On-board decision making in space with deep neural networks and risc-v vector processors

Di Mascio, Stefano; Menicucci, Alessandra; Gill, Eberhard; Furano, Gianluca; Monteleone, Claudio

DOI
10.2514/1.I010916

Publication date
2021

Document Version
Final published version

Published in
Journal of Aerospace Information Systems

JOURNAL OF AEROSPACE INFORMATION SYSTEMS
Vol. 18, No. 8, August 2021

On-Board Decision Making in Space with Deep Neural Networks and RISC-V Vector Processors

Stefano Di Mascio,* Alessandra Menicucci,† and Eberhard Gill‡
Delft University of Technology, 2629 HS Delft, The Netherlands
and
Gianluca Furano§ and Claudio Monteleone¶
European Space Agency, 2200 AG Noordwijk, The Netherlands

https://doi.org/10.2514/1.I010916

The use of deep neural networks (DNNs) in terrestrial applications went from niche to widespread in a few years, thanks to relatively inexpensive hardware for both training and inference, and large datasets available. The applicability of this paradigm to space systems, where both large datasets and inexpensive hardware are not readily available, is more difficult and thus still rare. This paper analyzes the impact of DNNs on the system-level capabilities of space systems in terms of on-board decision making (OBDM) and identifies the specific criticalities of deploying DNNs on satellites. The workload of DNNs for on-board image and telemetry analysis is analyzed, and the results are used to drive the preliminary design of a RISC-V vector processor to be employed as a generic platform to enable energy-efficient OBDM for both payload and platform applications. The design of the memory subsystem is carried out in detail to allow full exploitation of the computational resources in typically resource-constrained space systems.

I. Introduction

THE success of deep neural networks (DNNs) for terrestrial applications has been mainly due to the availability of large datasets (i.e., rise of "big data") and the availability of relatively inexpensive hardware that can run learning and inference in reasonable timescales, for instance, graphics processing units (GPUs) [1]. The space industry looks at this phenomenon with interest, although the availability of large datasets for space applications is limited and the hardware employed in space applications lags behind in terms of performance compared with its commercial counterpart.

One of the main issues in terms of hardware faced by the space industry is that it is not possible to reuse in a straightforward way the hardware platforms employed in terrestrial applications, given the specific constraints of satellite data systems especially in terms of robustness to ionizing radiation [2]. For instance, the GPU tested in [3] is reported to fail during an irradiation with high-energy proton beam roughly every 43 s. The main reason behind this very low mean time to failure (MTTF) is that GPUs are much larger (e.g., 2.2 billion transistors,** which corresponds to roughly 550 MGE if we assume four transistors per GE††) than single-core, single-issue processors (890 kGE for the one in [5]) typically employed in space. As a matter of fact, the failure rate of a processor (given a certain technology and environment) is proportional to its area (i.e., the number of sequential elements when considering only upsets in sequential elements) [6]. Therefore, even employing a Rad Hard By Design (RHBD) technology, a GPU is expected to fail almost three orders of magnitude more often than a state-of-the-art space processor.

A larger soft error vulnerability is not the only reason why simple microarchitectures with low parallelism are still the vast majority of processors employed in space. As a matter of fact, most of the tasks executed by processors in space data systems are non-compute-intensive workloads; i.e., they perform a low number of operations per byte read from and written to memory. The reason is that they are mainly employed for nondemanding control and housekeeping operations, whereas on-board data processing typically is not an attractive solution, because it can be executed in most cases on ground with much less expensive machines for a given computational need (unless it allows for improved capabilities of the satellite, e.g., data encryption or compression).

It is still matter of discussion whether artificial intelligence (AI) is of actual interest in space applications and whether it will be feasible to deploy it systematically on-board satellites in the next 10–15 years. Therefore, the first goal of this paper is to carry out an analysis of the impact and requirements of DNNs (the most successful form of AI in terrestrial applications) in space systems. To try to meet these requirements, the space industry is following three main approaches:

1) Work is being done to map efficiently DNNs on resource-constrained state-of-the-art space processors [7], accepting a consistent loss of performance compared with DNNs in high-performance processors for terrestrial applications. This approach can exploit synergies with the trend in Internet of Things (IoT) of implementing DNNs in low-power and resource constrained processors [8].

2) High-performance proprietary commercial-off-the-shelf (COTS) processors employed in terrestrial applications are being proposed [9]. Although they can achieve higher-order magnitude performance compared with state-of-the-art space processors [10], they come with a large "cost of ownership" to avoid losses in terms of dependability [11], and possible restrictions on its usage and knowledge of internal behavior [4].

3) Field programmable gate arrays (FPGAs) allow the design of a customized hardware accelerator, typically connected to either a hard or soft processor through an interconnect [10]. The accelerator can be either handcrafted in hardware description languages (HDLs) or autogenerated from software, after profiling to identify the most computational intensive functions. In [12] it is shown that Vivado HLS with enabled optimizations (i.e., pipelining and concurrent

Received 14 October 2020; revision received 7 April 2021; accepted for publication 1 May 2021; published online Open Access 24 June 2021.

*Ph.D. Candidate, Faculty of Aerospace Engineering, Space Systems Engineering; s.dimascio@tudelft.nl (Corresponding Author).

†Assistant Professor, Faculty of Aerospace Engineering, Space Systems Engineering; a.menicucci@tudelft.nl.

‡Professor, Faculty of Aerospace Engineering, Space Systems Engineering; e.k.a.gill@tudelft.nl.

§On-Board Computer Engineer, Microelectronics and Data Systems Division, European Space Technology Centre, Keplerlaan 1; gianluca.furano@esa.int.

¶On-Board Computer Engineer, Microelectronics and Data Systems Division, European Space Technology Centre, Keplerlaan 1; claudio.monteleone@esa.int.

**https://www.techpowerup.com/gpu-specs/radeon-e9173-pcie.c3031.

††A gate equivalent (GE) is a technology-independent unit of measure of the area of a design (normalized to a reference 2-input NAND gate) [4].

execution of operations) achieves a 6.23× speed-up for a small convolutional neural network (CNN) and 9× for a larger CNN on a Zynq compared with the software implementation on its hard processor (which can be considered roughly equivalent to space processors). However, in space applications this approach cannot be typically employed for application-specific integrated circuits (ASICs), given the niche-sized market available. Furthermore, in [13] it is noted that only 25% on average is spent waiting on accelerator computations, with the rest of the time taken up by data transfers (34%) and processor computations (42%).

The second goal of this paper is to show that a fourth approach can be followed, as performances of space processors can be substantially improved with data-level parallelism (DLP) to achieve performance of the same order of magnitude of high-performance terrestrial processors for DNNs. To do this, we develop in more detail the study of the RISC-V processors needed to enable on-board decision making (OBDM) carried out in [4], focusing on the preliminary design of a RISC-V vector processor specifically for space applications. This preliminary design is intended to serve as a baseline for future works, during the Very High Speed Integrated Circuit Hardware Description Language (VHDL) implementation of a RISC-V vector processor for space applications based on the NOEL-V platform (developed by Cobham Gaisler) [14].

RISC-V is an instruction set architecture (ISA) that is rapidly growing in popularity in both terrestrial and space applications [4]. Its main characteristics are simplicity, openness (being a free and open standard allows open-source implementations), and modularity (i.e., composed of a base ISA and many optional ISA extensions). Among the many ISA extensions defined in the standard, the RISC-V Vector Extension (RVVE) is being proposed to provide general support for data-parallel execution [15].

The paper starts by analyzing the benefits that DNNs can provide at the system level and the feasibility of the deep learning approach for space applications (Sec. II). Then, an analysis of the software workloads required for DNNs is carried out in Sec. III. The information collected is then used to define a suitable hardware platform (Secs. IV and V). To account for both computational and memory constraints, separate discussions are carried out for the microarchitecture of the processing core (Sec. IV) and its memory subsystem (Sec. V). Finally, Sec. VI concludes with a summary of the main findings and several recommendations to systematically enable OBDM with RISC-V vector processors in the medium-term.

II. Impact at System Level

The focus of the space industry in recent years shifted from large geostationary orbit (GEO) satellites to small (< 500 kg) low-Earth-orbit (LEO) satellites (especially CubeSats) [16,17].

While GEO satellites can continuously communicate with the ground station, LEO satellites can only communicate with the ground station periodically, sometimes with large periods between contacts [18]. In this way, the satellite may enter an unsafe state and the ground operator in the worst case can only intervene hours later.

However, there is a trend of launching LEO satellites in constellations and mega constellations [19], with the possibility of mitigating the risk of failure of a single satellite and replace them if they fail (as they are much cheaper than large GEO satellites). There is therefore a tradeoff to be made between dependability of a single satellite, its cost, and number of spare satellites.

Furthermore, space systems are inherently constrained in terms of power available (e.g., only a limited surface is available to collect power). Limited power implies that the data rate of the downlink given a certain target bit error rate (BER) is also limited, as the data rate is proportional to the power employed during the transmission [20].

Therefore, small satellites in LEO pose new challenges both in terms of amount of data that can be transmitted to the ground and in terms of dependability. In the following two subsections we will show how OBDM can help mitigating these shortcomings of LEO satellites. In Sec. II.C the feasibility of applying DNNs to these problems is investigated.

A. Downlink Efficiency

In [20] it is shown that, assuming a transmission power of 1 W for a CubeSat in LEO, a maximum data rate of 512 kbps can be obtained with a ultra-high-frequency (UHF) downlink (BER < 10^{-5}). On the other hand, in [20], an image from a simple VGA camera with 640×480 pixels is 900 KiB, and a cube from a hyperspectral sensor with 32 bands and 1024×1024 pixels is instead 32 MiB. In LEO a potential access duration to the ground station can be 5 min every orbit (around 90 min in [21]), and in this time span, only around 21 VGA images or roughly half a cube of 32 bands can be transmitted to the ground. Even considering the relatively large 6U CubeSat described in [22], the power budget for the downlink limits the data rate to 14 Mbps, whereas its spectrometer (with five spectral bands) generates 255 Mbps. Without considering data compression, during a single ground station pass, only up to 31.5 s worth of imaging data can be downlinked, which is only around 0.6% of the data that can be collected during a typical LEO orbit. Although it is not realistic to assume that the spectrometer operates continuously during the mission, this shows that there is a mismatch between the capability of a small satellite in LEO to generate data and its capability to transmit data to the ground.

1. *Benefits of Data Removal and Compression*

Given the tight power budgets and the expensive hardware required for on-board data processing, data processing is typically executed on ground. For instance, noise filtering can be executed on ground with cheaper hardware. On the other hand, sometimes on-board data processing provides an advantage over on-ground data processing in terms of satellite performance. For instance, data compression is already deployed in many missions (e.g., in [22] a 2:1 compression is employed) because it mitigates the bottleneck of the downlink. The efficiency of the downlink can be increased even further, removing useless data instead of sending it to the ground (i.e., data removal [23]). For instance, in the Landsat datasets [24], the average cloud cover in an archived scene is 34%, with 38% of the scenes containing less than 10% cloud cover. Therefore, selecting only images with less than 10% of cloud cover results in an average of 2.63× data reduction. Combining data removal with a 2:1 compression, the amount of useful data sent increases by 5.26× compared with a system without on-board data processing.

2. *Cost of Required Hardware*

When DNNs and other data processing algorithms are to be deployed on data produced by instruments, a payload processor is required to process the data. Although memories with long retention time and low power dissipation (e.g., flash memories) can be employed for mass memories, faster memories are required to act as main memory of the payload processors. Typically dynamic random-access memory (DRAM) arrays are chosen, ranging from single data rate (SDR) to double data rate 2 (DDR2) to double data rate 3 (DDR3), depending on the radiation resilience/performance tradeoff required [25]. From the datasheet [26] of the 1 Gb DDR2 DRAM tested in [25], a peak power consumption of around 0.5 W can be taken as an estimation of power consumption, and 1 W for the most powerful version of the vector processor in [27] running a peak-performance application. Assuming a requirement of 1 GiB of main memory, we consider 5 W as the cost in terms of power P_P of applying data reduction and compression. As a comparison, 1U CubeSats and 3U CubeSats in [20] generate, respectively, 1–2 W and 5–6 W, whereas the 6U CubeSat in [22] generates around 20 W.

Assuming a common amount of power allocated for the transmission and data processing subsystems (P_{TP}), we can estimate the amount of useful data transmitted per station contact D_C when data are not processed as $D_C = (P_{TP}/RR) * k$, where k is a constant (dependent on the transmission subsystem, receiver, propagation, and required BER) [20] and RR is the optimal removal rate, i.e., the ratio between useful data and data produced by the payload. When only useful data are selected and a data compression of CR:1 is applied, the amount of useful data transmitted is instead $D_C = CR * (P_{TP} - P_P) * k$. The ratio R between

the amount of useful data transmitted in the two cases is then $R = RR * CR * (P_{TP} - P_P)/P_{TP}$), which for $P_{TP} \gg P_P$ tends to its maximum, i.e., $RR * CR$. This means that data removal is more effective for larger satellites, which have more power available for transmission and processing.

To give an idea of what the effect of a more power-efficient solution would be, in Fig. 1 the ratio R depending on the power budget P_{TP} for two different values of power spent for data processing P_P (5 and 2.5 W) is shown. While the fraction of the maximum ratio achieved for a certain P_{TP} is independent from $RR * CR$, it depends on P_P. As a matter of fact, it takes a larger P_{TP} to achieve a certain fraction of the maximum improvement possible when P_P is increased.

B. Virtual Operator

In [21] it is assumed that a LEO satellite has an orbit duration of 90 min and that there is a contact with the ground station either 5 min every orbit (6% of the orbital period) or every 5 orbits (1%). In a similar scenario, the idea of an on-board virtual operator monitoring the status of the satellite and taking autonomous decisions when no communication with the ground operator is possible becomes of great interest. The on-board virtual operator can, for instance, enable autonomous failure detection (and forecasting) and autonomous safe mode management. For instance, DNNs can be employed to predict the telemetry of the next orbit given the previous one (or more) [28]. This can be used to help diagnose anomalous behaviors before the next contact with the ground station [28].

1. *Benefits of an On-Board Virtual Operator*

Assuming a constant failure rate λ (typical of soft errors [6]), the reliability of the spacecraft after the end of the ith contact and before $(i+1)$th contact can be expressed as $R(t) = R(t_i)e^{-\lambda(t-t_i)}$, where t_i is the time instant of the end of the ith contact. Assuming a ground operator capable of handling safely the failures of the spacecraft and a satellite not capable of handling safely failures in autonomy, the safety (we define safety as the percentage of time a satellite is working in nominal conditions or it is in a safe state because of a detected failure) $S(t)$ is 100% during contact and is $S(t) = R(t)$ when the satellite is not in contact. When considering an on-board virtual operator, a percentage of failures is detected with a certain detection factor (DF), then $S(t) = R(t_i)e^{-(1-DF)\lambda(t-t_i)}$ when not in contact. Figure 2 shows an example for $DF = 0.9$, a λ of 10e-4 failures/min (approximately one failure per week), and two different time periods between contacts (90 and 450 min). The average safety increases from 99.60 to 99.96% in the first case and from 97.83 to 99.78% in the second. Although this is a simple model and some on-board failure detection capabilities are possible without DNNs, it shows that improving the on-board capabilities of a satellites can help LEO satellites to achieve typical requirements for dependable systems (at least 99.9%).

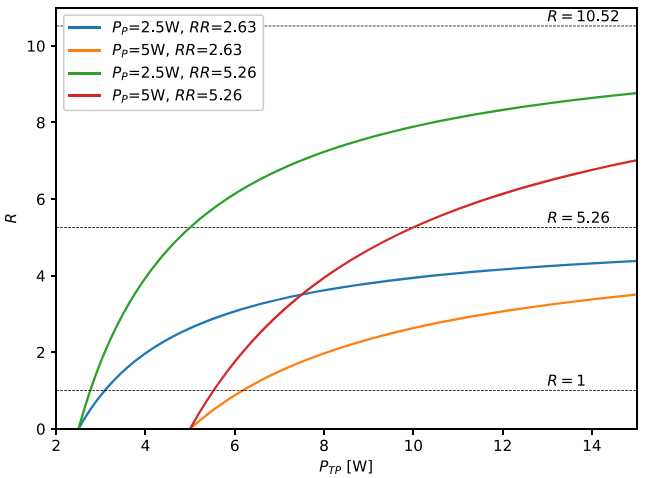

Fig. 1 Ratio R between useful data transmitted with and without data removal against P_{TP} allocated for the transmission subsystem and data removal for different P_P and RR. In all cases 2:1 compression is assumed.

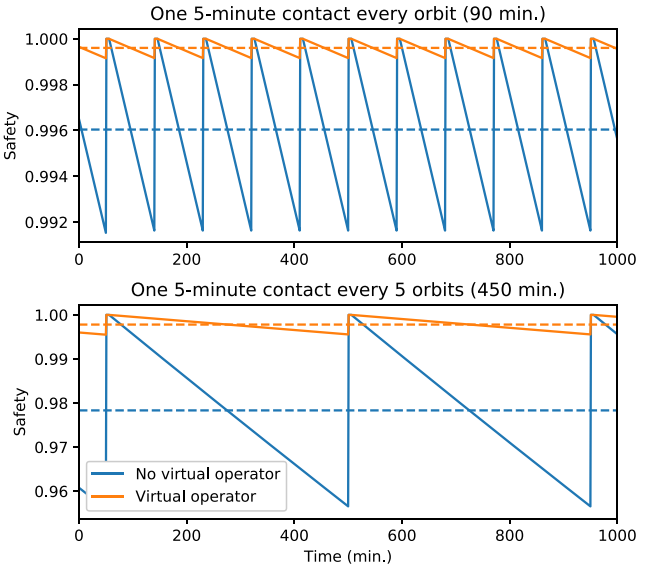

Fig. 2 Increase of safety for a virtual operator with $DF = 0.9$ for contact of 5 min with ground station every 90 min and every 450 min. Dashed lines represent average values.

2. *Cost of the Required Hardware*

The most attractive solution to deploy telemetry analysis and forecasting is to use an enhanced version of a typical on-board computer (OBC). As a matter of fact, the requirements for this kind of applications are less stringent compared with DNNs for image analysis. In [28], telemetry forecasting was implemented with a 64 Mb DRAM on a single core reaching 661 predictions per second, meaning that all the parameters of the satellite telemetry in [28] (13,216 in total) can be predicted in around 20 s. It is enough to execute this computation once per orbit to predict the telemetry of the following orbit, taking only 0.4% of an orbit period. Furthermore, enhancing a general purpose in processor with minimal vector facilities (i.e., two lanes) increases power consumption of the processor from 52 mW [5] to 138 mW [27] (+65%).

C. Feasibility of the DNN Approach

DNNs proved to be the best approach in classification and prediction when large datasets are available for training, reaching accuracies close to or slightly above human level [29]. When the training set is not large enough, other machine learning approaches or human-defined DSP algorithms may instead achieve better results. The degradation of DNN accuracy for small datasets is shown, for instance, in [30], where CNNs are trained for multiclass classification with different datasets sizes. It is shown that, for a three-class classification problem, using 5000 images per class achieves 97% accuracy in average, bringing the training set down to 1000 images per class lowers accuracy to 74% in average. When the problem becomes more complicated (i.e., larger number of classes), even using 45,000 images in total achieves an average accuracy below 94% (nine classes) [30].

One of the most popular datasets for terrestrial applications is ImageNet.[‡‡] It is a large image dataset typically used to assess the effectiveness of a certain neural network architecture for image classification, containing RGB images of 256×256 pixels for a total of 1000 classes [29]. There are 1.3M training images (ranging from 732 to 1300 per class) and 100,000 test images [29]. Large reference datasets available to the public are much less common for space applications. One of the most popular is the public Landsat 8 dataset,[§§] which provides hyperspectral images composed of 11 bands

[‡‡]http://www.image-net.org.

[§§]https://www.usgs.gov/land-resources/nli/landsat/landsat-8.

ranging from ultra-blue to thermal infrared. A large number of land cover classification solutions were developed on subsets of the Landsat datasets [31].

Setting up a reference, standardized dataset is instead more difficult for more specific applications, especially those involving inner behavior of the satellite, like telemetry forecasting or anomaly detection. As a matter of fact, design parameters like orbits, observed signals, and nominal values change from mission to mission. Furthermore, major prime contractors have very restrictive data policies concerning open access to telemetry data. However, some datasets of telemetries are available to the public, like those of the GOCE mission.[¶] Even in this case, it is difficult to pinpoint anomalies, as information about them is typically not shared by the mission teams with the public. However, public datasets can help to study the feasibility of telemetry forecasting, as done in [28].

In the future, the idea of deploying telemetry analysis on-board will have to face the problem of relying on ad-hoc datasets for specific applications to use for training and testing. One option is to wait for a certain period of nominal operation of a satellite and use the past telemetry to train the network on ground and then uplink the trained network in software. When the telemetry forecasting is to be deployed on a constellation composed of replicas of the same satellite, more statistics for larger datasets are available. As reported in [19], existing and planned constellations comprise hundreds to thousands satellites (e.g., 4200 for the planned constellation from Samsung), thus making DNNs potentially very effective also for mission-specific parameters.

III. Workloads Analysis

The run time of compute-intensive workloads composed of a certain amount of floating point calculations is typically expressed in terms of number of floating point operations (FLOPs) per second (FLOP/s) or number of FLOPs per clock cycle (FLOP/CC).[***] The number of FLOP/CC that can be achieved by a certain hardware platform has an upper bound defined by the number of functional units and the amount of operations these units can perform simultaneously. We call this upper bound maximum theoretical performance per clock cycle (MTP_{CC}). MTP_{CC} is independent of any other microarchitectural feature, like instruction-level parallelism (ILP), speculation, and caching. However, it is not possible to achieve $\#\text{FLOP/CC} \approx \text{MTP}_{\text{CC}}$ for every workload, as data are to be fetched from memory, and in some cases this cannot be done fast enough to keep the functional units busy all the time. To visualize whether a workload can achieve the MTP_{CC} (compute-bounded workloads) or the performances are bound from the memory bandwidth (memory-bounded workloads), the roofline model was introduced in [32]. According to this model, the fraction of MTP_{CC} that can be achieved by a workload on a certain platform depends on the operational intensity (OI) of the workload, which is

$$\text{OI} = \frac{\#\text{FLOP}}{\text{MT}} \tag{1}$$

where MT is the memory traffic composed of the read traffic RT plus the write traffic WT. For each hardware platform there is an OI^* for which workloads are memory-bounded if $\text{OI} < \text{OI}^*$ (therefore the performances are limited to $\#\text{FLOP/CC} = \text{BW} * \text{OI}$, where BW is the bandwidth of the memory) and compute-bounded if $\text{OI} > \text{OI}^*$ (where achieving the MTP_{CC} is actually possible with microarchitecture and software optimizations). Although based on several assumptions, for instance, that it is possible to overlap memory transfers and computations [33]), the roofline model is a successful tool to benchmark processors in an application-independent way, mainly focusing on the performance of popular kernels (e.g., [27]).

[¶]https://goce-ds.eo.esa.int/oads/access/collection/GOCE_Telemetry/.

***Normalizing by frequency is a common procedure to obtain technology-independent metrics that measure the effectiveness of a certain microarchitecture.

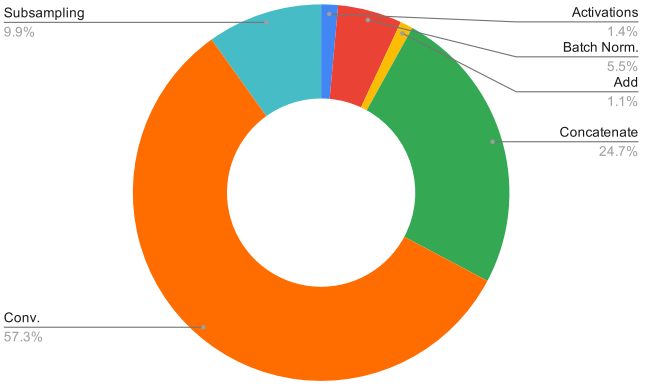

Fig. 3 Breakdown of the execution time for an inference of CloudNet.

A. CloudNet

As a case study of DNN for image analysis, we will consider the public code[†††] of CloudNet [34]. It is a fully convolutional network (FCN) [35] for cloud detection; i.e., its output is a mask of the same size of the input image indicating the pixels covered with clouds. The use of an FCN instead of a CNN helps in mapping efficiently the DNN in resource-constrained hardware, as it is possible to work on patches of a large image without the need of working on the entire image. The fraction of bits covered in clouds can then be averaged on the ∼400 patches. In the case of CloudNet four spectral bands of the large images of Landsat 8 (e.g., 7621×7791 pixels) are divided in nonoverlapping patches of 384×384 pixels, which are then downsampled to 192×192 pixels.

Analyzing the model in Keras,[‡‡‡] we find that CloudNet contains 38 two-dimensional convolutional layers (of which 5 are transposed), 15 addition layers, 31 batch normalization layers, 45 standalone activation layers, and 53 concatenate layers. To give an idea of the contribution of each of these layers, we profiled the execution of the model on a quad-core Intel i7-6600U. The breakdown of the execution type for each type of layer is shown in Fig. 3, and considerations on each of them are carried out in the remainder of this section. Furthermore, running a single inference per time requires a peak memory of 836.65 MiB. This value is compatible with values found in literature for other DNNs, typically ranging from 645 MiB to 1.49 GiB [36].

1. *Convolutional Layers*

As shown in Fig. 4, applying a convolutional layer with N kernels, each of dimensions $C \times J \times K$, kernels to an input of dimensions $C \times W \times H$ generates an output of N matrices, each of dimensions $U \times V$ [37], with U and V depending on the stride S and padding P of the convolutional layer with the equation (an analogous relationship holds replacing W, J, and U with, respectively, H, K, and V) [38]:

$$U = \lfloor (W - J + 2P)/S \rfloor + 1 \tag{2}$$

As straightforward software implementations of convolutions achieve low performance, performances are typically improved unrolling the convolutions into matrix–matrix multiplications [39]. In this case, the number of FLOPs for each layer is estimated as $\#\text{FLOP} = 2UVNCJK$, given that there are UVN output elements and for each of them CJK multiplications and accumulations are required. The read traffic is then $RT = 4(NCJK + UVCJK)$ and the write traffic is $WT = 4NUV$. In Table 1 we show the size of the unroll of the convolution for only the first 15 layers (for sake of brevity) of the network. Some observations can be made:

1) OIs are large (in the order of tens of FLOP/B), except for convolutions with $K = 1$ for which OI can go down to 1.60 FLOP/B.

[†††]https://github.com/SorourMo/Cloud-Net-A-semantic-segmentation-CNN-for-cloud-detection.

[‡‡‡]https://keras.io.

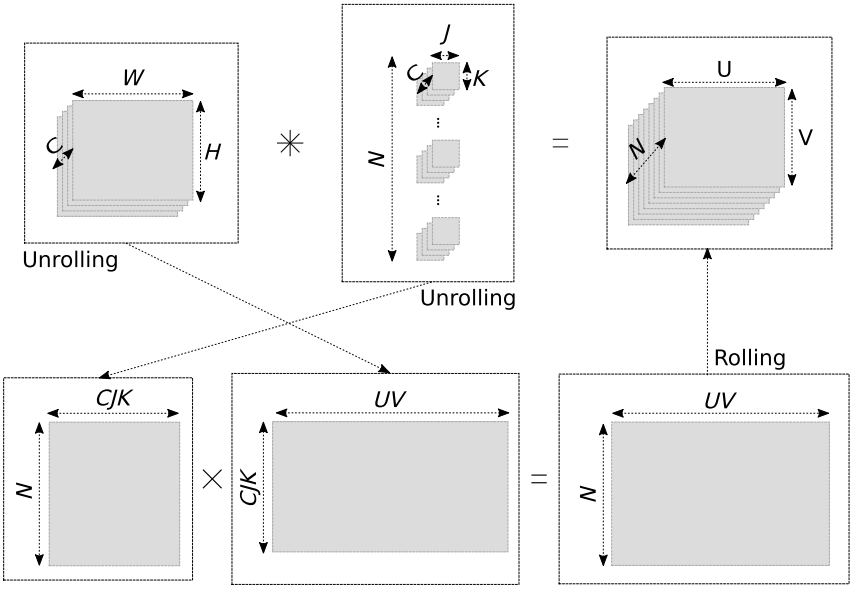

Fig. 4 Unrolling of a convolution (similarly to Ref. [37]).

Table 1 Workload characterization for the first 15 layers of CloudNet [34]

Convolution	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$W = H$	192	192	192	192	96	96	96	48	48	48	24	24	24	12	12
C	4	16	16	32	32	32	64	64	64	128	128	128	256	256	512
$K = J$	3	3	3	1	3	3	1	3	3	1	3	3	1	3	3
N	16	32	16	32	64	32	64	128	64	128	256	128	256	512	512
$U = V$	192	192	192	192	96	96	96	48	48	48	24	24	24	12	12
RT [MiB]	5.06	20.27	20.26	40.54	10.20	10.16	20.39	5.34	5.20	10.69	3.66	3.09	7.31	5.77	11.53
WT [MiB]	2.25	4.50	2.25	4.50	2.25	1.13	2.25	1.13	0.56	1.13	0.56	0.28	0.56	0.28	0.28
MT [MiB]	7.31	24.77	22.51	45.04	12.45	11.29	22.64	6.47	5.77	11.81	4.22	3.38	7.88	6.05	11.81
#MFLOP	42.5	340	170	75.5	340	170	75.5	340	170	75.5	340	170	75.5	340	679
OI [FLOP/B]	5.54	13.08	7.20	1.60	26.03	14.36	3.18	50.09	28.10	6.10	76.80	48.00	9.14	53.58	54.86

2) Even if OI is large and therefore the workloads can be assumed to be compute-bounded, the absolute amount of memory traffic is very high (3–45 MiB per layer). These values require a dedicated design of the memory subsystem compared with processors for non-compute-intensive workloads, which will be carried out in Sec. V.

3) The memory traffic is for a large majority composed by reads (92.83% in average).

Further performance enhancements can be obtained by mapping the matrix–matrix multiplication with optimized libraries. In [39] it is shown that using basic linear algebra subroutines (BLAS) instead of coding the unrolled version from scratch produces a speed-up ranging from 2.43× to 3× depending on the architecture and on the input size. Using BLAS subroutines, matrix–matrix multiplications are mapped to the *SGEMM* subroutine,[§§§] which (in its nontransposed form) implements the following algorithm:

$$A2 \leftarrow \alpha A0 \times A1 + \beta A2 \tag{3}$$

where $A0$, $A1$, and $A2$ are matrices of, respectively, size $n_1 \times n_2$, $n_2 \times n_3$, and $n_1 \times n_3$, and α and β are scalars. Assuming $\alpha = \beta = 1$ (as in the case of convolutions) and a square matrix at the output ($n_1 = n_3$), SGEMM has

$$\mathrm{OI} = \frac{n_1^2(1 + 2n_2)}{8(n_1^2 + n_1 * n_2)}$$

Assuming that $2n_2 \gg 1$, $\mathrm{OI} \approx (n_1 * n_2/4(n_1 + n_2))$,which given a certain memory traffic (i.e., $n_1 + n_2 = \text{const}$) is maximized for $n_1 = n_2$, reaching $\mathrm{OI} \approx n_1/8$. As OI is proportional to the size of the output matrix, SGEMM will eventually achieve the peak performance for a large enough matrix on a given hardware platform. For this reason, the SGEMM efficiency

$$E_{\mathrm{SGEMM}} = \frac{\mathrm{FLOP/CC}}{\mathrm{MTP_{CC}}}$$

(i.e., the fraction of time the functional units of the processor are busy when executing SGEMM) is typically given as a measure of attainable performance on a certain hardware platform [40]. When caching levels are present, increasing the size of the matrix multiplications to increase OI will eventually cause a drop in performance, as the operands will not fit anymore in the cache level responsible of peak performance and reads from lower levels (even main memory) are required during the matrix multiplication, breaking the assumption of the roofline model that memory traffic and computation overlap. This issue is analyzed in Sec. V.

2. *Concatenate and Addition*

Given that CloudNet is very deep (38 convolutional layers), it requires specific solutions in its architecture to mitigate the vanishing gradient problem [41]. The designers of CloudNet handled this problem using skip connections, and addition and concatenation

[§§§]Analogous subroutines are defined for different data types, and the first letter represents the data type. For instance, SGEMM is for single precision (SP), DGEMM is for double precision (DP), and IGEMM is for integers. In this paper, data will be assumed to be SP unless specified otherwise; therefore SGEMM will be used.

layers [34]. As can be seen in Fig. 3, although the impact of addition layers on the execution time is negligible (1.1%), concatenation layers take a considerable part of the execution time (24.7%). Furthermore, concatenate operations contain no FLOPs and consist mainly of memory transfers; therefore they cannot be sped up with increased computation capabilities. These considerations suggest avoiding architectures concatenation and using skip connection between layers with same dimensions (where concatenations are not needed), as done in [41].

3. Batch Normalization

Batch normalization layers are employed to speed up training and increase accuracy of DNNs [42]. This type of layer also acts as a regularizer, keeping the magnitude of the parameters low and avoiding overfitting [42,43]. When using batch normalization during inference, each element x_i of the activation vector x from the previous layers is normalized according to

$$\hat{x}_i = \frac{x_i - E[x_i]}{\sqrt{Var[x_i] + \epsilon}} \tag{4}$$

where $E[x_i]$ and $\mathrm{Var}[x_i]$ are, respectively, the expected value and the variance of x_i (obtained from cumulative statistics collected during training), and ϵ is a small constant to ensure convergence. The number of operations required for an activation vector of n elements is $2n$, and the number of elements to be read from and written to memory is $3n$; therefore $\mathrm{OI} = 1/6\,\mathrm{FLOP/B}$. This value shows that this layer is typically very memory-bounded, taking up a nonnegligible part of the total execution time of CloudNet (5.5%).

4. Activation Layers

Analyzing CloudNet in Keras, we found that a total of 84 activation layers are present. However, only 48 are nonlinear (i.e., 36 activation layers are composed of pass-through functions that do not have any computational impact), of which 47 are rectified linear units (ReLUs) and only 1 is a sigmoid (at the last layer). ReLU functions have low computational impact, as it is enough to set to 0 all the negative values [44]. Sigmoids (and hyperbolic tangents) are computationally more expensive, as in principle they require the calculation of a nonlinear math function. A typical approach to implement them is to use a lookup table [44] or a piecewise linear approximation [28].

5. Subsampling Layers

To reduce the number of computations and make features more robust [45], typically convolutional layers are followed by subsampling (or pooling) layers. In CloudNet all the subsampling layers are implemented with *max pooling*; i.e., the maximum value of a window is selected as the output value. This is common in state-of-the-art DNNs [37], although sometimes different approaches are employed, such as *average pooling* (the output is the average of the values in the window), a mix of max and average pooling, and stochastic pooling) [45]. Pooling can be either implemented as a nested for-loop over each window, or split into operations in one axis and then in the other (which usually provides better performance [44]).

B. Other Layers in DNNs for Image Analysis

When DNNs are employed for classification, the expected output is typically a vector containing the probability of classification for a certain object. In these cases, the last layers of the DNN after the convolutional layers are composed of fully connected (FC) layers to make a decision based on the information contained in groups of pixels. This type of network is usually called CNN. FC layers can be seen as convolutional layers where there is no sharing of coefficients, i.e., $J = K = W = H$ [37]. This implies that the output is a vector of size N, the number of operations is $\#\mathrm{FLOPs} = 2NCHW$, the memory traffic is $\mathrm{MT} = 4[CHW(N+1) + N]$, and

$$\mathrm{OI} = \frac{1}{2[1 + 1/N + 1/(CHW)]}$$

Therefore, OI reaches its maximum (0.5 FLOP/B) for very large CHW and N. To give an idea of how FC layers compare against convolutional layers, we compared the memory traffic and OI for the convolutional layers in Table 1 to FC layers with same C, W, H, and N. The MT of the FC layers ranges between 1.31× and 18.36× compared with the respective convolutional layer, whereas the OI is 3.3× to 154.2× smaller. The high MT associated with FC layers is confirmed by [37], although a trend can be noticed: for early CNNs with few convolutional layers (e.g., AlexNet with five convolutional layers and three FC layers) the percentage of parameters in the FC layers is very high (for AlexNet 96.07%), whereas state-of-the-art deeper CNNs (typically achieving higher accuracy) like ResNet have many convolutional layers (for ResNet the number of convolutional layers ranges from 53 to 155 and typically only one FC layer is present) and have a much lower percentage of parameters in the FC layers (ranging, respectively, from 8.04 to 3.42%).

Furthermore, the performance for FC layers can be improved employing batching, i.e., processing more input features in parallel [37]. This technique is particularly effective in the case of FC layers, as it allows reuse of the large amount of parameters read from memory over several input features.[¶¶] When processing B input features in parallel, the number of operations is $\#\mathrm{FLOP} = 2NBCHW$, the memory traffic is $\mathrm{MT} = 4[CHW(N + B) + BN]$ and the operational intensity is

$$\mathrm{OI} = \frac{1}{2[1/B + 1/N + 1/(CHW)]}$$

This equation shows that the effectiveness of batching eventually saturates. For instance, for $C = 512$, $N = 512$ and $W = H = 12$ without batching $\mathrm{OI} = 0.5$ FLOP/B. For small batching, i.e., $1/B \gg [1/N + 1/(CHW)]$, batching causes an almost linear increase of OI and $\mathrm{OI} \approx B/2$ (e.g., $\mathrm{OI} = 3.93$ FLOP/B for $B = 8$). The effectiveness of batching saturates for larger B until for very large batching an upper bound of

$$\mathrm{OI}_{\max} = \frac{1}{2[1/N + 1/(CHW)]}$$

is reached (in this example around $254\,\mathrm{FLOP}/B$). A relatively high value of B may be required to achieve an OI in the order of the tens (e.g., 15.05 FLOP/B for $B = 32$). Furthermore, batching introduces an extra latency, as to process a frame in the worst case $B - 1$ successive input feature maps have to be calculated. This effect of batching can be an issue in real-time applications and is further analyzed in Sec. III.C.

Despite the described criticalities of FC layers, they typically have limited impact on the execution time of CNNs. For instance, in [46] the breakdown of the execution time for inference according to the different type of layers is reported to be 90.7% for convolution layers, 9.15% for subsampling layers, 0.03% for ReLU activation layers, and 0.11% for FC layers. The breakdown of the number of layers is instead 25% convolution layers, 20% subsampling layers, 40% activation layers, and 15% FC layers.

C. DNNs for Telemetry Forecasting

Recurrent neural networks (RNNs) are typically employed in time series analysis like speech recognition and natural language processing (NLP) [47–49], and they can be applied, for instance, to early failure detection or to predict the telemetry of the next orbit given the telemetry of previous orbits, as done in [28]. RNNs are composed by a cascade of units with internal feedback, where each unit requires the output of the previous one to be ready to calculate the next activation. Typically long short-term memory (LSTM) implementations are chosen to achieve higher accuracy, whereas gated recurrent unit (GRU) implementations provide lower accuracy with higher

[¶¶] Batching is instead not effective with convolutions, as the amount of parameters in a convolution is very small (e.g., $3 \times 3 \times 16$).

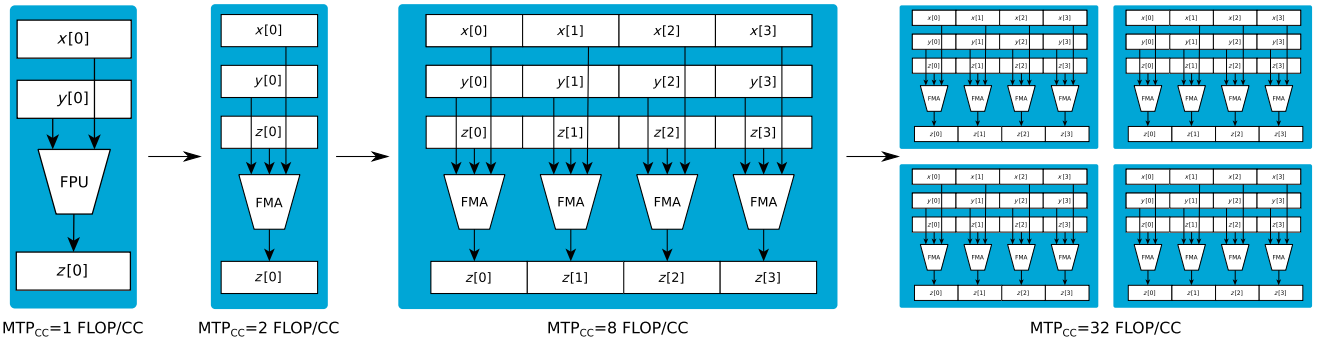

Fig. 5 Steps to increase the MTP$_{CC}$ of space processors in a power- and area-efficient way.

performance [28]. Furthermore, one or more FC layers are placed before the output [28].

LSTM layers are typically memory-bounded [50]. Similarly to [28], the linear part of the LSTM layer can be described as

$$s_t = W \cdot x_t + U \cdot h_{t-1} + b \tag{5}$$

where x_t, h_{t-1}, and s_t are column vectors, respectively, of length m, n, and n. W and U are, respectively, $[m \times n]$ and $[n \times n]$. Therefore the #FLOPs is $2(n^2 + n + nm)$, the MT seen by main memory is $4(n^2 + nm + 3n + m)$, and the OI is

$$\text{OI} = \frac{\#\text{FLOP}}{2(\#\text{FLOP} + 2n + m)} \tag{6}$$

with a maximum value of 0.5 FLOP/B for large matrices, i.e., $\#\text{FLOP} \gg 2n + m$. This low value can be increased with batching, as it turns matrix–vector multiplications into more computational intensive matrix–matrix multiplications (as B vectors are put together to create a matrix of dimensions $[n \times B]$). In this case,

$$\text{OI} = \frac{\#\text{FLOP} * B}{2[\#\text{FLOP} + (3B - 1)n + m * B]} \tag{7}$$

This equation shows that the efficacy of batching in terms of increase of OI saturates as B grows, until the upper bound of $\text{OI}_{\max} = \#\text{FLOP}/(6n + 2m)$ is achieved. This upper bound is a relatively large value, for instance, 27.29 for $m = 27$ and $n = 60$ (typical values in [28]). However, OI cannot be increased arbitrarily by batching in real-time applications, as batching requires that all the inputs to the LSTM layers of the batch are ready. For instance, in [50] increasing batching from 16 to 64 increases performance to 2.41× the original value, whereas the time required to complete execution in more than 99% of the cases increases from 7.2 to 21.3 ms (2.95×).

IV. RISC-V Vector Processors

State-of-the-art processors for space applications typically execute instructions on two scalar operands [51]. Considering a single core, this type of platform has an MTP$_{CC}$ of 1 FLOP/CC.

The simplest way of increasing the MTP$_{CC}$ of future space processors is to introduce ISA extensions with instructions defining fused multiply-add ($z \leftarrow wx + y$) and fused multiply-accumulate ($z \leftarrow xy + z$) operations,**** achieving an MTP$_{CC}$ of 2 FLOP/CC (as shown in Fig. 5). This requires modifications to the floating point unit (FPU) and arithmetic-logic unit (ALU). However, the cost of these changes in the FPUs and ALUs is limited (as, for instance, the area of these units is dominated by the multiplier). The biggest cost is instead on the complexity of the register file, which is required to provide more operators to the functional units [52].

To increase the MTP$_{CC}$ even further, DLP is the most energy-efficient solution available [53]. As a matter of fact, large part of the power consumption of a general-purpose scalar processor is spent on fetching instructions. For instance, the breakdown of energy dissipation on a scalar processor executing IGEMM in [5] shows that the instruction cache dissipates 19.63% of the total energy, instruction fetch and decode stages 4.69%, and the virtual memory (comprising both instruction and data) 7.41%. A percentage of energy dissipation ranging between around 24 and 32% can therefore be attributed to instructions fetching and decoding. Data parallel processors reduce this fraction of power, defining instructions that operate on arrays of D elements instead of scalar elements. Figure 5c shows an example with $D = 4$, which (together with FMA operations) achieves MTP$_{CC}$ = 8 FLOP/CC. However, DLP is the least flexible form of parallelism [53], as it can only be applied to calculations that can be vectorized (i.e., expressed with instructions on vectors), e.g., matrix–matrix multiplications in convolutional layers. As a matter of fact, in [54], the speed-up found in the convolutional layers of a CNN using the data-parallel NEON extension over the baseline ARM ranges from 2.45× to 2.78×, with a decrease of energy consumption per convolutional layer ranging from 59.11 to 82.04%. The energy efficiency of the data-parallel solution (i.e., performance in terms of executed layers per amount of energy) is in this case then 5.98× to 15.50× the energy efficiency of the non-data-parallel baseline. When the effectiveness of DLP saturates for large D, the solution left to increase the MTP$_{CC}$ is to replicate the processing core. In Fig. 5d the core is replicated four times ($P = 4$), achieving an MTP$_{CC}$ of 32 FLOP/CC (together with FMA and $D = 4$). Going above four cores typically reduces the utilization of the functional units. For instance, in [55] it is shown that with eight cores it is possible to obtain for CNNs' performances ranging from 3.99× to 5.76× the performance of a single core. Similarly, with eight cores it is possible to reach 5.55× the performance of a single-core implementation of an LSTM RNN [28].

A. Data-Parallel Processors

When compute-intensive applications were to be addressed in the commercial market, computer architects resorted to packed single instruction multiple data (SIMD) ISAs with the Intel's MMX extensions (1996) for integers [56] and the SSE extensions (1999) for floats [57]. The success of ARM in high-end embedded applications made the SIMD NEON extension, first introduced in the ARMv7-A Cortex-A8 (2005) [58], very popular. Also PULP, one of the most popular sets of RISC-V cores, employs the RI5CY packed SIMD extension (2016) defined outside of the RISC-V standard [59].

Packed SIMD extensions are typically chosen by hardware designers because they can be applied to scalar processors without extensive modifications to the microarchitecture [60]. However, the end of Moore's law is leading computer architects to use more efficient ISA extensions, and ARM recently (2017) released its ARMv8-A Scalable Vector Extension (SVE) [61]. Although previous Fujitsu's supercomputers were based on SIMD extensions of SPARC, the Fujitsu A64FX is the first processor based on the ARMv8-A SVE, targeting supercomputer applications. It achieves 2.7 DP-TFLOPS (7 nm process), a DGEMM efficiency >90% [62] and it is composed by 48 computing cores, each achieving around 57 DP-GFLOPS [62].

****Both will be indicated with FMA, unless a distinction is to be done.

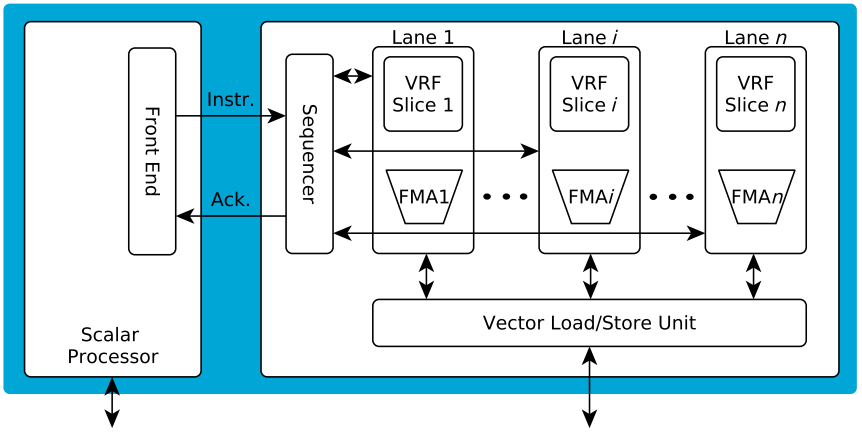

Fig. 6 Block diagram of a decoupled vector pipeline.

Vector extensions are already known to be more efficient than packed-SIMD, as they can be seen as more flexible versions of packed-SIMD thanks to their time-multiplexed and vector length-agnostic approach (the software is oblivious to the hardware vector length of a specific implementation and the same code executes using the largest parallelism available) [27,60,61]. In SIMD extensions instead, the data width of the operations is encoded in the instruction opcodes. When the architects of such ISAs wish to increase performance by widening the vectors, they must add a new set of instructions to process these vectors. For instance, Intel's newest AVX instructions are as long as 11 bytes [60]. Furthermore, application code compiled for previous versions cannot automatically leverage the widened vectors of new implementations. At the same time, code compiled for wider SIMD registers fails to execute on older machines as the new instructions are not known to older implementations. Furthermore, in SIMD extra code is needed to handle up to three fringe elements of stripe mine loops [60].

For these reasons, the proposal for packed-SIMD floating-point was dropped in favor of the V extension for large floating-point vector operations [15]. However, there was interest in packed-SIMD fixed-point operations for use in the integer registers of small RISC-V implementations. A task group is working to define the packed-SIMD P extension [15].

1. *RISC-V Vector Extension*

The RISC-V Vector Extension (RVVE) is similar to the ARMv8-A SVE and was heavily inspired by the Hwacha[††††] development [63]. Both RVVE and ARMv8-A SVE define a configurable vector unit with 32 vector registers (i.e., given a certain VRF size, the number of elements and size of elements can be configured with instructions) [15] and allow the same binary code to work efficiently across a variety of hardware implementations, varying in physical vector storage capacity and data path parallelism. Additionally, ARMv8-A SVE includes 16 scalable predicate registers (not defined in the baseline RVVE [64]) to optimize loops, using the predicate controlled loops vectorization style [61].

Although the RVVE is still in the process of being standardized, it plays such a crucial role in state-of-the-art applications that already several developments implementing the RVVE are described in literature. The two most notable examples are the Xuantie-910, a 12 nm RISC-V processor with 16 cores clocked up to 2.5 GHz with an out-of-order triple-issue 12-stage pipeline [65], and Ara, a RISC-V vector processor based on Ariane achieving up to 33 GFLOP/s and 41 GFLOP/J on 22 nm fully-depleted silicon-on-insulator (FD-SOI) technology. Furthermore, work is being done to support the RVVE in popular DNN frameworks like TensorFlow Lite [66].

[††††]The main difference with RISC-V Vector extension is that Hwacha fetches its own instructions, as there are two threads: a control thread running on the scalar core and a worker thread [60]. This can potentially lead to higher performance, but also higher complexity.

2. *Microarchitecture of Vector Processors*

There are two main approaches to design a vector processor. Vector processors for supercomputers, like the Fujitsu AF64X, typically have a joint scalar and vector pipeline with separated register files and execution units. The main disadvantage of this approach is that a vector load instruction stalls the pipeline also for scalar instructions, unless a superscalar pipeline with large ILP is employed (e.g., this is done in the Fujitsu AF64X with up to four ways). When the ILP is not high enough, using a decoupled vector pipeline, where the scalar pipeline pushes vector instructions into an instruction queue interfacing the vector pipeline, can mitigate this issue. The scalar pipeline can continue execution and the vector pipeline acknowledges completion of vector instructions and passes scalar results (when needed) to the scalar pipeline without passing through the bus. This approach is employed, for instance, for the Ara processor [27] and it is shown in Fig. 6. Another advantage of this approach is that it provides a more modular solution and a vector version of a RISC-V processor can be achieved with minimal modifications to the scalar design (i.e., the introduction of a front end).

The critical elements of a vector processor are shown in Fig. 6. The following subsections will focus on the vector register file (Sec. IV.B), and on the issues limiting scalability of performance (Sec. IV.C). Furthermore, Sec. IV.D provides insights on the soft error vulnerability of vector processors.

B. Vector Register File

Vector register files (VRFs) are typically more complex than register files (RFs), as they have in general more contention given FMA operations and masked execution[‡‡‡‡] [27]. When considering Ara, the worst case for contention for access to the VRF is the masked FMA (multiply-add) instruction, which reads four operands from four vector registers (one mask, two factors, and one addend) [27], executes the operation only if the mask has a certain value, and writes to a register the result of the operation. A straightforward solution to avoid contention in the VRF is therefore to employ a multiported static random-access memory (SRAM) with as many ports as needed, in this case four read ports and one write port (4R1W). However, multiported register files come with a large area overhead. In [53], the area of the VRF for the T0 vector processor according to the different number of ports employed is analyzed. As the T0 vector processor contains two arithmetic units and one multiplier per lane, to avoid contention it requires one read port and one write port for the multiplication, and two ports for read and one for write for each arithmetic unit (i.e., 5R3W). Different implementations in ASIC technology are proposed for the VRF, trading-off the number of banks and ports: one 5R3W bank of 256 elements (1×5R3W), two

[‡‡‡‡]RVVE provides for many instructions a field that specifies whether the instruction is to be executed or not according to the value of a bit in a specific vector register [64].

3R2W banks of 128 elements each (2×3R2W), and four 2R1W banks of 64 elements each (4×2R1W).

Data from [53] show that banking decreases the area occupied by the VRF by 31.7% when going from 1×5R3W to 2×3R2W. However, the efficacy of this technique saturates quickly, as going from 2×3R2W to 4×2R1W decreases the area only by 2.1%. This is due to the increase of overhead to handle the banks (storage cells compose 88.9% of the VRF for 1×5R3W, 83.1% for 2×3R2W, and only 41.7% for the 4×2R1W implementation).

Banking is also employed in Ara, where the VRF is composed of eight single-ported read-or-write banks (1RW). To help avoid contention, in Ara vectors are organized in SRAM banks with a shift of one element ("barber pole" shift) [27]. This is particular effective to avoid conflicts when the functional units fetch the first elements of two vectors [27]. However, this organization leaves some residual contention, which is addressed with a round-robin with two priority levels [27]. A way to completely solve bank contention is systolic execution. For instance, Hwacha uses four 1R1W (4×1R1W) dual port banks with stall-free systolic bank execution, capable to sustain n operands per cycle to the shared functional units after an initial n-cycle latency [63].

C. Scalability

Although existing RISC-V vector processors have good scalability in terms of peak performance and efficiency (as can be seen in Table 2), there are still criticalities to be addressed for small matrices and very high requirements of peak performance. The remainder of this subsection discusses how scalability influences frequency, efficiency, the effects of the issue rate on the achieved performance and the width of the interconnect.

1. *Frequency*

Most considerations in previous sections were based on the frequency-normalized value FLOP/CC, whereas a reduction of clock frequency decreases the peak performance in terms of FLOP/s (as $\#\text{FLOP}/s = f_{\text{CPU}} * \#\text{FLOP/CC}$) and therefore can decrease the efficiency of a platform with increased DLP.

In [27] Ara has been implemented in Global Foundries 22FDX process (FD-SOI). As can be seen in Table 2, the two-lane and four-lane versions of Ara achieve the same maximum nominal frequency. In both cases, the critical path is in the DP FMA FPU (1.2 GHz nominal, 0.92 GHz worst case), about 40 gate delays long. Another critical path (of the same length) is present in the combinational handshake between the Vector Load and Store Unit (VLSU) and operand queues in the lanes of the vector processor. When increasing the number of lanes, the second path becomes longer, and therefore the frequency is reduced (down to 1.04 GHz for 16 lanes). This is because the VLSU handles data to and from all the lanes simultaneously. Therefore, a larger number of lanes imply longer combinational paths. This shows that, in general, the scalability of the DLP in a vector processor is limited by the elements that act on all the lanes [27].

It should be noted that the maximum frequency of the scalar processor on the same technology is 1.7 GHz [5]. Therefore, the two-lane version already comes with a penalty of at least 30% compared with the scalar processor.

2. *Area and Energy Efficiency*

The increasing energy efficiency in Table 2 shows good scalability and suggests that the peak in energy efficiency may be obtained for an even larger number of lanes. On 22 nm FD-SOI, Ariane and Ara (depending on the number of lanes) consume between 138 (2 lanes) and 794 mW (16 lanes) at peak performance [27]. As energy efficiency depends on the ASIC technology employed, changing technology will provide different efficiency. Resorting to a 65 nm RHBD technology would decrease energy efficiency because of larger power consumption for a given clock frequency.

Area efficiency reaches a maximum for 8 lanes, as for 16 lanes the increase due to the decreased overhead of the scalar pipeline per vector lane is more than compensated by the greater complexity of the logic to handle the increased number of lanes. Therefore, area efficiency can be expected to be more critical than energy efficiency in vector processors. Ariane and Ara occupy together between 2228 and 10,735 kGE. In particular, the area of Ariane and Ara with four lanes is 3434 kGE. i.e., 4.28 times a single-core Ariane comprising level 1 (L1) caches. Therefore a four-lane vector processor has similar requirements in terms of die area compared with state-of-the-art quad-core processor for space [51].

Table 2 Scalability of Ara in terms of number of lanes (peak values in bold) for 22FDX process (FD-SOI) (data derived from [27])

	Number of lanes			
Performance metric	2	4	8	16
Max. frequency [normalized]	**1.00**	**1.00**	0.94	0.83
Max. FPU utilization [%]	**98.20**	98.00	97.22	97.36
Area efficiency [DP-kFLOP/s/GE]	2.20	2.85	**3.08**	3.02
Energy efficiency [DP-GFLOP/mJ]	35.58	37.84	39.91	**40.81**

3. *Small Matrices*

Along with the memory bound identified by the roofline model, the authors of Ara [27] show that the limited issue rate of instructions for a single-issue scalar pipeline limits the performance for matrices of sizes smaller than 256×256. Therefore, they suggest that the use of higher ILP and speculation in the scalar pipeline could improve performance for smaller matrices, where control operations (e.g., configuration of the lanes) have a larger overhead. Similarly to [27] for an $n \times n$ matrix multiplication with SP parameters, an upper bound due to the issue rate $\#\text{FLOP} = (16 * \text{OI}/\Delta_{\text{CC}_{\text{issue}}})$ can be found, and $\text{OI}^* = (\text{MTP}_{\text{CC}} * \Delta_{\text{CC}_{\text{issue}}}/16)$ due to the issue rate. This equation shows that doubling the issue rate (i.e., using a dual-issue microarchitecture) will halve the OI*. For instance, as an FMA instruction can be issued every five clock cycles (CCs) in Ara, the worst OI* is 5 FLOP/B (8 lanes version with $\text{MTP}_{\text{CC}} = 16$ FLOP/CC), whereas a dual-issue version lowers this value to 2.5 FLOP/B. As can be seen in Sec. V, these values are comparable with upper bounds due to memory bandwidth and therefore can have an impact on performance when they produce an higher OI* than memory bandwidth.

4. *Interconnect*

To increase the OI* due to the memory bandwidth, Ara uses a single $32 * N_L$-wide bus interface for all the lanes together,[§§§§] reaching 512 bits for 16 lanes. To keep the same value of 2 B/DP-FLOP, a 32-lane implementation would need a 1024-bit-wide bus interface. However, this problem can be mitigated using an L1 cache for vector data (L1V), which allows large bandwidth for data residing in it without requiring a wide crossbar (Fig. 7). The design of an area efficient memory subsystem for RISC-V vector processors is described in Sec. V.

D. Soft Error Vulnerability

Vector processors typically achieve high utilization of the FPU (e.g., 97% in [27]), whereas scalar processors typically work in memory-bounded conditions and therefore achieve much lower FPU utilization. This implies an increase of soft error vulnerability of arithmetic units, as suggested by the models in [68] relating utilization and soft error vulnerability. Furthermore, the increase of frequency compared with state-of-the-art processors for space (e.g., from 250 MHz to 1 GHz) points to an increased percentage of errors from combinational logic (as shown in [69]), which compose the majority of the area in FPUs and ALUs. For instance, we synthesized the BOOM processor[¶¶¶¶] on a 65 nm ASIC technology and the area of the FPU and ALUs (comprising hardware multiplication and division) results composed, respectively, for 79.52 and 86.11% of combinational logic. Finally, scaling efficiently at least up to 16 lanes, a vector processor can achieve high performance when

[§§§§] Hwacha, instead, uses an interface per lane [67].

[¶¶¶¶] https://github.com/riscv-boom/boom-template.git.

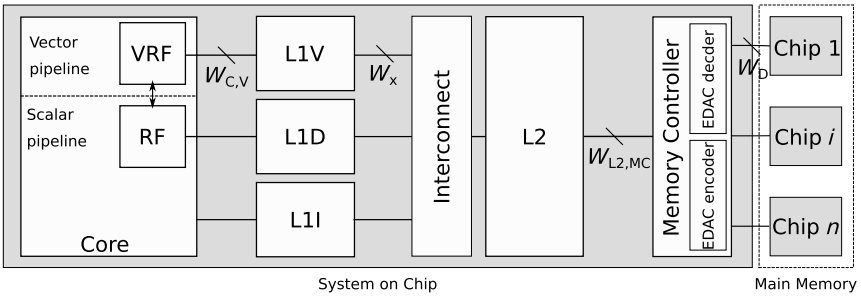

Fig. 7 Possible memory hierarchy for a vector processor. Other cores and peripherals (not shown in figure) can be connected to the interconnect.

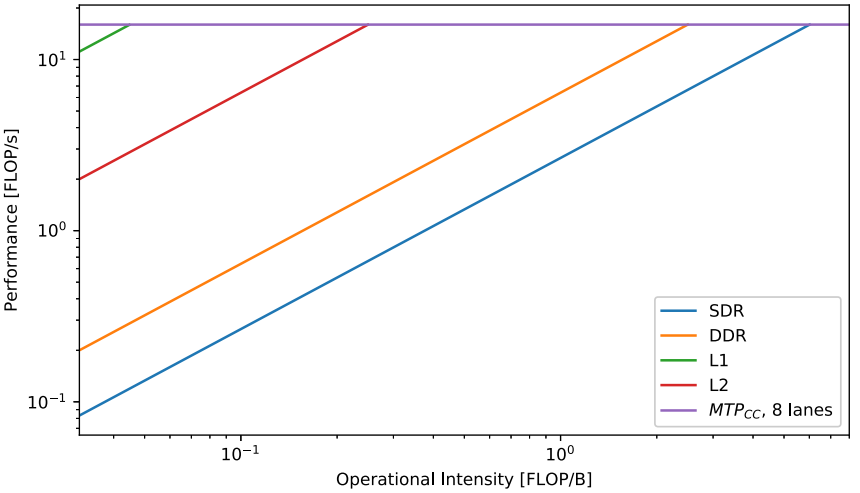

Fig. 8 Theoretical improvement for low OI workloads for matrices residing in L2 and L1V compared with SDR and DDR (single chip).

large ASIC implementations are possible. For this reason, small technology nodes should be preferred. However, in [70] it is reported that going below 28 nm increases the soft error rate (SER) in the terrestrial environment. In FD-SOI technologies this is mainly due to an increase of SER due to protons, whereas the SER due to alpha particles is slightly decreasing. Given that in space there is a different radiation environment, the technology node minimizing the SER may be different.

The separation between scalar and vector pipeline in decoupled vector processors allows for a selective hardening approach. Assuming that control operations are executed only in the scalar pipeline and computations only in the vector pipeline, redundancy to avoid catastrophic failures is required only in the scalar pipeline. In Ara, the critical path limiting the maximum frequency for the four-lane version is in the vector pipeline and allows for a maximum frequency of around 1 GHz, whereas the scalar pipeline has a critical path allowing up to 1.7 GHz [5]. Therefore, applying state-of-the-art techniques to improve fault tolerance only to the scalar pipeline, such as triple modular redundancy (TMR) at flip-flop level in the scalar pipeline and error detection and correction (EDAC) codes in the scalar register file, will not cause any penalties in terms of maximum frequency and hence in terms of MTP_{CC}. As a matter of fact, TMR and EDAC are reported to cause only 9% decrease in frequency in the LEON2 [71]. A similar decrease would keep the maximum frequency of Ariane from 1.7 GHz [5] to around 1.5 GHz, which is still above the maximum frequency possible in the vector pipeline.

V. Memory Hierarchy

Figure 7 shows a possible memory hierarchy for a vector processor. As a typical memory hierarchy for scalar processors, it comprises an L1 cache for scalar data (L1D), a L1 instruction cache (L1I), a unified level 2 cache (L2),***** and a main memory. However, an L1V is added to increase performance especially for workloads with low OI. The figure also indicates the width W_i of the interface between levels, which determines the bandwidth B_i of the interface together with its clock frequency f_{clk_i}, according to $B_i = f_{clk_i} * W_i$. For instance, the Sandy Bridge in [33] has a 384-bit interface and a maximum bandwidth of 384 b/CC. In the case of DRAMs, B_D is given by $R_D * C_D * f_{clk_D} * W_D$, where R_D is 1 for *SDR* and 2 for *DDR*, C_D is the number of channels for the DRAM, f_{clk} the clock frequency, and W the word size. For the DRAM employed in the Sandy Bridge in [33] $C_D = 2$, $f_{clk_D} = 0.8$ GHz, and $W_D = 64$, and therefore B_D is 25.6 GB/s.

A cache-aware roofline model [33], shown in Fig. 8, highlights the main benefits of adopting a memory hierarchy similar to Fig. 7. When data reside in main memory, OI* is around 2.50–6.02 FLOP/B (depending on the DRAM technology), whereas if data reside in an L2 (with $W_X = 64$ b) OI* becomes 0.25 FLOP/B and a dedicated L1V with $W_{C,V} = 356$ b reduces OI* to 0.04 FLOP/B. Furthermore, from Fig. 8 it can be deduced that keeping a processor in a compute-bounded state for a given OI puts increasingly higher requirements on the memory bandwidth when MTP_{CC} (hence the computational capabilities) is increased (e.g., an implementation with lower MTP_{CC} has a lower OI*). As a result, extremely high-performance processors for DNNs are actually memory-bounded except for very high OI [50].

A. Main Memory

The need for (at least) radiation-tolerant parts with solid flight heritage limits the use of state-of-the-art memories. As a result, main memories for space in ESA missions lag behind commercial counterparts in terms of performance. For instance, state-of-the-art OBCs typically employ single data rate (SDR) DRAM [72]. The SDR DRAM tested in [25] (ISSI IS42S86400B-7TL) has 16 bits for data I/O and achieves up to 166 MHz. Therefore, its B_D is 2.66 Gbps, i.e., two orders of magnitude less compared with the DDR3 DRAM memories used in [33]. Faster DRAMs are also being considered, as the DDR2 tested in [25] (IS43DR81280B-25DBLI), which has

*****This is typically the case of multicore processors (not shown in the figure), where more cores with their own L1 caches are connected to the L2 via an interconnect.

8 bits for I/O data and achieves up to 400 MHz. This means a B_D of 6.4 Gbps, which is still more than one order of magnitude lower compared with the DDR DRAM in [33].

1. EDAC Codes

In the space environment, DRAMs suffer from single event upsets (SEUs) and multiple bit upsets (MBUs) as SRAMs [73]. However, in DRAMs most of the upsets happen in weakened cells [74]. Furthermore, compared with SRAMs, DRAMs are also more likely to suffer from stuck bits (cells stuck to a value, mostly related to variable bit retention [75]) and single event functional interrupts (SEFIs). The effect of SEFIs in a DRAM ranges from some tens of bits to a full chip wrong per read cycle and can be recovered only with a chip reset or sometimes with a full power cycle [74]. To detect and correct these errors, EDAC codes are employed in the DRAM. Including EDAC checkbits in DRAMs decreases the bandwidth, as also checkbits are read and written, and increases latency, as the checkbits have to be calculated before storing the data in memory and checked before using the data read from the memory. For DRAMs in space embedded systems, typically Reed–Solomon (RS) codes are employed [51]. An $RS(n, k)$ code takes a word of k symbols and generates a code word of n symbols, where $n = k + 2t$ (with $2t$ being the number of check symbols). RS codes have a redundancy $r = 2t/k$, where r is typically 25 or 50%, meaning that they increase the number of bits required to express the information by r. RS codes can correct errors in up to t symbols [76]. Regarding the symbol size, conventional organization of DRAM-based memories uses several chips in parallel to constitute a rank of the desired data width (e.g., 64 bits) [77]. It is therefore a straightforward choice to use as symbol length the IO width of a single chip. We will assume the chip to have an IO width of 8 bits and therefore employ a byte-based RS, although different choices are possible. In this way, SEFIs can be masked and the failed chip can be reset when it is less detrimental to functional availability. For instance, in [51], 16 check bits or 32 check bits for 64 bits of data are employed, meaning, respectively, RS(10,8) and RS(12,8).

RS comes with substantial penalties in terms of performance. When adding RS with $r = 25\%$ and $r = 50\%$ to a memory module with n chips, the average effective data bandwidth per chip becomes, respectively, 75 and 50% of the original DRAM bandwidth. Therefore, the bandwidth of the SDR DRAM in [25] with RS reduces from 2.66 Gbps to 2.00 Gbps ($r = 0.25$) and 1.33 Gbps ($r = 0.5$), whereas the bandwidth of the DDR2 reduces from 5.1 Gbps to 3.8 Gbps ($r = 0.25$) and 2.6 Gbps ($r = 0.5$). Furthermore, RS codes come also with a substantial penalty in terms of latency. For instance, the decoder proposed in [78] has a latency of $L = n + 10t + 20$ CCs. Typically, critical paths of memory controllers (MCs) are deeper than those of processors and run at lower frequency. For instance, the length of the critical path reported in [79] ranges between 547 and 48 gates depending on the design complexity. Assuming 0.02 ns per gate as for Ara [27], this limits the frequency in a range between 1.04 GHz and 91 MHz. Therefore, we assume that the decoder runs at half the frequency of Ara and we partially compensate this with a doubled data width between the MC and L2 compared with the one between L2 and L1V. Therefore, $W_{L1,L2} = W_{L2,\mathrm{MC}}/(f_{\mathrm{CPU}}/f_{\mathrm{MC}})$, where f_{CPU} and f_{MC} are, respectively, the frequency of the vector processor and the frequency of the MC. Therefore, the latency expressed in terms of CCs of Ara, keeping into account that $n = (1 + r)k$ and $t = (r/2)k$, is $L_{\mathrm{CPU}} = (f_{\mathrm{CPU}}/f_{\mathrm{MC}})[(1 + r) * k + 6r * k + 20]$. Given that $k = W_{L2,\mathrm{MC}}/8$ (as a symbol is composed of 8 bits) and $W_{L1,L2} = 32 * N_L$ (following the rule of thumb reported in Sec. IV.C.4), the final expression is

$$L_{\mathrm{CPU}} = 4N_L(1 + 6r)\left(\frac{f_{\mathrm{CPU}}}{f_{\mathrm{MC}}}\right)^2 + 20\frac{f_{\mathrm{CPU}}}{f_{\mathrm{MC}}} \tag{8}$$

It should be noted that the latency of this design has a quadratic dependence on the ratio of the frequencies and only a linear dependence on the number of lanes N_L. Therefore, having a low $f_{\mathrm{CPU}}/f_{\mathrm{MC}}$ ratio is very effective to help the scaling of performance with the number of lanes. Substituting $N_L = 4$, $r = 0.25$, and $f_{\mathrm{CPU}}/f_{\mathrm{MC}} = 2$, we estimate 200 CCs of additional latency seen by the processor during reads due to the use of RS. This is a significant increase (e.g., read latency of the DRAM chip around 20 ns [26], i.e., 15–20 CCs for $f_{\mathrm{CPU}} = 1$ GHz), and therefore it may be required to lower the level of information redundancy or not applying EDAC altogether on vector data to achieve the required level of performance.

2. Vulnerability of DNN Parameters

To evaluate the effect of not applying EDAC on the DRAM when running a DNN, we estimate the effect of upsets on the parameters residing in the DRAM for CloudNet.

According to [74], a 512 Mb SDR DRAM memory (MMSD08512408S-Y) experiences 2.75e–11 upset/bit/day in LEO. Therefore, 0.19 upsets/day are to be expected for coefficients and feature maps residing in the DRAM (using the peak memory reported in Sec. III). To assess the sensitivity to SEUs, we ran a fault injection campaign on the DNN coefficients expressed in SP floating point (expected to reside in the memory buffer) during the inference. For each experiment a single error is injected and the accuracy of the classification over 9201 input patches is checked. The metric employed to estimate the accuracy is the overall accuracy (OA) defined in [34] as

$$\mathrm{OA} = \frac{\mathrm{TN} + \mathrm{TP}}{\#\mathrm{Pixels}} \tag{9}$$

where TN (true negatives) is the number of pixels correctly classified as without clouds, TP (true positives) is the number of pixels correctly classified as covered by clouds, and #Pixels is the total number of pixels (therefore comprising also false negatives and false positives). For a fault-free execution over the 9201 patches of the test set, the OA is 96.5%. In the majority of the cases, injecting upsets in the input images causes little or no damage to the accuracy of the DNN and the OA usually does not go below 96.5%, except for when the bit flip happens in the most significant bit (MSB) of the exponent. In this case, a single bit flip can change a very small number in a very large number and vice versa. For instance, 1.4293875e-05 (0x376FCFBA) can be turned into 4.8639537e+33 (0x776FCFBA). Therefore, even setting a very tight requirement on the OA, an SEU in a coefficient has a 1 in 32 chance of causing the DNN to fail. Another large deviation could take place when the bit flip happens in the sign bit and the data have a large magnitude. This is not the case in CloudNet, as the maximum magnitude found for the parameters is around 0.59. This is also to be expected in other DNNs, as typically regularization techniques that keep the magnitude of parameters low are employed to avoid overfitting [43]. As an extreme case condition for the upset rate, we ran also experiments with 10 upsets simultaneously. Also in this case we note that large deviations are present only if one of the upsets is in the MSB of the exponent (e.g., OA = 61.3%). Assuming 0.19 upsets/day and that only upsets in the MSB of the exponent will cause a failure due to insufficient quality of service (QoS), we can expect upsets to cause a failure due to SEU for insufficient QoS every 165.4 days.

Other DNN architectures may be more vulnerable to SEUs. For instance, in [80] it is shown that the FC layers in the last layers of CNNs are more vulnerable compared with early convolutional layers. However, the dependence of the vulnerability of a bit on its position is related to the format of the coefficients. For instance, in [81] the MSB of the exponent is found to be the most critical bit of the SP model coefficients also in CNNs and DNNs with LSTM layers. Furthermore, Ref. [80] shows that using half precision (HP) floating point can increase robustness for some architectures compared with SP floating point. Fixed point representation can mitigate the failure mechanism described for floating point thanks to their limited range [80]. However, if the fixed point representation has a large integer part (e.g., 1 bit for sign, 21 for the integer part, and 10 for the decimal part) the robustness of the DNN can be severely reduced compared with floating point representations [80].

3. Proposed Solutions for DRAMs

While the effect of SEUs on parameters can be tolerated by the intrinsic robustness of DNNs, SEFIs produce an unpredictable number of errors per CC and therefore require mitigation. According to data from [74], a 512 Mb SDR DRAM memory (MMSD08512408S-Y) experiences 1.33e−3 SEFI/device/day. To achieve the peak memory required, 14 chips are required and therefore not including any EDAC will produce a failure due to SEFIs every 53.7 days. This is unacceptable, as every inference after the SEFI is likely to have insufficient QoS until the next reset of the failing chip. As a mitigation, DRAM chips can be reset periodically. Assuming a reset every 2 h, the percentage of failed inferences due to SEFIs WI_{SEFI} in the worst case is

$$\text{WI}_{\text{SEFI}} = \frac{\text{Failures(SEFI)}}{\text{Total inferences}} = \frac{\Delta T_{\text{rst}}}{\text{MTTF}_{\text{SEFI}}} = 0.16\% \tag{10}$$

The contribution to wrong inferences of SEUs can be estimated with a similar equation, where the MTTF_{SEU} in the denominator is divided by 0.03 to account for the discussion in Sec. V.A.2 on the vulnerable bits of floating point coefficients and T_{rst} is replaced with the time required for a single inference T_{inf}. The value found is negligible (two orders of magnitude less than the contribution of SEFIs). However, the final value of average reliability $R_{\text{avg}} = 1 - WI_{\text{SEU}} - WI_{\text{SEFI}}$ (99.84%) can be not deemed enough for critical applications. The availability instead depends also on the maintenance time after a reset. If we assume a maintenance time of 30 s for each reset, we find that the availability of the service is 99.58%, whereas a maintenance time of 300 s produces an availability of 95.83%. Both values are below typical requirements of dependable systems (e.g., [82]).

A tradeoff between RS and no EDAC is represented by simpler EDAC codes. EDAC codes with lower redundancy, although they cannot mask SEFIs, can still detect some of the wrong bits caused by the SEFI. For instance, a parity bit per chip can detect an odd number of errors in a chip, and it is possible to keep track of them with a counter. When the number of errors from a chip exceeds a certain threshold in a certain time window, the DRAM chip is reset to recover from a probable SEFI. Assuming a threshold of three errors and an equal probability that the SEFI will cause an even or odd number of errors, the percentage of wrong inferences due to SEFIs is

$$\text{WI}_{\text{SEFI}} = \frac{2(N_{\text{thr}} + 1)}{\text{MTTF}_{\text{SEFI}}/\Delta T_{\text{inf}}} = 0.0009\% \tag{11}$$

Regarding SEUs, neglecting accumulation and MBUs, all the upsets are detected. Therefore $R_{av} = 99.9991\%$, which is a substantial increment compared with employing no EDAC. There is a substantial increment in availability too, with 99.9994 and 99.994%, respectively, for 30 and 300 s of unavailability per reset.

Table 3 summarizes the different EDAC and reset approaches discussed to protect DRAMs for DNNs.

B. L1 Vector Cache

Many vector processors use L1 caching for instructions and for scalar data, leaving vector data uncached (e.g., Ara [27]), as historically locality in vector workloads was assumed to be less pronounced compared with scalar workloads [83]. The work in [83] characterizes temporal and spatial locality in compute-intensive vector workloads and finds that caches can significantly improve the performance of a vector processor. Furthermore, in [84] it is shown that the use of caches helps masking memory latency, as increasing by 3.21× the latency of a memory access (from 14 CCs to 45 CCs) roughly triplicates the mean delay per memory reference for a processor with uncached vector data and less than doubles the access time for a processor with an L1 cache for vector data.

Table 3 Approaches suggested for applications with different criticality levels (reliability/availability) and achievable performance

	Approach		
Characteristic	No EDAC	Parity	RS
Reset strategy	Periodic	Threshold	Optimal
λ_{QoS}	$0.03\lambda_{\text{SEU}} + \lambda_{\text{SEFI}}$	λ_{SEFI}	$\simeq 0$
R_{avg}	Low	Medium	High
Availability	Low	Medium	High
Performance	High	Medium	Low

The following subsections will carry out a design exploration of the L1V to assess which sizes, organizations, and write policies are more efficient for vector processors.

1. Size

From Table 1, it is clear that the large matrices originating from unrolling of convolutional layers (ranging from 3 to 41 MiB) do not fit even in large L2 caches (e.g., 2 MiB [51]). This problem can be addressed with tiling, as shown in Fig. 9. In this approach, two levels of looping (shown in Fig. 9 with index i and j) select a subset of the matrix–matrix multiplication that produces one of the

$$\left\lceil \frac{UV}{b} \right\rceil \left\lceil \frac{N}{b} \right\rceil$$

tiles of the result, each composed of $b \times b$ elements. By increasing the size of the cache, it is possible to work on larger matrix blocks residing in the L1V. The subset of operations obtained in Fig 9b can be decomposed into

$$\left\lceil \frac{CJK}{b} \right\rceil$$

segments, and the results of these segments can be accumulated to generate the final result of the tile. The level (c) in Fig. 9 is where the mapping to SGEMM (described in Sec. III.A.1) can be applied.

One of the possible implementations of SGEMM (Fig. 9d) is a loop selecting the mth column of $A0$ and the mth row of $A1$ and generating a matrix where the pth column is the mth column of $A0$ multiplied by $A1_{mp}$. Vectorization is applied with a maximum vector length of V_L, with FMA (accumulate) operations between the vector $A0_m$ and a scalar $A1_{mp}$. A matrix representation of this implementation for a 2×2 example is shown below.[†††††]

$$A2 = \begin{pmatrix} A0_{11}A1_{11} + A0_{12}A1_{21} & A0_{11}A1_{12} + A0_{12}A1_{21} \\ A0_{21}A1_{11} + A0_{22}A1_{21} & A0_{21}A1_{12} + A0_{12}A1_{22} \end{pmatrix}$$
$$= \begin{pmatrix} | & | \\ A0_1A1_{11} & A0_1A1_{12} \\ | & | \end{pmatrix} + \begin{pmatrix} | & | \\ A0_2A1_{21} & A0_2A1_{22} \\ | & | \end{pmatrix}$$

As we are interested in investigating the speed increase due to the use of an L1V for small matrices, we will assume to be in memory-bounded conditions (the computations happen in parallel with part of the loads and stores, although with a shorter duration). In these conditions, the execution time can be estimated as the time required to read the matrices from main memory to the L1V and the time required to write to main memory the result a tile per time.

Loading a vector of length V_L from main memory takes

$$T_{L,V} = T_{LM} + \frac{S_E * V_L}{B_M}$$

where S_E is the size of a single element of the vector, B_M is the bandwidth of the main memory, and the latency of the first element of the vector from main memory is T_{LM}.[‡‡‡‡‡] The time required to copy a

[†††††] A similar implementation of SGEMM is described in [85].

[‡‡‡‡‡] Matrices are assumed to be stored in row-major order, as this is the order employed in the C language.

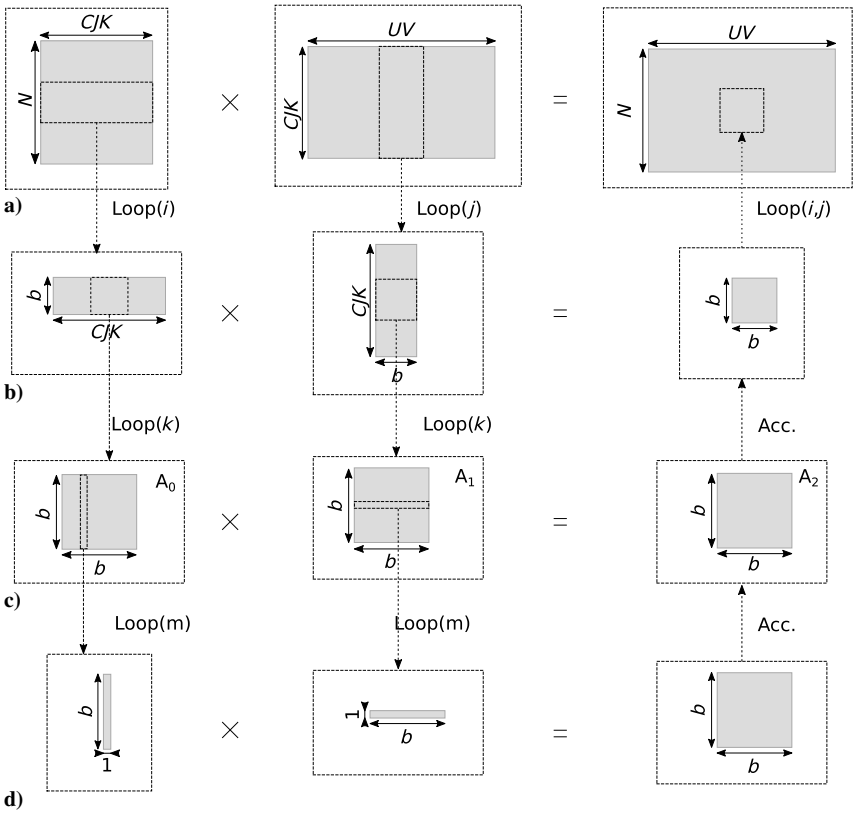

Fig. 9 Example of tiling of a matrix–matrix multiplication. "Acc." stands for accumulation.

row vector of length b from main memory to L1V is

$$T_{L,b} = T_{L,V} * \left\lceil \frac{b}{V_L} \right\rceil + \frac{SE * b}{B_M}$$

and the time required to read an entire $b \times b$ tile is $T_{L,b\times b} = T_{L,b} * b$. The time required to read a fringe $b \times b'$ tile with $b' < b$ is instead

$$T_{L,b\times b'} = \left(T_{L,V} * \left\lceil \frac{b'}{V_L} \right\rceil + \frac{SE * b'}{B_M} \right) * b$$

There are three possible implementations, depending on which tile (of the coefficient, input feature, and output feature matrix) is kept into the L1V during the innermost looping. Assuming that the output feature matrix is kept in L1V, the time required during the loop on $CJK \times b$ to load all the tiles in a $CJK \times b$ stripe of the $CJK \times UV$ input feature matrix (as shown in Fig. 9b) is

$$T_{L,CJK\times b} = T_{L,b\times b} * \frac{CJK}{b}$$

whereas for a $b \times CJK$ stripe of the $N \times CJK$ matrix

$$T_{L,b\times CJK} = T_{L,b\times b} * \left\lfloor \frac{CJK}{b} \right\rfloor + T_{b\times b'}$$

where $b' = (CJK) \bmod (b)$. As every column has to be multiplied for every row, the total time spent reading the coefficient matrix is

$$T_{L,N\times CJK} = T_{L,b\times CJK} * \left\lceil \frac{UV}{b} \right\rceil * \frac{N}{b}$$

where the ceiling is required because all the matrix of the coefficient is to be read again even if only one column of the input feature is left to be loaded. Similarly, the total time spent reading the $CJK \times UV$ matrix is instead

$$T_{L,CJK\times UV} = \left(T_{L,CJK\times b} * \left\lfloor \frac{UV}{b} \right\rfloor + T_{b\times b'} \frac{CJK}{b} \right) \left\lceil \frac{N}{b} \right\rceil$$

with $b' = (UV) \bmod (b)$.

Similar equations can be derived for storing the result, substituting the subscript L with S. Only the final result for each tile is written to main memory; therefore the time to store all the results is

$$T_{S,N\times UV} = \left(T_{S,N\times b} * \left\lfloor \frac{UV}{b} \right\rfloor + T_{L,b\times b'} * \frac{N}{b} \right)$$

with $b' = (UV) \bmod (b)$.

Considering the associate continuous functions (without modulo, ceiling, and floor functions), it is possible to prove that the fastest implementation is the one keeping in L1V the tile of the output feature matrix. This is because this implementation does not require loading and storing of the temporary tile of the output matrix during accumulation.

To trade off the speed-up against the increase in size due to a larger L1V, we consider the area efficiency in terms of FLOP/CC/GE for matrix multiplications with matrices residing in L1V. To give a realistic estimation of the cache size that maximizes the area efficiency, we consider what the effect of adding an L1V to Ara would be in terms of area. The area of Ariane and Ara ranges from 2228 for two lanes to 10,735 kGE for 16 lanes. As a worst case for memory-bounded conditions, we assume 16 lanes ($V_L = 16$), and in this case the area without L1V is 10,735 GE. The area of the L1 cache is estimated as $A_{L1V,\mathrm{GE}} = (6/4)N_b$, assuming 6T SRAM cells and a GE corresponding to four transistors.

We will consider four cases comprising all the combinations of memory with latency 50 CCs (representative of the latency without RS) and 300 CCs (representative of the latency with RS) and with bandwidths of 4 and 40 b/CC (respectively, representative of a memory module with 4 SDR chips and 4 DDR chips). Table 4 shows the results of this model. The main observations are that the optimal size of L1V is much larger (256 KiB-1 MiB) than a typical L1D (e.g., 16 KiB [51]) and that the most impacting factor on the area

Table 4 Estimates of area A_{tot} [MGE] and area efficiency AE [FLOP/CC/MGE] for a 16-lane vector processor with different sizes of L1V, main memory (latency and bandwidth), and maximum size of the tile $b \times b$ when applying tiling to the layers of CloudNet

Characteristic	64 KiB	128 KiB	256 KiB	512 KiB	1 MiB	2 MiB
b	40	60	84	120	168	240
A_{tot}	11.5	12.3	13.9	17.0	23.3	35.9
Layer 1: $C = 4$; $N = 16$; $J = K = 3$; $U = V = 192$						
$AE_{(50,40)}$	1.06E + 0	1.44E + 0	1.91E + 0	2.35E + 0	**2.44*E* + 0**	2.34E + 0
$AE_{(50,4)}$	4.09E − 1	5.44E − 1	7.23E − 1	8.59E − 1	**8.82*E* − 1**	8.28E − 1
$AE_{(300,4)}$	1.57E − 1	2.14E − 1	2.84E − 1	3.48E − 1	**3.59*E* − 1**	3.44E − 1
$AE_{(300,40)}$	2.06E − 1	2.83E − 1	3.76E − 1	4.68E − 1	**4.86*E* − 1**	4.71E − 1
Layer 11: $C = 128$, $N = 256$, $J = K = 3$, $U = V = 24$						
$AE_{(50,40)}$	4.04E − 1	1.58E + 0	2.03E + 0	**2.40*E* + 0**	2.33E + 0	2.08E + 0
$AE_{(50,4)}$	2.62E − 1	5.97E − 1	7.65E − 1	**8.72*E* − 1**	8.43E − 1	7.33E − 1
$AE_{(300,4)}$	6.48E − 2	2.35E − 1	3.01E − 1	**3.54*E* − 1**	3.43E − 1	3.05E − 1
$AE_{(300,40)}$	7.09E − 2	3.11E − 1	3.99E − 1	**4.78*E* − 1**	4.63E − 1	4.17E − 1
Layer 19: $C = 512$, $N = 1024$, $J = K = 3$, $U = V = 6$						
$AE_{(50,40)}$	6.24E − 1	7.07E − 1	**7.44*E* − 1**	7.10E − 1	5.74E − 1	4.15E − 1
$AE_{(50,4)}$	3.69E − 1	2.77E − 1	**2.89*E* − 1**	2.68E − 1	2.13E − 1	1.50E − 1
$AE_{(300,4)}$	9.35E − 2	1.06E − 1	**1.11*E* − 1**	1.05E − 1	8.51E − 2	6.11E − 2
$AE_{(300,40)}$	1.21E − 1	1.38E − 1	**1.45*E* − 1**	1.40E − 1	1.13E − 1	8.26E − 2

Peak values in bold.

efficiency is the dimensions of the convolution. For each layer, one cache size maximizes the area efficiency independently of latency and bandwidth. This value decreases from 1 MiB to 256 KiB when going from layers with large $U = V$ and small C and N to layers with small $U = V$ and large C and N. This means that processors intended to run deeper CNNs can employ smaller caches with lower penalty. However, the maximum area efficiency decreases going from layer 1 to layer 11 to layer 19.

2. Organization

The model in the previous section assumes that it is possible to keep the tiles in L1V, avoiding that loading a vector of one of the tiles causes the eviction of data belonging to one of the other tiles required. Whether this happens or not depends on the cache organization and an ineffective organization requires larger caches to allow the tiles to reside in the cache during computations.

Data-parallel ISA extensions (also the RVVE [64]) typically support vector load and store operations with nonunit stride V_S; i.e., two contiguous elements of the vector are placed in noncontiguous location separated by $V_S - 1$ elements. According to the model in [84], the fraction of nonunit strides in a workload determines whether organizations similar to those of scalar processors are enough to achieve acceptable performance or organizations specific for vector processors are required. One example of the latter is prime-mapped caches [84], which have a conflict-free memory organization for vectors with power-of-two strides. However, they have no advantage against direct-mapped caches (the simplest cache organization for scalar processor) when all the strides are unitary. In [53] the breakdown of vector access in terms of vector memory accesses for 20 benchmarks running on tree different vector machines (Cray90, Alliant FX/8, Convex C3) is reported. The respective percentages are 66.37% unit stride, 24.24% other strides, and 9.40% indexed (also known as "scatter and gather" and also supported by the RVVE [64]). The improvement with prime-mapped caches for a typical workload with unit stride of 70% is 2× over the cacheless version, whereas the improvement for direct mapped caches is below 1.5× [84].

Typical applications that require nonunit strides are fast Fourier transform (FFT) and its inverse (IFT) [84]. FFT is employed in several compute-intensive workloads. For instance, in [86] it is proposed to speed-up CNN execution, as convolutions can be substituted by a sequence of FFT, elementwise multiplication, and IFT.

To investigate whether vector loads and stores with nonunit strides are present in DNNs, we translated CloudNet into ARM NEON assembly (which supports vector load and store strides of size 1,2,3,4,8) using TVM.[§§§§§] The fraction of vector accesses for stride 1, stride 2, and stride 4 are, respectively, 97.13, 1.62, and 1.25%. No accesses with stride 3 (supported in NEON) have been found. Translating other DNNs leads instead to only unit stride accesses. For instance, translating the popular resnet18_v1 [41] model did not produce nonunit stride accesses.

These findings suggest that, although in a first phase this problem could be mitigated relying on certain choices of DNN architectures and software implementation to reduce the fraction of nonunit vector strides, in general different cache organizations are needed compared with those typically employed for scalar processors.

3. Write Policy

A microarchitecture with separated scalar and vector data caches requires a solution to handle memory coherence issues when data in one of the two is modified and an old value is read from the other. This can be addressed with a write-through policy for L1V and L1D, although this comes with substantial penalties especially in terms of power [87], memory traffic [88], and performance [89].

VI. Conclusions

The recent shift of focus of the space industry from large GEO to small LEO satellites opens up new challenges. Limited downlink data rates and short communication windows typically allow the transmission of just a fraction of the data generated by on-board sensors in small LEO satellites. The efficiency of the downlink can be increased with data compression and with data removal (e.g., removing images that have a certain percentage of pixels covered with clouds). This solution requires a dedicated processor that comes at relatively high cost in terms of power (around 5 W), which can be sustained only by relatively large satellites. Furthermore, long periods without contact with the base station require an on-board virtual operator, monitoring the status of the satellite and making decisions when the communication with the ground station is not possible.

These challenges in terms of downlink efficiency and dependability can be addressed with DNNs when it is possible to build relatively large datasets (e.g., thousands of images or months of telemetry). Therefore, there is a need for large, public, and standardized datasets to be used as challenges for DNN architectures to

[§§§§§]https://github.com/apache/incubator-tvm.

be deployed in space applications. However, part of future LEO satellites are planned to be launched in large constellations, making large datasets more easily available in the future.

The analysis of the workloads associated with DNNs shows that most parts are very compute-intensive and can be mapped to matrix–matrix multiplications, for which DLP is the most efficient microarchitectural solution to increase execution speed. Among the data-parallel ISA extensions available, the RVVE is gaining momentum because of its openness and efficiency. Although there are already processors based on the RVVE, the software ecosystem of the RVVE is in an early stage, as the ISA specifications are not frozen yet. Therefore, during the early development of a RISC-V vector processor, some adjustments may be required. This is a risk that can be accepted given the long development times of space processors.

The analysis of the microarchitecture of a vector processor shows possible criticalities both for computational capabilities and for the memory hierarchy. For instance, the scalability with the number of lanes can be an issue, especially for operations involving all of them. The width of the bus interface has also been found to be a possible bottleneck, and the use of an L1V has been suggested as a possible mitigation approach. L1 caches for vector data maximize the area efficiency when executing convolutional layers when their size is around 256 KiB–1 MiB. Furthermore, the microarchitecture of the scalar pipeline affects the performance for small OI, given the limited issue rate of microarchitectures with low ILP. Furthermore, it is possible to apply to decoupled vector and scalar pipelines different approaches in terms of redundancy to reduce penalties in terms of performance.

The relatively large size and the focus on high performance of vector processors requires the identification of a radiation-tolerant ASIC technology with a technology node around 28 nm (considering also the SER), whereas state-of-the-art processors in space systems are typically still based on RHBD 65 nm technologies. Furthermore, an ASIC technology with multiported SRAMs is required for an area-efficient implementation of the VRF.

Finally, this work investigated the performance and dependability characteristics of the main memory, one of the most important tradeoffs in space embedded systems. Demanding applications (e.g., image classification) require a main memory with around 1 GiB capacity, which is more than the typical DRAM capacity required in many space mission. When availability is not a primary concern, EDAC codes for DRAMs with low redundancy and latency can be employed to detect SEFIs and restart DRAM chips in non-critical applications. In even less critical applications, periodic resets of DRAM chips can be deemed sufficient. For critical applications RS is still required. Therefore, some performance-demanding applications requiring high availability (e.g., online processing) may be unfeasible.

Acknowledgments

This work was supported by the European Space Agency under the NPI Program, Cobham Gaisler AB, and Delft University of Technology.

References

[1] Lemley, J., Bazrafkan, S., and Corcoran, P., "Deep Learning for Consumer Devices and Services: Pushing the Limits for Machine Learning, Artificial Intelligence, and Computer Vision," *IEEE Consumer Electronics Magazine*, Vol. 6, No. 2, 2017, pp. 48–56. https://doi.org/10.1109/MCE.2016.2640698

[2] Schwank, J. R., Shaneyfelt, M. R., and Dodd, P. E., "Radiation Hardness Assurance Testing of Microelectronic Devices and Integrated Circuits: Radiation Environments, Physical Mechanisms, and Foundations for Hardness Assurance," *IEEE Transactions on Nuclear Science*, Vol. 60, No. 3, 2013, pp. 2074–2100. https://doi.org/10.1109/TNS.2013.2254722

[3] Wyrwas, E., "Proton Testing of AMD e9173 GPU," 2019, https://nepp.nasa.gov/files/30362/NEPP-TR-2019-Wyrwas-TR-19-022_AMD-e9173-GPU-2019 June02-TN72682.pdf.

[4] Di Mascio, S., Menicucci, A., Gill, E., Furano, G., and Monteleone, C., "Leveraging the Openness and Modularity of RISC-V in Space," *Journal of Aerospace Information Systems*, Vol. 16, No. 11, 2019, pp. 454–472. https://doi.org/10.2514/1.I010735

[5] Zaruba, F., and Benini, L., "The Cost of Application-Class Processing: Energy and Performance Analysis of a Linux-Ready 1.7-GHz 64-Bit RISC-V Core in 22-nm FDSOI Technology," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 27, No. 11, 2019, pp. 2629–2640. https://doi.org/10.1109/TVLSI.2019.2926114

[6] Li, X., Adve, S. V., Bose, P., and Rivers, J. A., "Architecture-Level Soft Error Analysis: Examining the Limits of Common Assumptions," *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, IEEE Publ., Piscataway, NJ, 2007, pp. 266–275. https://doi.org/10.1109/DSN.2007.15

[7] Blacker, P., Bridges, C. P., and Hadfield, S., "Rapid Prototyping of Deep Learning Models on Radiation Hardened CPUs," *2019 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, IEEE Publ., Piscataway, NJ, 2019, pp. 25–32. https://doi.org/10.1109/AHS.2019.000-4

[8] Lai, L., and Suda, N., "Enabling Deep Learning at the IoT Edge," *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, IEEE Publ., Piscataway, NJ, 2018, pp. 1–6. https://doi.org/10.1145/3240765.3243473

[9] Furano, G., Meoni, G., Dunne, A., Moloney, D., Ferlet-Cavrois, V., Tavoularis, A., Byrne, J., Buckley, L., Psarakis, M., Voss, K.-O., and Fanucci, L., "Towards the Use of Artificial Intelligence on the Edge in Space Systems: Challenges and Opportunities," *IEEE Aerospace and Electronic Systems Magazine*, Vol. 35, No. 12, 2020, pp. 44–56. https://doi.org/10.1109/MAES.2020.3008468

[10] Lentaris, G., Maragos, K., Stratakos, I., Papadopoulos, L., Papanikolaou, O., Soudris, D., Lourakis, M., Zabulis, X., Gonzalez-Arjona, D., and Furano, G., "High-Performance Embedded Computing in Space: Evaluation of Platforms for Vision-Based Navigation," *Journal of Aerospace Information Systems*, Vol. 15, No. 4, 2018, pp. 178–192. https://doi.org/10.2514/1.I010555

[11] Pignol, M., "COTS-Based Applications in Space Avionics," *2010 Design, Automation Test in Europe Conference Exhibition (DATE 2010)*, IEEE Publ., Piscataway, NJ, 2010, pp. 1213–1219. https://doi.org/10.1109/DATE.2010.5456992

[12] Del Sozzo, E., Solazzo, A., Miele, A., and Santambrogio, M. D., "On the Automation of High Level Synthesis of Convolutional Neural Networks," *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, IEEE Publ., Piscataway, NJ, 2016, pp. 217–224. https://doi.org/10.1109/IPDPSW.2016.153

[13] Xi, S. L., Yao, Y., Bhardwaj, K., Whatmough, P., Wei, G.-Y., and Brooks, D., "SMAUG: End-to-End Full-Stack Simulation Infrastructure for Deep Learning Workloads," *ACM Transactions on Architecture and Code Optimization*, Vol. 17, No. 4, 2020, pp. 1–26. https://doi.org/10.1145/3424669

[14] Andersson, J., "Development of a NOEL-V RISC-V SoC Targeting Space Applications," *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, IEEE Computer Soc., Los Alamitos, CA, 2020, pp. 66–67. https://doi.org/10.1109/DSN-W50199.2020.00020

[15] "The RISC-V Instruction Set Manual Volume I: Unprivileged ISA, Document Version 20190608-Base-Ratified," RISC-V Foundation, 2019, https://content.riscv.org/wp-content/uploads/2019/06/riscv-spec.pdf.

[16] Henry, C., "Geostationary Satellite Orders Bouncing Back," 2020, https://spacenews.com/geostationary-satellite-orders-bouncing-back/.

[17] Lal, B., Sylak-Glassman, E., Mineiro, M., Gupta, N., Pratt, L., and Azari, A., "Global Trends in Space Volume 2: Trends by Subsector and Factors that Could Disrupt Them," Vol. 2, Inst. for Defense Analyses, Science & Technology Policy Inst., IDA Paper P-5242, 2015, https://www.ida.org/-/media/feature/publications/g/gl/global-trends-in-space-volume-2-trends-by-subsector-and-factors-that-could-disrupt-them/p5242v2.ashx.

[18] Maral, G., Bousquet, M., and Sun, Z., *Satellite Communications Systems: Systems, Techniques and Technology*, Wiley, Hoboken, NJ, 2020, Chap. 1.

[19] Radtke, J., Kebschull, C., and Stoll, E., "Interactions of the Space Debris Environment with Mega Constellations—Using the Example of the OneWeb Constellation," *Acta Astronautica*, Vol. 131, Feb. 2017, pp. 55–68. https://doi.org/10.1016/j.actaastro.2016.11.021

[20] Selva, D., and Krejci, D., "A Survey and Assessment of the Capabilities of Cubesats for Earth Observation," *Acta Astronautica*, Vol. 74, May

2012, pp. 50–68.
https://doi.org/10.1016/j.actaastro.2011.12.014
[21] OMeara, C., Schlag, L., and Wickler, M., "Applications of Deep Learning Neural Networks to Satellite Telemetry Monitoring," *2018 SpaceOps Conference*, AIAA Paper 2018-2558, 2018.
https://doi.org/10.2514/6.2018-2558
[22] Tsitas, S., and Kingston, J., "6U CubeSat Design for Earth Observation with 6.5m GSD, Five Spectral Bands and 14 Mbps Downlink," *Aeronautical Journal*, Vol. 114, No. 1161, 2010, pp. 689–697.
https://doi.org/10.1017/S0001924000004176
[23] Gillette, A., Wilson, C., and George, A. D., "Efficient and Autonomous Processing and Classification of Images on Small Spacecraft," *2017 IEEE National Aerospace and Electronics Conference (NAECON)*, IEEE Publ., Piscataway, NJ, 2017, pp. 135–141.
https://doi.org/10.1109/NAECON.2017.8268758
[24] Goward, S. N., Masek, J. G., Williams, D. L., Irons, J. R., and Thompson, R. J., "The Landsat 7 Mission: Terrestrial Research and Applications for the 21st Century," *Remote Sensing of Environment*, Vol. 78, No. 1, 2001, pp. 3–12.
https://doi.org/10.1016/S0034-4257(01)00262-0
[25] Guertin, S. M., and Amrbar, M., "Single Event Testing of SDRAM, DDR2 and DDR3 Memories," *2016 IEEE Radiation Effects Data Workshop (REDW)*, IEEE Publ., Piscataway, NJ, 2016, pp. 1–7.
https://doi.org/10.1109/NSREC.2016.7891742
[26] "IS43/46DR81280B(L), IS43/46DR16640B(L) Datasheet," Integrated Silicon Solution, Inc. (ISSI), 2015, http://www.issi.com/WW/pdf/43-46DR81280B-16640B.pdf.
[27] Cavalcante, M., Schuiki, F., Zaruba, F., Schaffner, M., and Benini, L., "Ara: A 1-GHz+ Scalable and Energy-Efficient RISC-V Vector Processor with Multiprecision Floating-Point Support in 22-nm FD-SOI," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 28, No. 2, 2020, pp. 530–543.
https://doi.org/10.1109/TVLSI.2019.2950087
[28] Cappellone, D., Di Mascio, S., Furano, G., and Ottavi, A. M. M., "On Board Satellite Telemetry Forecasting with RNN on RISC-V Based Multicore Processor," *2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, IEEE Publ., Piscataway, NJ, 2020, pp. 1–6.
https://doi.org/10.1109/DFT50435.2020.9250796
[29] Sze, V., Chen, Y.-H., Yang, T.-J., and Emer, J. S., "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," *Proceedings of the IEEE*, Vol. 105, No. 12, 2017, pp. 2295–2329.
https://doi.org/10.1109/JPROC.2017.2761740
[30] Luo, C., Li, X., Wang, L., He, J., Li, D., and Zhou, J., "How Does the Data Set Affect CNN-based Image Classification Performance?" *2018 5th International Conference on Systems and Informatics (ICSAI)*, IEEE Publ., Piscataway, NJ, 2018, pp. 361–366.
https://doi.org/10.1109/ICSAI.2018.8599448
[31] Phiri, D., and Morgenroth, J., "Developments in Landsat Land Cover Classification Methods: A Review," *Remote Sensing*, Vol. 9, No. 9, 2017, pp. 967.
https://doi.org/10.3390/rs9090967
[32] Williams, S., Waterman, A., and Patterson, D., "Roofline: An Insightful Visual Performance Model for Multicore Architectures," *Communications of the ACM*, Vol. 52, No. 4, 2009, p. 65–76.
https://doi.org/10.1145/1498765.1498785
[33] Ilic, A., Pratas, F., and Sousa, L., "Cache-Aware Roofline Model: Upgrading the Loft," *IEEE Computer Architecture Letters*, Vol. 13, No. 1, 2014, pp. 21–24.
https://doi.org/10.1109/L-CA.2013.6
[34] Mohajerani, S., and Saeedi, P., "Cloud-Net: An End-to-End Cloud Detection Algorithm for Landsat 8 Imagery," *IGARSS 2019—2019 IEEE International Geoscience and Remote Sensing Symposium*, IEEE Publ., Piscataway, NJ, 2019, pp. 1029–1032.
https://doi.org/10.1109/IGARSS.2019.8898776
[35] Shelhamer, E., Long, J., and Darrell, T., "Fully Convolutional Networks for Semantic Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 39, No. 4, 2017, pp. 640–651.
https://doi.org/10.1109/TPAMI.2016.2572683
[36] Bianco, S., Cadene, R., Celona, L., and Napoletano, P., "Benchmark Analysis of Representative Deep Neural Network Architectures," *IEEE Access*, Vol. 6, Oct. 2018, pp. 64,270–64,277.
https://doi.org/10.1109/ACCESS.2018.2877890
[37] Abdelouahab, K., Pelcat, M., Sérot, J., and Berry, F., "Accelerating CNN inference on FPGAs: A Survey," 2018, http://arxiv.org/abs/1806.01683.
[38] Dumoulin, V., and Visin, F., "A Guide to Convolution Arithmetic for Deep Learning," arXiv preprint arXiv:1603.07285, 2016.
[39] Chellapilla, K., Puri, S., and Simard, P., "High Performance Convolutional Neural Networks for Document Processing," *Tenth International Workshop on Frontiers in Handwriting Recognition*, edited by G. Lorette, Univ. de Rennes 1, Suvisoft, La Baule (France), 2006, https://hal.inria.fr/inria-00112631.
[40] Heinecke, A., Vaidyanathan, K., Smelyanskiy, M., Kobotov, A., Dubtsov, R., Henry, G., Shet, A. G., Chrysos, G., and Dubey, P., "Design and Implementation of the Linpack Benchmark for Single and Multi-node Systems Based on Intel® Xeon Phi Coprocessor," *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, 2013, pp. 126–137.
https://doi.org/10.1109/IPDPS.2013.113
[41] He, K., Zhang, X., Ren, S., and Sun, J., "Deep Residual Learning for Image Recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE Publ., Piscataway, NJ, 2016, pp. 770–778.
https://doi.org/10.1109/CVPR.2016.90
[42] Ioffe, S., and Szegedy, C., "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *arXiv preprint arXiv:1502.03167*, 2015.
[43] Phaisangittisagul, E., "An Analysis of the Regularization Between L2 and Dropout in Single Hidden Layer Neural Network," *2016 7th International Conference on Intelligent Systems, Modelling and Simulation (ISMS)*, IEEE Publ., Piscataway, NJ, 2016, pp. 174–179.
https://doi.org/10.1109/ISMS.2016.14
[44] Lai, L., Suda, N., and Chandra, V., "Cmsis-nn: Efficient Neural Network Kernels for Arm Cortex-m cpus," arXiv preprint arXiv:1801.06601, 2018.
[45] Lee, C.-Y., Gallagher, P., and Tu, Z., "Generalizing Pooling Functions in CNNs: Mixed, Gated, and Tree," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 40, No. 4, 2018, pp. 863–875.
https://doi.org/10.1109/TPAMI.2017.2703082
[46] Cong, J., and Xiao, B., "Minimizing Computation in Convolutional Neural Networks," *Artificial Neural Networks and Machine Learning—ICANN 2014*, edited byS. Wermter, C. Weber, W. Duch, T. Honkela, P. Koprinkova-Hristova, S. Magg, G. Palm, and A. E. P. Villa, Springer International Publishing, Cham, Switzerland, 2014, pp. 281–290.
https://doi.org/10.1007/978-3-319-11179-7_36.
[47] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y., "Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation," *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Assoc. for Computational Linguistics, Stroudsburg, PA, 2014, pp. 1724–1734.
[48] Graves, A., "Supervised Sequence Labelling with Recurrent Neural Networks," Ph.D. Dissertation, Technical Univ. of Munich, Munich, 2008.
[49] Graves, A., Mohamed, A.-R., and Hinton, G., "Speech Recognition with Deep Recurrent Neural Networks," *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, Inst. of Electrical and Electronics Engineers, New York, 2013, pp. 6645–6649.
[50] Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., Boyle, R., Cantin, P.-L., Chao, C., Clark, C., Coriell, J., Daley, M., Dau, M., Dean, J., Gelb, B., Ghaemmaghami, T. V., Gottipati, R., Gulland, W., Hagmann, R., Ho, C. R., Hogberg, D., Hu, J., Hundt, R., Hurt, D., Ibarz, J., Jaffey, A., Jaworski, A., Kaplan, A., Khaitan, H., Killebrew, D., Koch, A., Kumar, N., Lacy, S., Laudon, J., Law, J., Le, D., Leary, C., Liu, Z., Lucke, K., Lundin, A., MacKean, G., Maggiore, A., Mahony, M., Miller, K., Nagarajan, R., Narayanaswami, R., Ni, R., Nix, K., Norrie, T., Omernick, M., Penukonda, N., Phelps, A., Ross, J., Ross, M., Salek, A., Samadiani, E., Severn, C., Sizikov, G., Snelham, M., Souter, J., Steinberg, D., Swing, A., Tan, M., Thorson, G., Tian, B., Toma, H., Tuttle, E., Vasudevan, V., Walter, R., Wang, W., Wilcox, E., and Yoon, D. H., "In-Datacenter Performance Analysis of a Tensor Processing Unit," *SIGARCH Computer Architecture News*, Vol. 45, No. 2, 2017, p. 1–12.
https://doi.org/10.1145/3140659.3080246
[51] Andersson, J., Hjorth, M., Johansson, F., and Habinc, S., "LEON Processor Devices for Space Missions: First 20 Years of LEON in Space," *2017 6th International Conference on Space Mission Challenges for Information Technology (SMC-IT)*, IEEE Publ., Piscataway, NJ, 2017, pp. 136–141.
https://doi.org/10.1109/SMC-IT.2017.31
[52] Lopez, D., Llosa, J., Ayguade, E., and Valero, M., "Impact on Performance of Fused Multiply-Add Units in Aggressive VLIW Architectures," *Proceedings of the 1999 International Conference on Parallel Processing*, IEEE Publ., Piscataway, NJ, 1999, pp. 22–29.
[53] Asanovic, K., and Wawrzynek, J., *Vector Microprocessors*, Univ. of California, Berkeley, CA, 1998.
[54] Lee, S.-J., Park, S.-S., and Chung, K.-S., "Efficient SIMD Implementation for Accelerating Convolutional Neural Network," *Proceedings of the 4th*

International Conference on Communication and Information Processing, Assoc. for Computing Machinery, New York, 2018, pp. 174–179. https://doi.org/10.1145/3290420.3290444

[55] Flamand, E., Rossi, D., Conti, F., Loi, I., Pullini, A., Rotenberg, F., and Benini, L., "GAP-8: A RISC-V SoC for AI at the Edge of the IoT," *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, IEEE Publ., Piscataway, NJ, 2018, pp. 1–4. https://doi.org/10.1109/ASAP.2018.8445101.

[56] Peleg, A., and Weiser, U., "MMX Technology Extension to the Intel Architecture," *IEEE Micro*, Vol. 16, No. 4, 1996, pp. 42–50. https://doi.org/10.1109/40.526924

[57] Thakkur, S., and Huff, T., "Internet Streaming SIMD Extensions," *Computer*, Vol. 32, No. 12, 1999, pp. 26–34. https://doi.org/10.1109/2.809248

[58] Doolan, D. C., Tabirca, S., and Yang, L. T., "Mobile Parallel Computing," *2006 Fifth International Symposium on Parallel and Distributed Computing*, IEEE Publ., Piscataway, NJ, 2006, pp. 161–167. https://doi.org/10.1109/ISPDC.2006.33

[59] Gautschi, M., Schiavone, P. D., Traber, A., Loi, I., Pullini, A., Rossi, D., Flamand, E., Gürkaynak, F. K., and Benini, L., "Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 25, No. 10, 2017, pp. 2700–2713. https://doi.org/10.1109/TVLSI.2017.2654506

[60] Dabbelt, D., Schmidt, C., Love, E., Mao, H., Karandikar, S., and Asanovic, K., "Vector Processors for Energy-Efficient Embedded Systems," *Proceedings of the Third ACM International Workshop on Many-Core Embedded Systems*, Assoc. for Computing Machinery, New York, 2016, pp. 10–16. https://doi.org/10.1145/2934495.2934497

[61] Stephens, N., Biles, S., Boettcher, M., Eapen, J., Eyole, M., Gabrielli, G., Horsnell, M., Magklis, G., Martinez, A., Premillieu, N., Reid, A., Rico, A., and Walker, P., "The ARM Scalable Vector Extension," *IEEE Micro*, Vol. 37, No. 2, 2017, pp. 26–39. https://doi.org/10.1109/MM.2017.35

[62] Shimizu, T., "Post-K Supercomputer with Fujitsu's Original CPU, A64FX Powered by Arm ISA," 2018, https://www.fujitsu.com/global/Images/post-k_supercomputer_with_fujitsu%27s_original_cpu_a64fx_powered_by_arm_isa.pdf.

[63] Lee, Y., Ou, A., Schmidt, C., Karandikar, S., Mao, H., and Asanovic, K., "The Hwacha Microarchitecture Manual, Version 3.8.1," Electrical Engineering and Computer Sciences Dept., Univ. of California TR UCB/EECS-2015-263, Berkeley, CA, 2015, https://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-263.html.

[64] "RISC-V 'V' Vector Extension, Version 0.9," 2020, https://github.com/riscv/riscv-v-spec/releases/download/0.9/riscv-v-spec-0.9.pdf [retrieved 2 July 2020].

[65] Chen, C., Xiang, X., Liu, C., Shang, Y., Guo, R., Liu, D., Lu, Y., Hao, Z., Luo, J., Chen, Z., Li, C., Pu, Y., Meng, J., Yan, X., Xie, Y., and Qi, X., "Xuantie-910: A Commercial Multi-Core 12-Stage Pipeline Out-of-Order 64-Bit High Performance RISC-V Processor with Vector Extension : Industrial Product," *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, IEEE Publ., Piscataway, NJ, 2020, pp. 52–64. https://doi.org/10.1109/ISCA45697.2020.00016

[66] Louis, M. S., Azad, Z., Delshadtehrani, L., Gupta, S., Warden, P., Reddi, V. J., and Joshi, A., "Towards Deep learning Using TensorFlow Lite on RISC-V," *Third Workshop on Computer Architecture Research with RISC-V (CARRV)*, 2019, Paper 7, https://carrv.github.io/2019/papers/carrv2019_paper_7.pdf.

[67] Lee, Y., Waterman, A., Avizienis, R., Cook, H., Sun, C., Stojanović, V., and Asanović, K., "A 45 nm 1.3 GHz 16.7 Double-Precision GFLOPS/W RISC-V Processor with Vector Accelerators," *ESSCIRC 2014—40th European Solid State Circuits Conference (ESSCIRC)*, IEEE Publ., Piscataway, NJ, 2014, pp. 199–202. https://doi.org/10.1109/ESSCIRC.2014.6942056

[68] Mukherjee, S. S., Weaver, C., Emer, J., Reinhardt, S. K., and Austin, T., "A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor," *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36*, IEEE Publ., Piscataway, NJ, 2003, pp. 29–40. https://doi.org/10.1109/MICRO.2003.1253181

[69] Ebrahimi, M., Evans, A., Tahoori, M. B., Costenaro, E., Alexandrescu, D., Chandra, V., and Seyyedi, R., "Comprehensive Analysis of Sequential and Combinational Soft Errors in an Embedded Processor," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 34, No. 10, 2015, pp. 1586–1599. https://doi.org/10.1109/TCAD.2015.2422845

[70] Hubert, G., Artola, L., and Regis, D., "Impact of Scaling on the Soft Error Sensitivity of Bulk, FDSOI and FinFET Technologies due to Atmospheric Radiation," *Integration*, Vol. 50, June 2015, pp. 39–47. https://doi.org/10.1016/j.vlsi.2015.01.003

[71] Gaisler, J., "A Portable and Fault-Tolerant Microprocessor Based on the SPARC v8 Architecture," *Proceedings International Conference on Dependable Systems and Networks*, IEEE Publ., Piscataway, NJ, 2002, pp. 409–415. https://doi.org/10.1109/DSN.2002.1028926

[72] "OSCAR OBC," Airbus, 2018, https://www.airbus.com/content/dam/products-and-solutions/space/spacecraft-equipment/sce-datasheets/Publication-sce-oscar.pdf.

[73] Petit, S., David, J. P., Falguere, D., Duzellier, S., Inguimbert, C., Nuns, T., and Ecoffet, R., "Memories Response to MBU and Semi-Empirical Approach for SEE Rate Calculation," *IEEE Transactions on Nuclear Science*, Vol. 53, No. 4, 2006, pp. 1787–1793. https://doi.org/10.1109/TNS.2006.872153

[74] Samaras, A., Bezerra, F., Lorfevre, E., and Ecoffet, R., "CARMEN-2: In Flight Observation of Nondestructive Single Event Phenomena on Memories," *2011 12th European Conference on Radiation and Its Effects on Components and Systems*, IEEE Publ., Piscataway, NJ, 2011, pp. 839–848. https://doi.org/10.1109/RADECS.2011.6131314

[75] Bacchini, A., Furano, G., Rovatti, M., and Ottavi, M., "Total Ionizing Dose Effects on DRAM Data Retention Time," *IEEE Transactions on Nuclear Science*, Vol. 61, No. 6, 2014, pp. 3690–3693. https://doi.org/10.1109/TNS.2014.2365532

[76] Kumar, A., and Sawitzki, S., "High-Throughput and Low-Power Architectures for Reed Solomon Decoder," *Conference Record of the Thirty-Ninth Asilomar Conference on Signals, Systems and Computers*, IEEE Publ., Piscataway, NJ, 2005, pp. 990–994. https://doi.org/10.1109/ACSSC.2005.1599906

[77] Udipi, A. N., Muralimanohar, N., Chatterjee, N., Balasubramonian, R., Davis, A., and Jouppi, N. P., "Rethinking DRAM Design and Organization for Energy-Constrained Multi-Cores," *SIGARCH Computer Architecture News*, Vol. 38, No. 3, 2010, p. 175–186. https://doi.org/10.1145/1816038.1815983

[78] Hanho, L., "High-Speed VLSI Architecture for Parallel Reed-Solomon Decoder," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 11, No. 2, 2003, pp. 288–294. https://doi.org/10.1109/TVLSI.2003.810782

[79] Shayan, Y. R., and Le-Ngoc, T., "A Cellular Structure for a Versatile Reed-Solomon Decoder," *IEEE Transactions on Computers*, Vol. 46, No. 1, 1997, pp. 80–85. https://doi.org/10.1109/12.559805

[80] Li, G., Hari, S. K. S., Sullivan, M., Tsai, T., Pattabiraman, K., Emer, J., and Keckler, S. W., "Understanding Error Propagation in Deep Learning Neural Network (DNN) Accelerators and Applications," *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Assoc. for Computing Machinery, New York, 2017. https://doi.org/10.1145/3126908.3126964

[81] Zhang, Z., Huang, L., Huang, R., Xu, W., and Katz, D. S., "Quantifying the Impact of Memory Errors in Deep Learning," *2019 IEEE International Conference on Cluster Computing (CLUSTER)*, IEEE Publ., Piscataway, NJ, 2019, pp. 1–12. https://doi.org/10.1109/CLUSTER.2019.8890989

[82] Kosinski, B., and Dodson, K., "Key Attributes to Achieving >99.99 Satellite Availability," *2018 IEEE International Reliability Physics Symposium (IRPS)*, IEEE Publ., Piscataway, NJ, 2018, pp. 6A.3-1–6A.3-10. https://doi.org/10.1109/IRPS.2018.8353620

[83] Gee, J. D., and Smith, A. J., "Vector Processor Caches," Electrical Engineering and Computer Sciences Dept., Univ. of California, TR UCB/CSD-92-707, Berkeley, CA, Oct. 1992, http://www2.eecs.berkeley.edu/Pubs/TechRpts/1992/6251.html.

[84] Yang, Q., "Introducing a New Cache Design into Vector Computers," *IEEE Transactions on Computers*, Vol. 42, No. 12, 1993, pp. 1411–1424. https://doi.org/10.1109/12.260632

[85] "RISC-V 'V' Vector Extension, Version 0.8," 2019, https://github.com/riscv/riscv-v-spec/releases/download/0.8/riscv-v-spec-0.8.pdf [retrieved 5 Nov. 2020].

[86] Abtahi, T., Shea, C., Kulkarni, A., and Mohsenin, T., "Accelerating Convolutional Neural Network With FFT on Embedded Hardware," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 26, No. 9, 2018, pp. 1737–1749. https://doi.org/10.1109/TVLSI.2018.2825145

[87] Wang, S., Hu, J., and Ziavras, S. G., "On the Characterization of Data Cache Vulnerability in High-Performance Embedded Microprocessors," *2006 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation*, IEEE Publ., Piscataway, NJ, 2006, pp. 14–20. https://doi.org/10.1109/ICSAMOS.2006.300803

[88] Sadler, N. N., and Sorin, D. J., "Choosing an Error Protection Scheme for a Microprocessor's L1 Data Cache," *2006 International Conference on Computer Design*, IEEE Publ., Piscataway, NJ, 2006, pp. 499–505. https://doi.org/10.1109/ICCD.2006.4380862

[89] Fernández, M., Gioiosa, R., Quiñones, E., Fossati, L., Zulianello, M., and Cazorla, F. J., "Assessing the Suitability of the NGMP Multi-Core Processor in the Space Domain," *Proceedings of the Tenth ACM International Conference on Embedded Software*, Assoc. for Computing Machinery, New York, 2012, pp. 175–184. https://doi.org/10.1145/2380356.2380389

Z. Sunberg
Associate Editor