RELIABLE MACHINE LEARNING ACCELERATION FOR FUTURE SPACE PROCESSORS AND FPGAS: LEON, NOEL-V AND TASTE

Marc Solé[1,2], **Jannis Wolf**[1,3], **and Leonidas Kosmidis**[2,1]

[1] *Universitat Politècnica de Catalunya (UPC), Barcelona, Spain*
[2] *Barcelona Supercomputing Center (BSC), Barcelona, Spain*
[3] *Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Erlangen, Germany*

ABSTRACT

In this paper we present the design and implementation of two hardware designs for the acceleration of AI processing in space. First we present a SIMD module extension for Cobham Gaisler's LEON3 and NOEL-V space processors with support for fast AI operations, which achieves up to 4× performance improvement for commonly used ML operations compared to a minimal LEON3 configuration. Next, we present a Binary Neural Network (BNN) FPGA accelerator which leverages ESA's TASTE model-based framework to facilitate communication with the host space processor. Our simulation results show an estimated performance improvement of 32× compared to LEON3.

Key words: AI, ML, SWAR, LEON3, NOEL-V, BNN, TASTE.

1. INTRODUCTION

In recent years there has been an increasing interest in artificial intelligence (AI) and machine learning (ML) in space, specifically for on-board processing [1]. For example, current missions from both ESA and NASA include AI processing for autonomy and efficiency. In particular, NASA's latest Mars Rover, Perseverance, uses AI for navigation, however the low processing power of existing space processors permits only offline navigation planning and only over small distances and at low speed. For this reason, COTS accelerators such as Intel Movidius are explored for use in space systems, such as in ESA's Φ-Sat-1 technology demonstration mission, where AI is used for detecting clouds in satellite earth observation images, in order to save bandwidth from transmitting them. However, due to the low radiation tolerance of COTS systems in general, these solutions cannot be currently used beyond low-earth orbit (LEO) or for long term missions such as institutional ones. Moreover, COTS software stacks used in such accelerators, frequently depend on non-space qualified and non-real-time operating systems such as Linux, which creates another challenge to their adoption.

Therefore, ML acceleration features are needed in already qualified space processors and FPGAs, so that they comply with space requirements and to reduce their qualification time and cost. In this paper, we describe our on-going work on two such solutions, one for increasing the AI performance of space processors using a low-cost, short vector unit co-designed for common machine learning operations, and another one implementing a state-of-the-art low-cost binarised neural networks (BNN) on FPGAs, using ESA's TASTE framework for a reliable software stack. Both solutions are implemented in VHDL and are open-source [2][3] . Our early experimental results show that both proposals can provide a significant boost to the on-board machine learning processing capabilities of existing space systems.

In the following sections we provide the current design details and implementation status of these two hardware proposals as well as preliminary results of our experimental evaluation. Section 2 describes the architectural design of our low-cost vector unit, while Section 3 describes the architecture of the binarised-network accelerator. Section 4 evaluates each of the proposals and Section 5 presents the conclusion and our plans for future work.

2. LOW-COST SHORT VECTOR SUPPORT FOR LEON3 AND NOEL-V FOR ML

2.1. Architectural Design

Our work provides a low-cost hardware unit to speed-up AI applications through the use of short vector operations and special instructions which improve the performance when performing machine learning operations. The design has been driven by analyzing the most common operations in machine learning [4].

The module is portable and has been designed for the dual-licensed LEON3 [5] and NOEL-V [6] processors targeting the space domain designed by Cobham Gaisler.

Similar to the processors, the module is written in VHDL and available as open source.

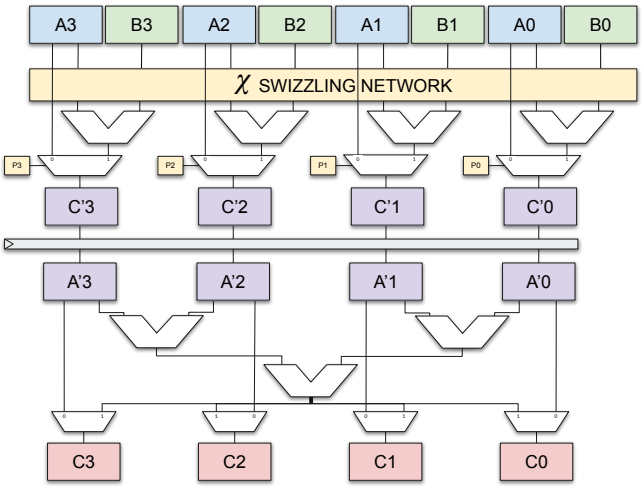

Figure 1. Outline of the SIMD module.

Despite the increased performance boost that traditional vector processing units offer, they have a significant hardware overhead, occupying large portions of the precious die area of embedded processors, which for the older processing technologies used for fabricating space chips are even higher. The main reason for this overhead is the large additional vector register files as well as the large vector width required by their functional units, which are frequently operating in floating point format. As such, they occupy an additional pipeline in the design.

In order to minimize the hardware overhead of our implementation, our design follows the next principles. First, it has been recently shown in the literature that 8-bit integer instructions are enough for machine learning [7], while there is a trend for even smaller sizes to the extreme of a single bit in the case of binarized neural networks, which we explore with our second contribution presented in Section 3. Therefore in our module, we only support integer and bitwise operations of 8-bit components, with and without saturation logic.

The fact that we operate on integer values, allows us to use the existing integer register file, avoiding extra hardware overhead and limiting our SIMD processing to 4 8-bit values which fit within a register. This independent parallel processing within a processor register is known in the literature as SIMD within a register - SWAR [8]. The original use of the term SWAR [8] was coined to refer to a generic, portable software programming model that could be implemented on top of the multimedia/short vector instructions of commodity microprocessors such as Intel's MMX/SSE, PowerPC's Altivec or SPARC v9's VIC extensions, but nowadays it is used with a broader meaning which includes all (mainly short) vector architectures whose instructions operate over multiple elements than can fit in a CPU register, either a vector or a scalar one.

Recently, SWAR extensions for the floating point instructions of the LEON2FT space processor have been explored within an ESA-funded activity [9]. In that work, short vector instructions were introduced for accelerating software defined radio (SDR)/digital signal processing (DSP) operations frequently found Global Navigation Satellite System (GNSS) receiver software such as correlation, demodulation and table lookups used in sine and cosine computations.

Inspired by [9], we apply a similar methodology focused on instructions frequently found in Machine Learning processing algorithms, and we extend this concept further, making some key architectural decisions which differentiate our proposal and offer additional benefits. The fact that we are reusing the existing integer register file enables us to utilise the existing processor instructions for loading, setting and performing some modifications to these registers. Therefore, this minimizes the additional SIMD instructions which need to be added to the processor. Moreover, this choice allows us to reuse the integer pipeline, requiring minimal hardware overhead and design changes. Our short vector module consists of two pipeline stages as shown in Figure 1, specifically tailored for the acceleration of dot product and matrix multiplication operations, which are dominating machine learning applications, as well as other operations found in neural network layers.

The first one operates in parallel with the execution stage of the integer pipeline, where the 32-bit integer Arithmetic Logic Unit (ALU) of the LEON3/NOEL-V is located, since only one of them can be used at the same time. The second pipeline stage of our module, implements reduction operations (maximum, minimum, sum or XOR) among the 4 8-bit components and it is bypassed when it is not needed, incurring no penalty.

In order to integrate the module we had to add support for new instructions in the processor. Each instruction encodes the operation code for each stage and the source and destination registers. In case there is no need to do an operation in one stage, it is bypassed by setting the corresponding operation code to 0. Although it was designed to work with both LEON3 and NOEL-V, currently the module has been implemented only in the LEON3 processor. In this implementation, we have reused the unused opcode space of Sparc V8 ISA, in order to preserve backwards compatibility with existing binaries. The NOEL-V implementation is currently work in progress and we are using the opcodes dedicated to custom ISA extensions.

The introduced SIMD instructions include also immediate versions. Since the available instruction space was limited, we encoded some commonly used values for each operation, such as powers of two, and the powers of two plus or minus one, similar to operations found in embedded GPU ISAs[10]. This is a key difference with [9], which allows to reduce register pressure and increase the instruction throughput, while accelerating numerous operations.

Finally in order to further improve the performance for certain operations, we added also the following features which were not used by [9] but were inspired by embedded GPUs and other vector extensions in microprocessors:

1) Masking: The mask, or predication vector, is a bit-vector which controls whether the result of each 8-bit component is written in the destination register.

2) Swizzling: The swizzling vector allows reordering the components of both input operands, as well as their duplication or masking. It is implemented with a series of multiplexers, shown for clarity as "swizzling network" in Figure 1.

Both bit-vectors are set using the write instruction to the new special register `%scr` (SIMD Control Register).

Our design is open source and can be accessed at [2]

2.2. Programming

We are currently working towards adding compiler support for our SIMD unit. For the moment, we only have added assembly support within the binutils of LEON3, which is provided as a part of Gaisler's LEON Bare-C Cross Compilation System (BCC) [11]. This allows to program the target algorithms in C using inline assembly for the SIMD operations, using the gnu assembler syntax. Listing in Figure 2 shows an example of this for a trivial example of vector addition with saturation.

Given the fact that our module operates on existing integer registers, with inline assembly we can allocate vector variables to specific registers (lines 5-7) and use them for normal operations in C (lines 10, 12, 16) or with our SIMD instructions using the explicit inline assembly instruction (line 14). Conceptually, this is similar to programming with vector intrinsics found in any short vector architecture eg. ARM NEON, so we are currently working on the addition of such intrinsics in the compiler. In the future, we plan to add further support in the compiler, to allow the exploitation of the SIMD architecture by "regular" GPU-like vector code as the one shown in the comments in the Figure 2 or even automatic SIMD vectorisation from scalar C code.

```
1  unsigned char weights[32*32];
2  unsigned char next_layer[32*32];
3
4  /* Allocate short vectors to specific registers*/
5  register unsigned int a asm("%g4");
6  register unsigned int b asm("%g5");
7  register unsigned int result asm("%g6");
8
9  /* initialise all a components to 0, ie a.xyzw=0 */
10 a = 0;
11 /* b.xyzw = weights[0].xyzw */
12 b = *((unsigned int*) &weights[0]);
13 /* result.xyzw = a.xyzw + b.xyzw */
14 asm("add_ %g4, %g5, %g6");
15 /* next_layer[0].xyzw = result.xyzw */
16 *((unsigned int*) &next_layer[0])=result;
```

Figure 2. Example of SIMD programming in C with inline assembly

3. BNN ACCELERATOR

3.1. Architectural Design

Depending on the type and the size of the actual neural network and on other mission requirements, an FPGA accelerator might be a preferred option. Moreover, recently Binary Neural Networks (BNNs) have been proposed [12], which reduce significantly the memory and computational resources required for inference.

While FPGA support for this type of neural networks already exists or can be implemented in various FPGA toolkits such as Xilinx's FINN [13] framework, Xilinx's Versal AI core [14], Microchip's VectorBlox [15] and others including high-level synthesis solutions, there can be certain settings in high criticality space systems which require full control or ownership of the hardware and software stack of the accelerator, such as integration with real-time operating systems (RTOSes), portability across FPGA vendors and other requirements.

In order to satisfy this need, ESA frequently relies on model-based engineering approaches. In particular, ESA has developed the open source TASTE model-based engineering framework [16][17], which supports both software and hardware code generation, while it supports solutions from multiple vendors relevant to the spacec domain.

In this section, we present an implementation of a BNN accelerator for FPGA systems which is appropriate for use in critical space systems, since its software stack and hardware communication implementation with the host CPU is correct-by-construction, leveraging TASTE.

We use TASTE in order to automatically generate the communication between the host space CPU processor, LEON 3, and our accelerator, based on an Abstract Syntax Notation One (ASN.1) [18] specification. In the ASN.1 specification, we define the amount of data we want to be transferred to the accelerator and their types, which is input data over which we want to perform the inference. Moreover, we describe the data which are sent back to the host CPU to indicate the inference results, which in our case is the short bit-vector of the last BNN layer used for the computation. Then the host processor translates this bit-vector to useful information, eg. to determine the class of the inferred object in case of image recognition.

The TASTE framework generates the host CPU code required for the communication based on the provided specification, which is in our case C code for bare metal execution. However, TASTE supports also code generation for Ada and its high-integrity Spark subset, as well

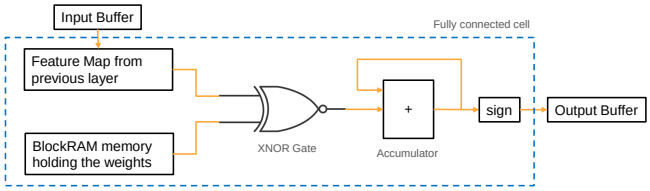

Figure 3. Functioning core of the accelerator architecture: fully connected cell.

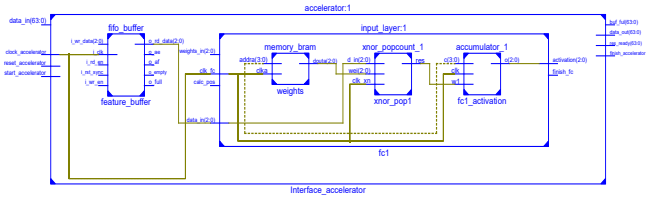

Figure 4. Sample accelerator schematic showing only one fully connected cell.

as integration with the RTEMS real-time operating system, including its SMP variant. Moreover, it generates the hardware implementation of the specified communication protocol in VHDL as well the project configuration targeting specific FPGAs, allowing a large variety of devices from different vendors. As a consequence, we can guarantee that these parts are free of errors, and therefore we can only focus on the implementation and verification of the BNN accelerator functionality.

As opposed to the input data which are streamed to the accelerator, the BNN weights are stored in the FPGA in block ram and they are reused for all inference operations over the different input data. In BNN networks, each activation and weights have two possible values, 1 or -1, which can reduce the memory consumption up to 8-64 times compared to conventional neural networks implemented with integer operations. Since only 1 bit is used for encoding these values, -1 is represented by the value 0. The more important benefit of BNNs is that instead of the expensive Multiply-and-Accumulate (MAC) operation used in the most expensive parts of the conventional neural networks, the implementation of fully connected layers, this is replaced by an exclusive nor (XNOR) operation and bit-count. In a similar way, other operations found in conventional NNs are replaced by bit-wise operations in BNNs.

In our accelerator, we currently implement fully connected networks using interconnected fully connected cells. The architecture of each cell is shown in Figure 3. From an input buffer the feature map gets loaded and flows through an xnor gate together with the weights loaded from the BlockRAM. An accumulator adds up the values until the full feature vector passed the xnor gate. The sign function calculates the outcome as follows: $result = 2 * p - n$ where p is the number of set bits and n is the length of the feature vector. If the result is positive, the activation of the neuron is 1, if its negative it is 0, respectively.

The RTL schematic of parts of the accelerator are shown in Figure 4. The fully connected layer receives the data from the FIFO module from the accelerator or the previous layers, as well as the weights from the memory modules. The fully connected layer as well as the memory is clocked and connected with an address counter. In our current implementation the loading of the weights takes twice as many cycles as the number of neurons in the accelerator layers. After the calculation an accumulator module sends the activations to the next layer. We can further optimize the design by using more BRAM modules for storing and loading the weights in parallel, however, the current design can provide enough performance as we show in the next section, so this is left as future work. The full code of our accelerator is released as open source [3].

4. EVALUATION

4.1. Vector Unit

Hardware Overhead: We synthesized the smallest possible LEON 3 processor configuration from GRLIB version GPL 2020.4-b4261 (LEON3-MIN), integrated with our module, using Vivado 2020.1 for the Artix 7 (part xc7a100tcsg324-2) FPGA. Table 1 shows the synthesis results of our module, including both the SIMD module as well as the changes in the control path for the implementation of the new instructions.

Table 1. Hardware Cost

Resource	Absolute Value	% of the design
LUTs	1869	25 %
FF	168	5.9 %

We can see that our modifications account for 25% of the resources of the resulting integrated LEON 3 design. This is significantly smaller than the typical overhead of vector units as well as compared to similar state of the art vector designs for embedded systems [19]. Moreover, considering that we are using the smallest LEON 3 configuration, with larger LEON 3 designs, our relative overhead is going to be even smaller. Our modifications reduce the frequency of the processor from 100 MHz to 72MHz. The biggest contributor to the critical path is the multiplication operation, whose impact is amplified by the cascade of ALUs in the second pipeline stage of the module, so in the future we are going to improve this part. Without the multipliers, we could achieve a 90MHz frequency. However, even with this frequency reduction, our module provides an important performance improvement in ML processing as we show next.

Performance: We collected preliminary results with the most commonly used operation in machine learning, matrix multiplication, which is used for implementing both

convolutions as well as fully-connected layers in neural networks. We used different sizes and we compared our SIMD implementation with two versions of the matrix multiplication running on an unmodified LEON 3 as shown in Table 2. The baseline `Leon3 char` uses 8 bit values and simulates saturation using comparisons, and `Leon3 int` uses 32 bits values so saturation is not required. We introduced the SIMD instructions for our module with inline assembly in C as shown in Figure 2 and we make sure that the results of our SIMD version are identical to the ones produced by the baseline versions.

Table 2 shows the cycle count for each test, for the different matrix sizes. As we can see, there is a considerable reduction in the number of cycles when using the SIMD module, up to 5×. Interestingly, performing the same operations on LEON3 using 8-bit arithmetic does not present a meaningful improvement over the 32-bit version. This shows that LEON3 cannot benefit simply by reducing the size of neural network operations which is a common approach in tinyML systems, but it needs the assistance of a SIMD unit or accelerator to provide high AI performance.

Given that our modifications reduce the CPU frequency, we provide the actual speed-up in Table 3 for better interpretation of the results. We notice that even in this case, we have reached up to 3.81× speed-up in the 8x8 matrices. In the other scenarios the speed-up is also considerable proving the utility of the designed SIMD module in machine learning applications. Moreover, the results show that the overall performance benefit is larger than the hardware and frequency overhead.

In addition to this simple evaluation, in order to have a more space relevant evaluation of our design, we ported the CIFAR-10 complex inference application from the GPU4S Bench open source benchmarking suite [20][21][22]. For inference over 32×32 image sizes, the speedup we get over the integer implementation is 4.13×. The application consists of multiple neural network layers such as fully connected, convolutional, max pooling etc. We notice that the overall benefit we can get by processing a series of neural network layers is larger than the improvements we noticed for a single fully connected layer, since some of the networks can benefit much more from our SIMD additions.

As a rough indication of the speedup offered by our SIMD implementation to the baseline LEON3, we can consider that on this benchmark, the use of GPU in an NVIDIA Xavier platform offers 3.61× speedup over a parallel 4-core version of the application running on its high-end ARMv8 compliant Denver CPU [23].

4.2. BNN Accelerator

We have implemented our accelerator using the latest version of the TASTE framework and Xilinx's latest ISE version (14.7), which is the only FPGA flow from Xilinx which is currently integrated with it.

Table 2. Matrix multiplication cycle comparison.

Matrix Size	***4x4***	***8x8***	***16x16***	***32x32***
LEON3 int	2160	12246	86580	657071
LEON3 char	1968	11526	83179	641903
SIMD module	638	2329	20342	140619

Table 3. SIMD module Speed-up computed based on the physical execution time of each configuration.

Matrix Size	***4x4***	***8x8***	***16x16***	***32x32***
Speedup over LEON3 int	2.45	3.81	3.08	3.39
Speedup over LEON3 char	2.24	3.59	2.96	3.31

We have trained the BNNs for our test cases in Python using TensorFlow and Keras in order to obtain the trained binary weights which we use in our accelerator. Currently we have not performed the integration of all the system on chip in the FPGA (i.e. including the LEON 3 CPU). Instead, we have validated our design in simulation with ISE, using a testbench for testing the accelerator from end-to-end inference: transfer data to the accelerator, inference and transfer data back from the accelerator. We validate that the inference results using our accelerator match exactly the CPU implementation.

For the functional validation of the accelerator, we have used two test cases: one for the classification of the Iris data set [24] obtained from [25], which consists of 150 4-dimensional feature vectors who group into three classes, and one for the MNIST dataset [26]. The Irist dataset has good characteristics for testing neural network performance with respect to prediction. In terms of computational cost, a 4D feature vector does not require very expensive calculations, but it is a good way to verify our implementation. Similarly, the MNIST dataset is small enough to allow validation by simulation.

For the performance evaluation, we perform synthesis for the Xilinx Spartan3 FPGA and according to ISE's timing estimation, without place and route our accelerator can achieve a maximum frequency of 114.943MHz. We compare our performance against a simulated version of the LEON 3 using Cobham Gaisler's TSIM version 3.0.2, which models a LEON 3 configuration clocked at 50MHz and equipped with a 4 KiB cache, with 16 bytes per cache line.

Due to the memory size and simulated instruction limitations of the evaluation version of TSIM, we cannot use our validation case studies, which despite their small size exceed these limits. For this reason, we model only a 512×512 fully connected layer, which fits in the processor cache, therefore the achieved performance is only limited by the computational capabilities of the CPU. When compiling with -O2, the LEON 3 can perform the inference task in 4.8 Million cycles, while our accelerator needs 65 thousands cycles for the same computation.

Given the fact that our accelerator operates with double frequency compared to the CPU, this results in an impressive speedup of 147×.

If we consider the data transfer of these values to the accelerator, the performance benefit is going to be smaller. For example, in the extreme case of such a small network that we are considering which doesn't make much computational demands and in the case that the input data fit in the processor cache, the benefit is about 32×. In case of larger BNNs, which consist of multiple connected layers and therefore they will require a higher ratio of computations compared to the transferred input data, the benefit is expected to be higher.

However, he have to highlight that both these results are obtained with simulation so they have to be taken with a grain of salt. Once we will perform the integration of the LEON 3 and our accelerator in an FPGA, we will be able to obtain performance results with higher confidence. Moreover, we will be able to compare their area and frequency trade-offs in a realistic way, using synthesis results after placement and routing. Last but not least, we will use a relevant case study from a safety-critical domain similar to [20] or [22].

5. CONCLUSION AND FUTURE WORK

In this paper we introduced two different work-in progress approaches aiming to the improvement of artificial intelligence computation in space computing. The SIMD module presents a portable design that with minimal hardware cost can speed-up the AI processing capabilities of existing space processors such as the LEON3 or the NOEL-V up 4×. All of this while maintaining the baseline architecture providing backwards compatibility and allowing the re-utilization of the existing software.

The BNN FPGA accelerator design confirmed once again the benefits of the TASTE framework, which allows developers to concentrate on the design of their hardware without being concerned about the software stack for communication with the accelerator. This accelerator has demonstrated incredible results with simulation, showing the advantages of BNNs. Although we take them with care, we are optimistic about the actual performance benefit that our accelerator can have on a real FPGA.

Both our designs are interesting approaches for space AI acceleration and they are complementary to each other, so they can also work well together. We leave the evaluation of a system using both designs simultaneously as future work .

For the SIMD module, in addition to the optimization of the multiplication operation to further reduce our hardware and timing cost, our next steps include compiler support for our new instructions, to simplify the code generation and maximize the optimization in the code. With these modifications the performance of the SIMD instructions may increase further. Moreover, this will allow us to perform a more extensive evaluation, with standard machine learning workloads targeting the space domain [22]. Finally the integration of our module in the NOEL-V processor will also be a goal to reach in the short-term. This would bring even more opportunities for improvement in the AI computation in space processors.

On the other hand, regarding the BNN accelerator, we are planning to create a VHDL code generator for the implementation of the AI accelerator directly from Python to the TASTE framework. In addition, we will integrate our solution with a deep learning library such as PyTorch or TensorFlow, in order to allow interoperability with existing widely used machine learning frameworks.

ACKNOWLEDGMENTS

This work is partially supported by ESA under the GPU4S (GPU for Space) project (ITT AO/1-9010/17/NL/AF), by the Spanish Ministry of Economy and Competitiveness (MINECO) under grants PID2019-107255GB and FJCI-2017-34095. It received also support by the European Commission's Horizon 2020 programme under the UP2DATE project (grant agreement 871465), the HiPEAC Network of Excellence, the Xilinx University Program (XUP) and XUP Board Partner Red Pitaya.

REFERENCES

[1] Jan-Gerd Meß, Frank Dannemann, and Fabian Greif. Techniques of Artificial Intelligence for Space Applications - A Survey. In *European Workshop on On-Board Data Processing (OBDP)*, 2019.

[2] Marc Solé Bonet. GRLIB AI extension. `https://gitlab.bsc.es/msolebon/grlib-ai-extension`, 2021.

[3] Jannis Wolf. FPGA BNN Accelerator. `http://www.github.com/JannisWolf/fpga_bnn_accelerator`, 2021.

[4] Daofu Liu, Tianshi Chen, Shaoli Liu, Jinhong Zhou, Shengyuan Zhou, Olivier Teman, Xiaobing Feng, Xuehai Zhou, and Yunji Chen. PuDianNao: A Polyvalent Machine Learning Accelerator. In *ASPLOS*, 2015.

[5] Cobham Gaisler. LEON3 Processor. https://www.gaisler.com/index.php/products/processors/leon3. (Accessed June 07, 2021).

[6] Cobham Gaisler. NOEL-V Processor. https://www.gaisler.com/index.php/products/processors/noel-v. (Accessed June 07, 2021).

[7] N. P. Jouppi et al. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *ISCA*, 2017.

[8] Randall James Fisher. *General-purpose SIMD Within a Register: Parallel Processing on Consumer Microprocessors*. PhD thesis, Purdue University, 2003.

[9] Martin Daněk. ESA IP Core Extensions for LEON2: daiFPU and SWAR. In *ESA TEC-ED & TEC-SW Final Presentation Days*, May 2020.

[10] Matina Maria Trompouki and Leonidas Kosmidis. Towards general purpose computations on low-end mobile gpus. In *DATE*, 2016.

[11] Cobham Gaisler. LEON Bare-C Cross Compilation System (BCC). https://www.gaisler.com/index.php/products/operating-systems/bcc. (Accessed June 07, 2021).

[12] H. Qin, R. Gong, X. Liu, X. Bai, J. Song, and N. Sebe. Binary Neural Networks: A Survey. *Pattern Recognition, Special Issue: Explainable Deep Learning for Efficient and Robust Pattern Recognition*, 105, 2020.

[13] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. Finn: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '17, page 65–74, New York, NY, USA, 2017. Association for Computing Machinery.

[14] Brian Gaide, Dinesh Gaitonde, Chirag Ravishankar, and Trevor Bauer. Xilinx adaptive compute acceleration platform: Versaltm architecture. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '19, page 84–93, New York, NY, USA, 2019. Association for Computing Machinery.

[15] Joseph James Edwards. *Real-time Computer Vision in Software using Custom Vector Overlays*. PhD thesis, University of British Columbia, 2018.

[16] European Space Agency (ESA). TASTE. https://essr.esa.int/project/taste, 2021.

[17] Maxime Perrotin, Eric Conquet, Julien Delange, André Schiele, and Thanassis Tsiodras. TASTE: A Real-Time Software Engineering Tool-Chain Overview, Status, and Future. In *SDL 2011: Integrating System and Software Modeling, LNCS, Vol. 7083*, pages 26–37, 01 2011.

[18] International Standards Organization (ISO). Information technology - abstract syntax notation one (asn.1): Specification of basic notation. (ISO/IEC 8824-1:2015), 2015.

[19] M. Johns and T. J. Kazmierski. A Minimal RISC-V Vector Processor for Embedded Systems. In *2020 Forum for Specification and Design Languages (FDL)*, 2020.

[20] I. Rodriguez et al. GPU4S Bench: Design and Implementation of an Open GPU Benchmarking Suite for Space On-board Processing. Technical Report UPC-DAC-RR-CAP-2019-1, Universitat Politecnica de Catalunya. `https://www.ac.upc.edu/app/research-reports/public/html/research_center_index-CAP-2019,en.html`.

[21] Leonidas Kosmidis, Iván Rodriguez, Alvaro Jover-Alvarez, Sergi Alcaide, Jérôme Lachaize, Olivier Notebaert, Antoine Certain, and David Steenari. GPU4S (GPUs for Space): Are we there yet? In *European Workshop on On-Board Data Processing (OBDP)*, 2021.

[22] David Steenari, Leonidas Kosmidis, Ivan Rodriquez, Alvaro Jover, and Kyra Förster. OBPMark (On-Board Processing Benchmarks) - Open Source Computational Performance Benchmarks for Space Applications. In *European Workshop on On-Board Data Processing (OBDP)*, 2021.

[23] Alvaro Jover-Alvarez, Alejandro J. Calderón, Iván Rodriguez, Leonidas Kosmidis, Kazi Asifuzzaman, Patrick Uven, Kim Grüttner, Tomaso Poggi, and Irune Agirre. The UP2DATE Baseline Research Platforms. In *Proceedings of the 2021 Design, Automation and Test in Europe Conference and Exhibition, DATE 2021*, 2021.

[24] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, 1936.

[25] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[26] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 1998.