PARALLELIZING ON-BOARD DATA ANALYSIS APPLICATIONS FOR A DISTRIBUTED PROCESSING ARCHITECTURE

Patrick Kenny[1], Kurt Schwenk[2], Daniel Herschmann[2], Andreas Lund[1], Vishav Bansal[3], Zain Alabedin Haj Hammadeh[3], Andreas Gerndt[3,4], and Daniel Lüdtke[3]

[1]*German Aerospace Center (DLR), Institute for Software Technology, 82234 Weßling, Germany*
[2]*German Aerospace Center (DLR), Space Operations and Astronaut Training, 82234 Weßling, Germany*
[3]*German Aerospace Center (DLR), Institute for Software Technology, 38108 Braunschweig, Germany*
[4]*Center for Industrial Mathematics (ZeTeM), University of Bremen, 28359 Bremen, Germany*

ABSTRACT

The high data generation rates and real-time requirements of many modern satellite missions require increasing on-board computational performance. In this work we demonstrate the use of a data-flow-driven parallelization of an on-board application on a scalable, distributed, on-board computing architecture to achieve high computational performance for a neural-network based image classification application. The approach is based on the Scalable On-Board Computing for Space Avionics (ScOSA) architecture and the On-Board Data Analysis and Real-Time Information System (ODARIS) software. The analysis is performed on a lab setup of the ScOSA system consisting of three interconnected dual-core Xilinx Zynq processors running between one and six image processing tasks in parallel. Each processor executes up to two of these tasks. A centralized distribution task assigns each frame to one of the image processing tasks, allowing concurrent processing of multiple frames without requiring modification to the image processing algorithm. Using this setup, an increased image processing throughput was achieved by increasing the number of processing tasks. Using three processing tasks, a throughput 2.5 times as high as with a single processing task was achieved. The approach demonstrates the ability of the distributed on-board data processing approach to deliver a solution which provides scalability and high performance. Processing complete data packages concurrently, rather than splitting each package for parallel processing, makes this method feasible for any application where increasing package throughput is required and additional processors can be added to meet the demand.

Key words: distributed systems; scalability; concurrent execution; parallelization; data-flow.

1. INTRODUCTION

Modern satellite applications increasingly generate large amounts of raw data on board. The traditional method of downlinking data to perform the data analysis on the ground is poorly suited to such large data volumes, especially when results are required in real time. To meet this demand, on-board data analysis is an area of significant interest. By processing the raw data on board, only the relevant results need to be transmitted to ground, which can decrease the required data transfer rate by several orders of magnitude.

This work is based on the ScOSA project, an architecture for a distributed and scalable on-board computer combining multiple commercial-off-the-shelf and radiation-tolerant processors, together with a middleware to enable the practical use of such a computer by applications, developed at the German Aerospace Center (DLR). The work uses the ODARIS system, also developed at DLR, as a space-relevant application with high requirements for computational performance. ODARIS provides an image analysis and low-latency information system using neural-network-based image processing.

The focus of this paper is the increase in data processing throughput by processing multiple data packets simultaneously on different cores and different nodes. This coarse-grained parallelization approach provides good scalability, as processing nodes can be added as needed to provide the required computational power. Using the ScOSA platform, applications can take advantage of these extra nodes with minimal concern for the underlying distributed hardware structure. Another advantage of this approach is that it does not rely on an application-specific parallelization. Whereas parallelizing the processing of a single image by, for example, processing each line simultaneously requires detailed knowledge of the algorithm and access to the algorithm's source code, as well as sophisticated and algorithm-specific techniques, the approach presented here is possible even with black-box algorithms. It can be easily implemented without specialist knowledge of the algorithms and when source code is not available, as may be the case for functions from data-processing libraries.

A further advantage is that the application can run on a CPU with any architecture and operating system supported by ScOSA. While many applications can benefit from significant speed increases by implementations in hardware, for instance using FPGAs, such an implemen-

tation must be written in a hardware description language, which requires both detailed knowledge and a large time expenditure.

This paper aims to demonstrate these advantages of the ScOSA system and the coarse-grained parallelization, using a neural network-based object-identification algorithm running on the ODARIS platform. We operate three networked processors in a lab setup running up to six processing tasks in parallel, with an additional task to manage the distribution of images to available processing tasks. We achieve a data-processing throughput over 2.5 times as high as with a single task.

The remainder of the paper is structured as follows: Section 2 provides an overview of other efforts to provide high-performance on-board processing, and Section 3 describes the ScOSA and ODARIS platforms on which the system is based, as well as the specific implementation for this paper. Section 4 presents results from testing the system, and Section 5 presents the conclusions and plans for further development of the system.

2. RELATED WORK

Processing images from Earth-observation satellites is an area of high demand for on-board data analysis. Image-processing applications with real-time requirements range from forest fire and environmental monitoring (such as the German Aerospace Center's FireBIRD constellation [1], and [2]) to hyperspectral target identification for defense, security and civilian uses [3]. Artificial intelligence and on-board autonomy are also drivers of increased demand for on-board computational power [4].

One approach to improving data processing capability is through the use of on-board hardware. Field-programmable gate arrays (FPGAs) are a promising technology to achieve high performance with low power consumption. Qi et al. [5] propose an algorithm to allow implementation of pre-processing in hardware and use a combined FPGA and digital signal processor (DSP) to perform on-board data processing. A study by Kalomoiris et al. [6] also investigated FPGAs as a platform for on-board processing of convolutional neural networks (CNNs) and found favorable results compared to graphics processing units (GPUs). A distributed on-board processing system developed by Parache et al. [7] divides image processing into compression and transmission, and executes each function on a separate FPGA processor, with the two connected by a Controller Area Network (CAN) bus.

GPUs also offer potential for fast on-board processing via their inherent parallelism. Setoain et al. [8] investigated the use of GPUs for on-board processing of hyperspectral images and found significant promise, with the GPUs delivering high performance at low cost. However, Mahendra et al. [9] evaluated the use of GPUs as an alternative to central processing units (CPUs) and FPGAs and concluded that FPGAs are superior for on-board data processing due to their lower power consumption and comparable or superior performance.

The data-processing algorithms executing on board have been the target of research to develop more efficient and parallelizable designs. Du and Nekovei [10] propose modified algorithms for the processing of hyperspectral images in real time, using a small portion of the pixels to update the inverse data correlation matrix. Zhang et al. [11] use Cholesky decomposition along with a linear solving system to analyze incoming image data in a linewise fashion to achieve fast, real-time performance.

The performance advantage of commercial-off-the-shelf (COTS) components over radiation-hardened components makes them an attractive option for high-performance on-board data processing. However, their susceptibility to radiation effects, such as single-event effects, requires countermeasures in order to maintain the necessary reliability. The development of reliable, on-board computers based on COTS components remains an active area of research. Czajkowski et al. [12] used a combination of COTS microprocessors with time triple modular redundancy and radiation-hardened components to achieve high performance and radiation tolerance. The HiRel program [13] sponsored by the European Space Agency also aimed to develop a highly reliable on-board computer using COTS components, including a CPU based on the PowerPC architecture and an FPGA. A combination of software techniques and special hardware provided good resilience to single-event effects.

Distributed systems offer the potential to increase on-board processing power by combining the performance of several individual processors. Fayyaz and Vladimirova [14] propose a distributed, fault-tolerant on-board computer using on-the-fly reconfiguration and wireless networking to connect the processing nodes. Weiss et al. [15] have demonstrated the usefulness of a distributed embedded computer architecture for achieving robustness to failures for safety-critical applications in an automotive context. By dynamically remapping tasks to computing nodes in the event of a node failure, quality of service can be maintained longer without increasing hardware redundancy.

3. METHOD

This section describes the two base components of the system used in this work, ODARIS and ScOSA, as well as the component developed atop this base for the current case study.

3.1. Low-Latency Real-Time Communication System

The On-Board Data Analysis and Real-Time Information System (ODARIS) addresses the increasing demand of on-board data analysis and the amount of time required to make valuable and critical information available for end users [16]. The system provides three main features:

1. On-board analysis of sensor data (e.g. camera data).
2. An information service to:
   (a) process user requests about specific data.
   (b) automatically send push notifications to users when pre-defined events occur.
3. Low-latency communication with end users via a global satellite communication network.

To extend the capabilities of on-board data analysis and to improve the quality of results, current state-of-the-art machine learning algorithms are used. Furthermore, the aim of ODARIS is not only to transfer the processed data as telemetry via a ground station, but also to send them immediately to the ground by utilizing low-latency, low-bandwidth communication channels, provided by global satellite communication networks, such as Iridium or Globalstar [17, 18].

ODARIS is suitable for applications such as the detection of unregistered shipping, where real-time on-board image processing and low-latency communication are necessary in order to both detect objects of interest and inform end users on the ground quickly enough to allow action to be taken.

The software is derived from the former German Aerospace Center (DLR) project Autonomous Real-Time Detection of Moving Maritime Objects (AMARO) [19] which focused on ship detection via computer vision algorithms. The real-time communication was provided via an Iridium-based message service [20]. The AMARO system was successfully demonstrated on an airplane mission in 2018 [21].

The generic ODARIS system consists of several services to provide all of its features. The main components are:

1. Image analysis service: Processes incoming data on board and stores the results.
2. Query and push service: Provides information to end users either by a user request or as automated push notifications triggered by customizable, pre-defined events.
3. Real-time communication service: Interface to a global satellite network for real-time communication.
4. Telemetry and telecommand service: Interface for the telemetry and telecommunication data in a satellite mission.
5. System management service: Interface to ScOSA middleware for general commanding of ODARIS.

For this experiment the image analysis service of ODARIS is used within ScOSA to measure the computational time of an object detection algorithm. The TensorFlow library provides a suitable machine learning framework to run the inference for image processing [22]. The application is directly written in C++ to minimize resource consumption and uses the highly optimized TensorFlow Lite C++ API for the implementation. The data processing task receives images as a serialized byte array. Afterwards, the inference to detect objects is performed via a pre-trained machine learning algorithm using four processing threads. The focus here is on the area of convolutional neural networks (CNNs). The algorithm is stored as a model file in a TensorFlow Lite format. The results of the image processing mainly consist primarily of:

1. The type of the object detected in the image.
2. The confidence in the type of the detected object.
3. The computational time needed for the inference process.

The stored results can be further used by other ODARIS services on demand.

TensorFlow's first mobile computer vision model MobileNetV1 is used as the TensorFlow Lite model. This model is pre-trained for about 1000 common objects and is open-sourced by Google [23]. The network is classified as a CNN and uses an enhanced concept called depthwise separable convolution. Instead of processing a complete 3 x 3 convolutional layer, the network splits the convolution into a 3 x 3 depthwise convolution and a 1 x 1 pointwise convolution [24]. With this method the number of parameters will be reduced compared to regular convolutional layers resulting in increased performance for embedded devices. More details about the neural network itself can be found within the work of Howard et al. [25].

3.2. The Distributed On-Board Computer

The other main building block of this work is the Scalable On-Board Computing for Space Avionics (ScOSA) system, a distributed computing architecture developed at DLR [26]. The hardware design of ScOSA combines multiple COTS processors with radiation-hardened, space-qualified processors, to provide both high computational performance and high reliability. The processors are connected via a SpaceWire or Ethernet network. The COTS processors provide high computational

performance, and are referred to as high-performance nodes (HPNs), while the radiation-hardened processors, referred to as reliable computing nodes, provide robustness against radiation-induced failure.

To allow applications to effectively utilize the distributed system, the middleware portion of the ScOSA system [27] provides a framework, called the Tasking Framework [28], based on a data-flow paradigm. Applications are divided into *tasks* which are assigned to processing *nodes* and connected via *channels*. Tasks are typically triggered for execution when data is available on all input channels. A simple example of the task-channel model of the Tasking Framework is shown in Figure 1. This framework has several advantages over a more typical procedural programming paradigm: reliability is enhanced by allowing tasks to be easily reconfigured to execute on a different node (for example, in response to the failure of a node); it allows structured, convenient communication between tasks executing on heterogeneous nodes; and, it assists in parallelizing applications in a scalable way. This final benefit, scalable parallelization, is the focus of this paper.

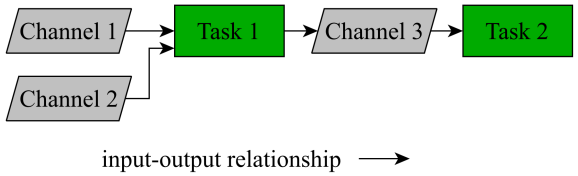

Figure 1. A simple example of the task-channel model of the Tasking Framework. Channels are persistent objects which store data and trigger the execution of tasks. Tasks execute processing operations but typically do not store data. Channels are connected to tasks as inputs or outputs. Tasks are usually configured to execute when data is available on all input channels, and will push data to outputs channels when complete. In this example, Channels 1 and 2 are inputs to Task 1. Channel 3 is an output of Task 1 and an input to Task 2.

The lab implementation of the ScOSA system used for the current experiment consists of three COTS high-performance nodes. Each HPN is based on a Trenz Electronic TE0720-03-1CFA with a Xilinx Zynq-7000 [29] system-on-chip (SoC). The HPNs are connected to each other via Ethernet and use the UDP/IP protocol for communication. All three SoCs use a Linux distribution created using Petalinux [30], a toolchain for Xilinx devices based on the Yocto software development kit.

3.3. Parallel Data Processing

In this experiment, we use the distributed extension of the Tasking Framework provided by ScOSA to implement a data-driven architecture to process multiple images in parallel. We create multiple instances of the *processing* task and configure these for execution on different nodes. The task-channel structure for the case of six processing tasks on three nodes is shown in Figure 2. Cases with fewer processing tasks follow the same pattern. The numbers in the corner of the processing tasks in the figure indicate the order in which these tasks are added as the number of tasks is increased from one to six. The ODARIS system, with its convolution of the neural network to identify objects in the images, is responsible for the majority of the entire system's CPU usage, and is the limiting factor in the image processing throughput. It is therefore the target of parallelization and distribution among processors in this experiment, and is executed within the processing tasks.

To achieve the distributed and parallel execution of the data processing, a new task, the *distribution* task, was created, which executes on Node 1. This task and its connections to other tasks directly related to the data processing are shown in Figure 2. It is executed with a regular period and reads unprocessed images as input.

The distribution task has an output channel for each processing task through which it transfers the unprocessed image and triggers the execution of the processing task (solid lines in Figure 2). Each processing task has an output channel which feeds back to the distribution task (dashed lines in Figure 2). These feedback channels are used to signal the availability of the processing task to receive and process new images. When the processing task receives an image from the distribution task, it uses the output to communicate that it is now busy and currently unable to receive further images. When it has completed the processing of the image, it uses this channel to indicate to the distribution task that it is once more available. This logic is shown in pseudo-code in Algorithm 1.

Algorithm 1: Data distribution task

```
1  execute()
2  image ← ReadNextImage()
3  foreach processing task, t_n do
4      chan_{feedback,n} ← GetFeedbackChanFromTask(t_n)
5      availability ← ReadFeedbackChan(chan_{feedback,n})
6      if availability is True then
7          chan_{data,n} ← GetDataChanToTask(t_n)
8          SendImageToChan(image, chan_{data,n})
9          return
10     end
11 end
```

When the distribution task is triggered and reads an image, it scans the feedback channels from the processing tasks for an available task, then sends the image to this task. If no processing tasks are available, the distribution task discards the image and will attempt to find a processor for the next image when triggered again.

4. EXPERIMENT

This section describes the method used to carry out a benchmark test to demonstrate the system described in Section 3 and presents the resulting data.

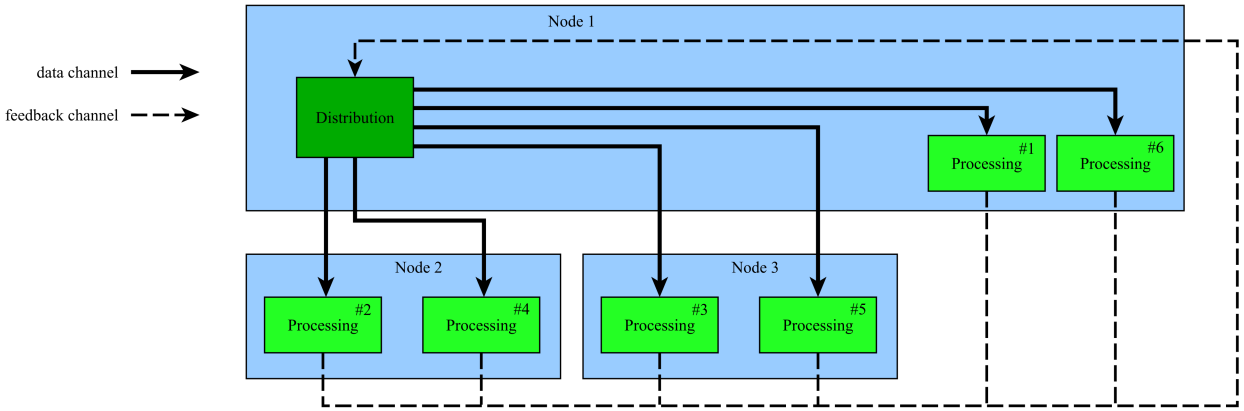

Figure 2. Structure of the tasks, channels and nodes involved in the parallel data processing for the case of six processing tasks spread across three nodes. The distribution task on Node 1 receives input data and assigns the data to a processing task, which may be on any node. Feedback channels from the processing tasks inform the distribution task about available processing capacity.

Table 1. Number of processing tasks allocated to each HPN

	Total number of processing tasks					
	1	2	3	4	5	6
HPN 1	1	1	1	1	1	2
HPN 2	0	1	1	2	2	2
HPN 3	0	0	1	1	2	2

4.1. Experimental Setup

As each of the HPNs used in this experiment has a dual-core processor, up to two processing tasks are assigned to each HPN. To investigate the effect of parallelization on the image throughput, the experiment was repeated with between one and six processing tasks divided between up to three HPNs. The allocation of processing tasks to HPNs is detailed in Table 1.

For the lab-based experiment, the data distribution task was configured to read pre-captured images with a resolution of 500 x 500 pixels cyclically from the flash memory file system of Node 1. Each pixel consists of red, green and blue channels, with 8 bits per channel, giving a data size of 6 Mbit per image.

In a satellite-based application, image acquisition rate, and thus the execution frequency of the distribution task, would be chosen based on mission requirements to provide the necessary quality of service. The data processing rate, and thus the number of processing nodes, would then be chosen to ensure the processing capacity exceeds the required production rate. For the benchmark test shown here, however, the aim is to demonstrate the maximum processing capacity of the system. We therefore choose a frequency for the distribution task which produces data faster than the processing tasks can process it to ensure that the processing tasks are not idle due to lack of input data. In this situation, when the distribution task executes but all processors are busy, the image is discarded and a new image is loaded when the distribution task executes one period later.

For each repetition of the experiment, the program was allowed to run for 120 seconds and the number of images processed by each task was recorded.

4.2. Results

The image processing throughput achieved in the experiments is shown in Figure 3. As the number of processing tasks is increased from one to three, an approximately linear increase in the system's image throughput is achieved, as each new processing task is executing alone on a new high-performance node. With three processing nodes, the throughput is over 2.5 times the throughput with a single task. Increasing the number of processing tasks beyond three results in only a small increase in the system's image throughput. With six processing tasks, the throughput is 2.8 times as high as with a single task. The diminishing returns beyond three tasks is due to the limited number of processing nodes. Processing tasks beyond the third are sharing a node with existing tasks and competing for CPU resources. As the CNN interference is multi-threaded, it is able to make use of both cores of a CPU even when running in a single task.

It is expected that if used with a single-threaded application, performance would increase substantially until the number of processing tasks equaled the number of CPU cores.

Figure 4 shows the average time required to process each

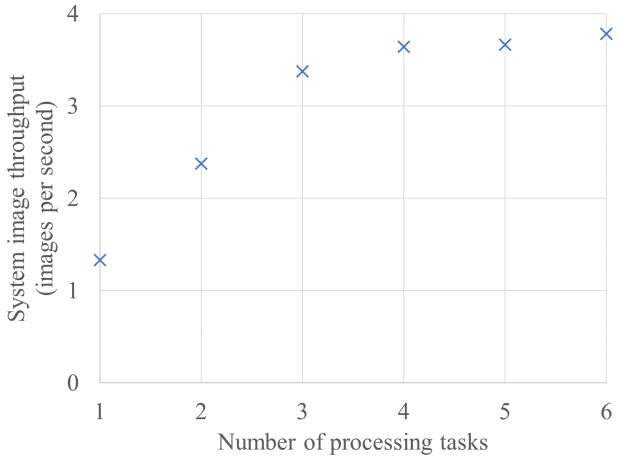

Figure 3. Image throughput of complete system with between one and six processing tasks. Utilizing all three nodes with three processing tasks results in a 2.5-fold gain in throughput compared to a single task. Running six tasks across three nodes results in a 2.8-fold gain compared to a single task.

image as the number of processing tasks is increased. As expected, this increases somewhat as more image processors are added, due to the overhead required to distribute the tasks and the network delay in transferring images between nodes. Where the number of processing tasks is no greater than the number of nodes, this increase is small. With three processing tasks, the average time to process one image is 19% higher than with a single node. Adding a second task to an existing node increases this time considerably, as the node's CPU is shared between the two concurrent tasks.

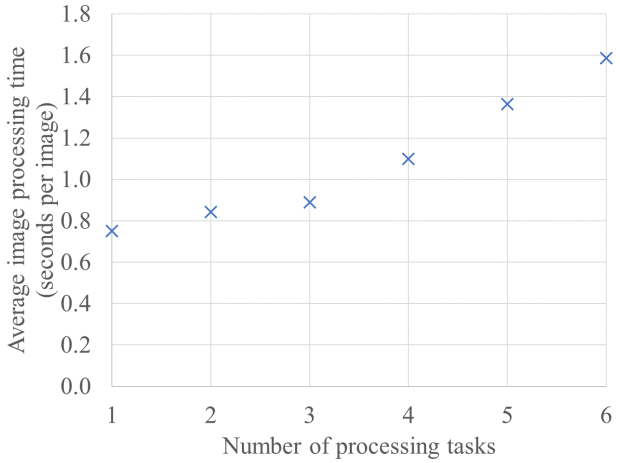

Figure 4. Average processing time per image for systems of between one and six processing tasks. The average processing time per image is increased by 19% when running three tasks across three nodes. Running six tasks across three nodes results in a 111% increase.

5. CONCLUSIONS AND FUTURE WORK

The distributed version of the Tasking Framework in the ScOSA system allows for parallelization of processing-intensive tasks without requiring in-depth knowledge of or modification to the underlying algorithms, and without requiring a large implementation effort.

This provides a scalable method to increase the throughput of on-board data processing tasks. The system's data throughput increases in approximately linear proportion to the number of processors available. The total data throughput achievable by such a system is likely to be limited by the amount of hardware, and the associated electrical and thermal considerations, before limitations inherent to the ScOSA system become restrictive. Where the tasks are multi-threaded and able to utilize all cores of the processor, adding additional tasks to the processor has minimal benefit.

The parallelization in the described system has little effect on the processing time for a single task, as long each processing task executes on a dedicated processor. While this approach is not suitable for situations where decreasing the latency of a single data packet's processing is of primary importance, the overhead required to achieve the increased throughput is achieved with little impact on the latency. Any applications for which the rate of processing is critical could potentially benefit from the technique described in this paper, allowing applications to process data with higher frequency, resolution or analytical detail.

5.1. Future Work

To continue the work described in this paper, we plan to deploy the ScOSA-ODARIS software to the European Space Agency's OPS-SAT. The dual-core processor on the OPS-SAT's on-board computer will be used to concurrently run two image processing tasks. This will provide experience operating the system in orbit and allow the use of images acquired by the on-board camera in real time.

The system is also planned to be deployed to the upcoming DLR compact satellite, which will contain a custom, distributed on-board computer designed as part of the ScOSA Flight Experiment project, described in [27].

For these flight experiments, the image processing neural network of ODARIS will be trained for an application-specific purpose, such as ship or cloud detection.

REFERENCES

[1] Olaf Frauenberger, Erik Borg, Winfried Halle, Eckehard Lorenz, Jens Richter, and Thomas Terzibaschian. The DLR FireBIRD Mission - A Technological Experiment for Operational Wildfire Monitoring. In M. JA. Marow et al., editors, *Trudy LII*

Chtenij, Posvjashhennyh Razrabotke Nauchnogo Nasledija I Razvitiju Idej K. Je. Ciolkovskogo. Sekcija "Problemy Raketnoj I Kosmicheskoj Tehniki", Kaluga, 19-21 Sentjabrja 2017 G., Kazan, Russia, 2018.

[2] Telmo Adão, Jonáš Hruška, Luís Pádua, José Bessa, Emanuel Peres, Raul Morais, and Joaquim João Sousa. Hyperspectral Imaging: A Review on UAV-Based Sensors, Data Processing and Applications for Agriculture and Forestry. *Remote Sensing*, 9(11):1110, November 2017. Number: 11 Publisher: Multidisciplinary Digital Publishing Institute.

[3] Michael T. Eismann, Joseph Meola, and Alan D. Stocker. Automated hyperspectral target detection and change detection from an airborne platform: Progress and challenges. In *2010 IEEE International Geoscience and Remote Sensing Symposium*, pages 4354–4357, July 2010. ISSN: 2153-7003.

[4] Jan-Gerd Meß, Frank Dannemann, and Fabian Greif. Techniques of Artificial Intelligence for Space Applications - A Survey. In *European Workshop on On-Board Data Processing (OBDP2019)*, Noordwijk, Netherlands, February 2019. European Space Agency.

[5] Baogui Qi, Hao Shi, Yin Zhuang, He Chen, and Liang Chen. On-Board, Real-Time Preprocessing System for Optical Remote-Sensing Imagery. *Sensors*, 18(5):1328, May 2018. Number: 5 Publisher: Multidisciplinary Digital Publishing Institute.

[6] Ioannis Kalomoiris, George Pitsis, Grigorios Tsagkatakis, Aggelos Ioannou, Christos Kozanitis, Apostolos Dollas, Panagiotis Tsakalides, and Manolis GH Katevenis. An Experimental Analysis of the Opportunities to Use Field Programmable Gate Array Multiprocessors for On-board Satellite Deep Learning Classification of Spectroscopic Observations from Future ESA Space Missions. In *European Workshop on On-Board Data Processing (OBDP2019)*, page 9, Noordwijk, Netherlands, 2019. European Space Agency.

[7] Yago Isasi Parache, Dr Pablo Ghiglino, and Nicolas Perzo. High Performance On-Board Image Processing using CANOpen for Earth Observation Satellites. In *European Workshop on On-Board Data Processing (OBDP2019)*, page 7, Noordwijk, Netherlands, 2019. European Space Agency.

[8] Javier Setoain, Manuel Prieto, Christian Tenllado, and Francisco Tirado. GPU for Parallel On-Board Hyperspectral Image Processing. *The International Journal of High Performance Computing Applications*, 22(4):424–437, November 2008. Publisher: SAGE Publications Ltd STM.

[9] H. N. Mahendra, S. Mallikarjunaswamy, G. K. Siddesh, M. Komala, and N. Sharmila. Evolution of real-time onboard processing and classification of remotely sensed data. *Indian Journal of Science and Technology*, 13(20):2010–2020, May 2020.

[10] Qian Du and Reza Nekovei. Fast real-time onboard processing of hyperspectral imagery for detection and classification. *Journal of Real-Time Image Processing*, 4(3):273–286, August 2009.

[11] Lifu Zhang, Bo Peng, Feizhou Zhang, Lizhe Wang, Hongming Zhang, Peng Zhang, and Qingxi Tong. Fast Real-Time Causal Linewise Progressive Hyperspectral Anomaly Detection via Cholesky Decomposition. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 10(10):4614–4629, October 2017.

[12] D.R. Czajkowski, M.P. Pagey, P.K. Samudrala, M. Goksel, and M.J. Viehman. Low Power, High-Speed Radiation Hardened Computer Flight Experiment. In *2005 IEEE Aerospace Conference*, pages 1–10, March 2005. ISSN: 1095-323X.

[13] S. Esposito, C. Albanese, M. Alderighi, F. Casini, L. Giganti, M. L. Esposti, C. Monteleone, and M. Violante. COTS-Based High-Performance Computing for Space Applications. *IEEE Transactions on Nuclear Science*, 62(6):2687–2694, December 2015.

[14] Muhammad Fayyaz and Tanya Vladimirova. Survey and future directions of fault-tolerant distributed computing on board spacecraft. *Advances in Space Research*, 58(11):2352–2375, December 2016.

[15] Philipp Weiss, Andreas Weichslgartner, Felix Reimann, and Sebastian Steinhorst. Fail-Operational Automotive Software Design Using Agent-Based Graceful Degradation. In *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1169–1174, March 2020. ISSN: 1558-1101.

[16] Kurt Schwenk and Daniel Herschmann. On-board data analysis and real-time information system. In *Deutscher Luft- und Raumfahrtkongress 2020*, Oktober 2020.

[17] Iridium Communications Inc. Official webpage, 2021. https://www.iridium.com/. Accessed 2021-05-31.

[18] Globalstar Inc. Official webpage, 2021. https://www.globalstar.com. Accessed 2021-05-31.

[19] Kurt Schwenk, Katharina A. M. Willburger, and Sebastian Pless. Amaro-autonomous real-time detection of moving maritime objects: introducing a flight experiment for an on-board ship detection system. In *Earth Resources and Environmental Remote Sensing/GIS Applications VIII*, volume 10428, page 1042808. SPIE Digital Library, Oktober 2017.

[20] Iridium Communications Inc. Iridium short burst data (SBD), 2021. https://www.iridium.com/services/details/iridium-sbd. Accessed 2021-05-31.

[21] Katharina Willburger, Kurt Schwenk, and Jörg Brauchle. AMARO - An on-board ship detection and real-time information system. *Sensors*, 20(5):1324, Februar 2020.

[22] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, Savannah, GA, November 2016. USENIX Association.

[23] Andrew G. Howard, Menglong Zhu. Mobilenets: Open-source models for efficient on-device vision. https://ai.googleblog.com/2017/06/mobilenets-open-source-models-for.html. Accessed 2021-05-31.

[24] Abhijeet Pujara. Mobilenet convolutional neural network machine learning algorithms. https://medium.com/analytics-vidhya/image-classification-with-mobilenet-cc6fbb2cd470. Accessed 2021-05-31.

[25] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

[26] Carl Johann Treudler, Heike Benninghoff, Kai Borchers, Bernhard Brunner, Jan Cremer, Michael Dumke, Thomas Gärtner, Kilian Johann Höflinger, Daniel Lüdtke, Ting Peng, Eicke-Alexander Risse, Kurt Schwenk, Martin Stelzer, Moritz Ulmer, Simon Vellas, and Karsten Westerdorff. ScOSA - Scalable On-Board Computing for Space Avionics. In *Proceedings of 69th International Astronautical Congress (IAC 2018)*, October 2018.

[27] Andreas Lund, Zain Alabedin Haj Hammadeh, Patrick Kenny, Vishav Vishav, Andrii Kovalov, Hannes Watolla, Andreas Gerndt, and Daniel Lüdtke. ScOSA system software: the reliable and scalable middleware for a heterogeneous and distributed on-board computer architecture. *CEAS Space Journal*, May 2021.

[28] Zain A. H. Hammadeh, Tobias Franz, Olaf Maibaum, Andreas Gerndt, and Daniel Lüdtke. Event-Driven Multithreading Execution Platform for Real-Time On-Board Software Systems. In *Proceedings of the 15th annual workshop on Operating Systems Platforms for Embedded Real-time Applications*, pages 29–34, Stuttgart, Germany, July 2019.

[29] Xilinx. *Zynq-7000 SoC Technical Reference Manual*, 2021.

[30] Xilinx. *PetaLinux Tools Documentation*, 2020.