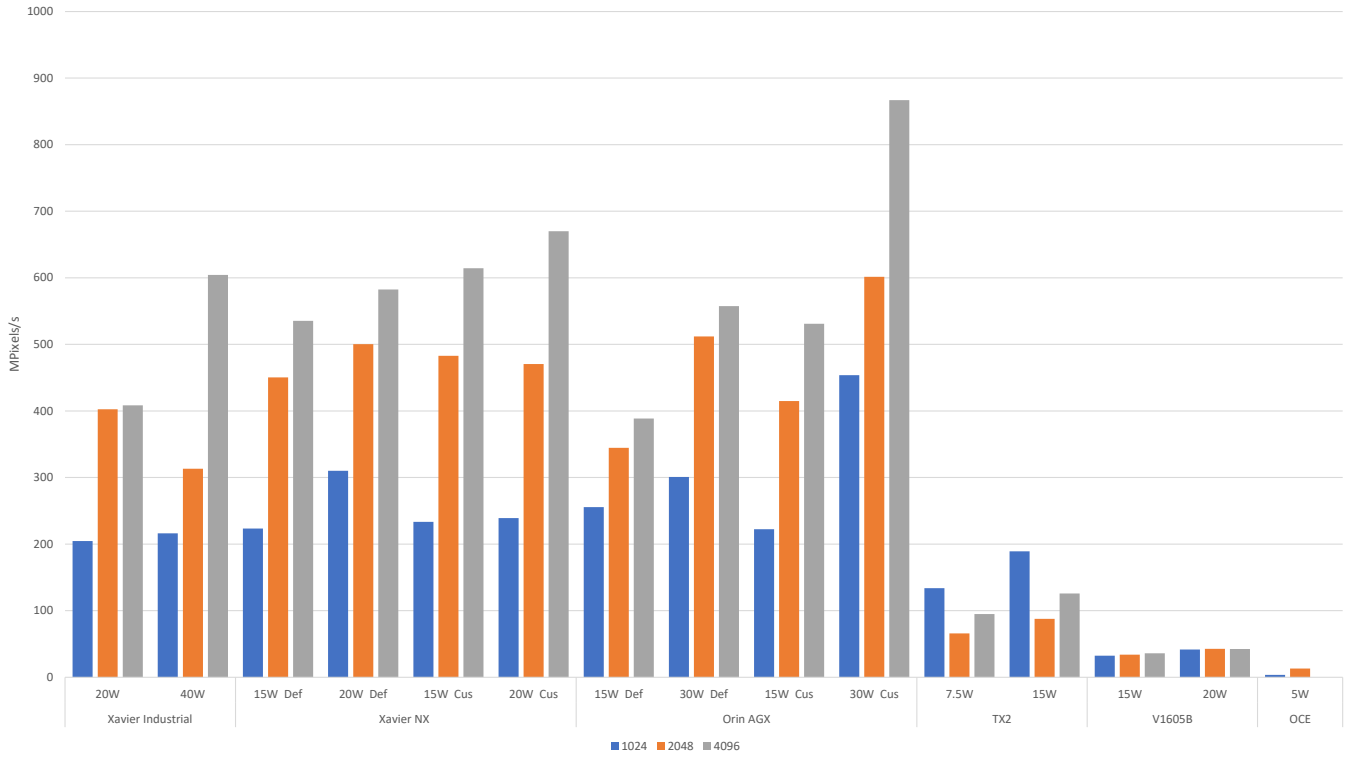

Fig. 4. GPU Performance

with the 1K size. When we compare it with the 2K or 4K we can observe a substantial decrease in performance. This is because for the 1K size we can have most of the processing with data on the caches, but for the larger sizes, the data does not fit in caches and need to be fetched from main memory, resulting in a significant performance decrease.

For the radiation tolerant and radiation hardened processors used in space, we observe that they have the lowest performance over all evaluated devices, as it was expected due to the older, reliable technology that are based on. The OCE board and the GR740 provide the lowest multicore performance among all tested devices. In the case of OCE, this is mainly attributed to the fact that the A9 core is an ARM CPU design from 2007 with lower clock speeds. GR740 provides half of the OCE processing performance, but it is the only radiation hardened device among the evaluated ones, which can provide very high reliability. Moreover, given that it has 3x lower TDP than the OCE (only 1.5W the lowest among all the evaluated devices), GR740 has a better energy efficiency.

Overall, at the 15W budget which has been identified as the upper bound for the power consumption for on-board processing devices, if we overlook the custom performance modes, the highest performance is provided by the Orin, followed by the AMD V1605B platform, and the NVIDIA Xavier NX at the default 15W power mode. All 3 platforms achieve this high performance using 4 cores. However, if we consider the custom modes, Orin provides the highest performance, with almost twice the performance of the Xavier NX, using 10 and 6 CPUs respectively.

D. GPU Performance

Figure 4 shows the processing capability of the GPU implementations. For all devices and power modes, we notice that the processing rate is improving when the size of the image is increased. In particular, the NVIDIA Xavier NX at the 15W custom power mode provides a significantly high processing rate, up to 670 Mpixels/s for the $4K \times 4K$ size, which is 5 times higher than the best multicore configuration (NVIDIA Orin for the $1K \times 1K$ image size but in the 30W custom performance mode). Larger image sizes can potentially provide much higher processing as we have seen in [7] for Euclid NIR, which makes GPUs very promising for on-board image processing tasks.

As in the multicore CPU results, the increase of the power mode results in higher performance, as expected. However, for the NVIDIA TX2, the opposite effect is observed. When moving from the 7.5W to the 15W, the performance is not doubled. Interestingly, the GPU performance of the NVIDIA Xavier NX is almost double compared to NVIDIA Orin's GPU performance under the same 15W TDP. This can be explained from the fact that Xavier uses 8 SMs compared to the 6 SMs used by Orin and with higher frequency (670 MHz vs 420 MHz).

Another interesting observation is that Orin under the default 30W power mode provides slightly higher performance than the Xavier in the 15W but only for the $1K \times 1K$ image size. For higher image sizes the Xavier provides significantly

better performance than the Orin, in which the performance is mostly saturated.

Using the custom power modes we can improve the performance for the 30W TDP achieving the maximum recorded processing of 866 MPixels/s for Orin at the $4K \times 4K$ image size. This performance is an order of magnitude higher than the best multicore performance from Figure 3. It is worth mentioning the efficiency of the Xavier in the 15W custom power mode, which can exceed the one of the Orin even in the 15W custom power mode.

The AMD V1605B provides the highest performance of the non-CUDA devices, which is 3-6 times slower than the the NVIDIA TX2. The lowest performance is obtained by the OCE, as expected, due to the older technology which is using compared with the COTS devices. An important point to mention is the huge improvement obtained on the OCE by switching from multicore to the GPU. The multicore performance is 0.88 MPixels/s, but for the GPU the performance is 13.1 MPixels/s, which is more than 13 times improvement by using the accelerator, with the extra benefit of also freeing the CPU to be used for other tasks.

IV. Conclusion

In this paper we have evaluated the computational capabilities of several embedded COTS multicore and GPU devices, using a parallel implementation of an open source image processing application, part of the ESA's OBPMark benchmarking suite [3] [4]. We described in detail the application software, as well as the processing performance we obtained on multicore devices using OpenMP, and on the GPU using CUDA and OpenCL.

Our results indicate that both multicore CPUs and GPUs can provide a very high processing rate. However, GPUs can provide significantly higher performance and their benefit is even higher when the image size is increasing, which makes them ideal candidates for on board image processing. From the devices we have evaluated, the best multicore and GPU performance was achieved by the NVIDIA Orin in the 30W power mode. If we constrain the design to 15W we also see that the best multicore performance is achieved by the NVIDIA Orin but for GPU performance the NVIDIA Xavier NX achieves the top numbers, with the GPU being $8\times$ faster. Also is worth mention the case of the OCE platform, that manages to get a significant increase in performance compared to its multicore performance. Overall, we conclude that embedded GPUs can provide an order of magnitude higher performance than the corresponding multicores for space image processing tasks.

Acknowledgements

The authors would like to thank Frontgrade Gaisler for providing access to GR740, and Marc Sole and Jannis Wolf from Barcelona Supercomputing Center, for their assistance in obtaining the results on the device.

This work was supported by ESA through the 4000136514/21/NL/GLC/my co-funded PhD activity "Mixed Software/Hardware-based Fault-tolerance Techniques for Complex COTS System-on-Chip in Radiation Environments" and the GPU4S (GPU for Space) ESA-funded project.

Moreover, it was supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 772773) and European Community's Horizon Europe programme under the METASAT project (grant agreement 101082622). In addition, it was partially supported by the Spanish Ministry of Economy and Competitiveness under grants PID2019-107255GB-C21 and IJC-2020-045931-I (Spanish State Research Agency / Agencia Española de Investigación (AEI) / http://dx.doi.org/10.13039/501100011033) and by the Department of Research and Universities of the Government of Catalonia with a grant to the CAOS Research Group (Code: 2021 SGR 00637).

References

[1] I. Rodriguez, L. Kosmidis, O. Notebaert, F. J. Cazorla, and D. Steenari, "An On-board Algorithm Implementation on an Embedded GPU: A Space Case Study," in *Design Automation and Test in Europe Conference (DATE)*, 2020.

[2] L. Kosmidis, I. Rodriguez, A. Jover-Alvarez, S. Alcaide, J. Lachaize, O. Notebaert, A. Certain, and D. Steenari, "GPU4S: Major Project Outcomes, Lessons Learnt and Way Forward," in *Design Automation and Test in Europe Conference (DATE)*, 2021.

[3] D. Steenari, L. Kosmidis, I. Rodriguez-Ferrandez, A. Jover-Alvarez, and K. Förster, in *2nd European Workshop on On-Board Data Processing (OBDP2021)*, 2021, https://doi.org/10.5281/zenodo.5638577.

[4] ESA, "OBPMark GitHub Repository," http://obpmark.org.

[5] M. Harwit, "The Herschel Mission," in *Advances in Space Research*, vol. 34, no. 3, 2004, pp. 568–572.

[6] Nvidia, *Jetson Power Estimator*, jul 2023. [Online]. Available: https://jetson-tools.nvidia.com/powerestimator/userguide/

[7] I. Rodríguez Ferrandez, "An On-board Algorithm Implementation on an Embedded GPU: A Space Case Study," Master's thesis, Universitat Politècnica de Catalunya, 2021, https://upcommons.upc.edu/handle/2117/344892.