# Spin-in of RISC-V Processors in Space Embedded Systems

Di Mascio, S.

**DOI**
10.4233/uuid:e515547e-62bc-4893-b299-87c1286b5d55

**Publication date**
2022

**Document Version**
Final published version

**Citation (APA)**
Di Mascio, S. (2022). *Spin-in of RISC-V Processors in Space Embedded Systems*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:e515547e-62bc-4893-b299-87c1286b5d55

**Important note**
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

# SPIN-IN OF RISC-V PROCESSORS IN SPACE EMBEDDED SYSTEMS

# SPIN-IN OF RISC-V PROCESSORS IN SPACE EMBEDDED SYSTEMS

## Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology
by the authority of the Rector Magnificus prof.dr.ir. T.H.J.J. van der Hagen
chair of the Board for Doctorates
to be defended publicly on
Monday 12 September 2022 at 15:00 o'clock

by

**Stefano DI MASCIO**

Master of Science in Electronics Engineering,
University of Rome "Tor Vergata", Rome, Italy,
born in Rome, Italy.

This dissertation has been approved by the promotors:

promotor: Prof. dr. E.K.A. Gill
copromotor: Dr. A. Menicucci

Composition of the doctoral committee:

| | |
|---|---|
| Rector Magnificus, | chairperson |
| Prof. dr. E.K.A. Gill , | Delft University of Technology, promotor |
| Dr. A. Menicucci, | Delft University of Technology, copromotor |

*Independent Members:*

| | |
|---|---|
| Prof. dr. O. Mutlu, | ETH Zürich |
| Prof. dr. H.P. Hofstee, | Delft University of Technology |
| Prof. dr. G. Setti, | Politecnico di Torino |
| Dr. S. Wong, | Delft University of Technology |
| J. Andersson (MSc), | Cobham Gaisler |

*Reserve Member:*

| | |
|---|---|
| Prof. dr. A.J. van der Veen, | Delft University of Technology |

This research was funded by Cobham Gaisler and the European Space Agency.




*Keywords:* Satellite Data Systems, Processors, Fault Tolerance, Space Systems, Artificial Intelligence, RISC-V, Application Specific Integrated Circuits, Small Satellites

*Printed by:* Ipskamp printing

*Front & Back:* Gustaf Wilhelm Palm, View of the Tuscolo theater (1852)



ISBN 978-94-6421-851-0

An electronic version of this dissertation is available at
http://repository.tudelft.nl/.

*I suppose it is tempting,*
*if the only tool you have is a hammer,*
*to treat everything as if it were a nail.*

Abraham Maslow, The Psychology of Science: A Reconnaissance (1966)

# CONTENTS

# SUMMARY

The usage of terrestrial processors in space applications is not straightforward, as processors in space face unique challenges due to the effects of the space environment, like ionizing radiation causing Single Event Effects (SEEs). In the nineties, the European Space Agency chose the Scalable Processor ARChitecture (SPARC) Instruction Set Architectures (ISA) for its processors, as it was the only solution available at that time providing both openness and available software support in terrestrial applications. Currently, a large part of the worldwide space community is using SPARC-based radiation-hardened (rad-hard) or radiation-tolerant (rad-tol) LEON processors in ongoing and planned missions, although SPARC processors virtually disappeared from terrestrial applications. Rad-hard and rad-tol processors for space applications typically lag more than a decade behind their commercial counterparts in terms of performance and the gap is widening every year. This is mainly due to the use of Rad-Hard-By-Design (RHBD) cells and older technology nodes. The larger vulnerability to SEEs of complex microarchitectures 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. The reason is that they are mainly employed for non-demanding control and housekeeping operations. Therefore, enabling demanding tasks, such as the execution of Artificial Intelligence (AI) algorithms in space embedded systems, requires a large leap in space-grade processors, especially because space data systems in satellites are typically power-constrained.

Recently, RISC-V, a novel free and open ISA, has risen in popularity in terrestrial applications, drawing the attention of several universities and companies. Given the similarity between SPARC and RISC-V, this dissertation starts by analyzing the advantages of using RISC-V in space applications. The openness of RISC-V already enabled a vast field of research activities for terrestrial applications, with many tools and models at different level of abstraction already available. Therefore, the space industry can spin-in developments from academia and industry, focusing efforts mainly on improvements concerning specific needs in space applications and without wasting efforts on other activities. In order to fully exploit modularity, the need of defining the types of processors required in space application was identified in this dissertation. The modularity of RISC-V was employed to identify several applications in space data systems and RISC-V processor profiles to address them. They were defined in this work by the ISA subset, Instruction-Level Parallelism (ILP), Data-Level Parallelism (DLP), Processor-Level Parallelism (PLP), reference implementation and expected performance. The processors profiles defined range from microcontrollers to general-purpose implementations to high-performance processors for AI. Finally, a roadmap to bring RISC-V IP cores for terrestrial applications to space level was defined, identifying the steps and models required.

After the thorough analysis of the state-of-the-art of RISC-V processors was completed, two different sets of activities were identified.

1. Increase the fault tolerance of microarchitectures of next-generation space processors: In order to do this, a literature study was carried out and a model to evaluate the vulnerability of microarchitectures to SEEs was developed. During the creation of this model, it was found that the most impacting factors on the SEE vulnerability are technology, environment and microarchitecture. The most vulnerable parts identified in both next-generation and state-of-the-art processors were caches, mainly because of the large area occupied. For this reason, addressing the upsets in caches reduces failure rate for state-of-the-art and next-generation processors by 96% and 93% respectively. Several redundancy techniques were introduced and their effect on microarchitectures similar to state-of-the art and next-generation space processors were evaluated with this model.

2. Increase performance of next-generation processors: Instead of focusing on the optimization of existing microarchitecures, the focus here was on identifying new types of applications for space processors for which performance can be dramatically improved by new microarchitectures, as state-of-the-art processors were not designed to execute them efficiently. An analysis of the requirements of processors to execute Deep Neural Networks (DNNs) was performed. Then, a study of compute-intensive workloads required for execution of DNNs was carried out, based on the case study of CloudNet, a DNN for cloud detection. Since it was found that the workload is largely composed by matrix operations, the RISC-V Vector Extension (RVVE) was proposed to speedup the execution of these matrix operations. The effect of employing vector instructions was investigated in detail using an ISA simulator and measuring the reduction in terms of number of instructions. In the case of matrix-multiplications between 128×128 matrices, vector processors are shown to provide a potential speedup up to 89×.

Finally, vector instructions were implemented in Very High Speed Integrated Circuit Hardware Description Language (VHDL) on a baseline NOEL-V processor from Cobham Gaisler and the obtained vector processor was employed to generate a hardware prototype on a Field Programmable Gate Array (FPGA) achieving Technology Readiness Level (TRL) 4. Furthermore, a detailed benchmarking with specific kernels including kernels from CloudNet was carried out, to show the benefits of implementing vector instructions. Several configurations were considered with high- and low-latency main memory, based respectively on a Synchronous Dynamic Random-Access Memory (SDRAM) and Synchronous Random-Access Memory (SRAM) memory. The presented prototype is capable of executing a DNN for cloud detection roughly 20× faster than the baseline processor.

# SAMENVATTING

Het gebruik van terrestrische processors in ruimtetoepassingen is niet eenvoudig, aangezien processors in de ruimte voor unieke uitdagingen staan vanwege de effecten van de ruimteomgeving, zoals ioniserende straling die *Single Event Effects* (SEE's) veroorzaakt. In de jaren negentig koos de *European Space Agency* voor de *Scalable Processor ARChitecture* (SPARC) *Instruction Set Architectures* (ISA) voor zijn processors, omdat dit de enige beschikbare oplossing was op dat moment die zowel openheid als diffuse softwareondersteuning in terrestrische toepassingen bood. Momenteel gebruikt een groot deel van de wereldwijde ruimtevaartgemeenschap SPARC-gebaseerde *radiation-hardened* (rad-hard) of *radiation-tolerant* (rad-tol) LEON-processors in alle lopende en geplande missies, hoewel SPARC-processors vrijwel verdwenen zijn uit terrestrische toepassingen. Rad-hard- en rad-tol-processors voor ruimtevaarttoepassingen lopen doorgaans meer dan tien jaar achter op hun commerciële tegenhangers in termen van prestaties en de kloof wordt elk jaar groter. Dit komt vooral door het gebruik van *Rad-Hard-By-Design* (RHBD)-cellen en grotere technology nodes. De grotere kwetsbaarheid voor SEE's van complexe microarchitecturen is niet de enige reden waarom eenvoudige microarchitecturen met een laag parallellisme nog steeds de overgrote meerderheid zijn van processors die in de ruimte worden gebruikt. In feite zijn de meeste taken die worden uitgevoerd door processors in ruimtegegevenssystemen niet-rekenintensieve werkbelastingen. De reden is dat ze voornamelijk worden ingezet voor niet-veeleisende controle- en housekeeping werkzaamheden. Om de uitvoering van *Artificial Intelligence* (AI)-algoritmen in *space embedded systems* mogelijk te maken, is daarom een grote sprong voorwaarts nodig in space-grade processors, vooral omdat ruimtegegevenssystemen in satellieten doorgaans power-constrained zijn.

Onlangs is RISC-V, een nieuwe *free* en *open* ISA, steeds populairder geworden in terrestrische toepassingen en heeft het de aandacht getrokken van verschillende universiteiten en bedrijven. Gezien de overeenkomst tussen SPARC en RISC-V, begint dit proefschrift met het analyseren van de voordelen van het gebruik van RISC-V in ruimtetoepassingen. De openheid van RISC-V maakte al een enorm veld van onderzoeksactiviteiten voor terrestrische toepassingen mogelijk, met veel tools en modellen op verschillende abstractieniveaus die al beschikbaar zijn. Daarom kan de ruimtevaartindustrie ontwikkelingen uit de academische wereld en de industrie hergebruiken, waarbij de inspanningen voornamelijk worden gericht op verbeteringen met betrekking tot specifieke behoeften op het gebied van ruimtetoepassingen en zonder inspanningen te verspillen aan andere activiteiten. Om de modulariteit volledig te benutten, werd in dit proefschrift de noodzaak geïdentificeerd van het definiëren van de typen processors die nodig zijn voor ruimtetoepassingen. De modulariteit van RISC-V werd gebruikt om verschillende toepassingen in ruimtegegevenssystemen en RISC-V-processorprofielen te identificeren om ze aan te pakken. Ze werden in dit werk gedefinieerd door de *ISA-subset*, *Instruction-Level Parallelism* (ILP), *Data-Level Parallelism* (DLP), *Processor-Level Paral-*

*lelism* (PLP), referentie-implementatie en verwachte prestaties. De gedefinieerde processorprofielen variëren van microcontrollers tot implementaties voor algemene doeleinden tot high-performance processors voor AI. Ten slotte werd een routekaart gedefinieerd om RISC-V IP-cores voor terrestrische toepassingen op ruimteniveau te brengen, waarbij de vereiste stappen en modellen werden geïdentificeerd.

Nadat de grondige analyse van de *state of the art* van RISC-V-processors was voltooid, werden twee verschillende sets van activiteiten geïdentificeerd.

1. Verhoog de *fault tolerance* van micro-architecturen van ruimteprocessors van de volgende generatie: Om dit te doen werd een literatuurstudie uitgevoerd en werd een model ontwikkeld om de kwetsbaarheid van microarchitecturen voor SEE's te evalueren. Tijdens het maken van dit model bleek dat de meest invloedrijke factoren op de SEE-kwetsbaarheid technologie, omgeving en microarchitectuur zijn. De meest kwetsbare onderdelen die werden geïdentificeerd in zowel de volgende generatie als de modernste processors waren caches, voornamelijk vanwege het grote bezette gebied. Om deze reden vermindert het aanpakken van de storingen in caches het uitvalpercentage voor state-of-the-art en next-generation processors met respectievelijk 96% en 93%. Verschillende redundantietechnieken werden geïntroduceerd en hun effect op micro-architecturen vergelijkbaar met *state-of-the-art* en *next-generation* ruimteprocessors geëvalueerd met dit model.

2. Verhoog de prestaties van de volgende generatie processors: In plaats van te focussen op de optimalisatie van bestaande micro-architecturen, lag de focus hier op het identificeren van nieuwe soorten toepassingen voor ruimteprocessors waarvoor de prestaties drastisch kunnen worden verbeterd door nieuwe microarchitecturen, aangezien state-of-the-art processors niet zijn ontworpen om ze efficiënt uit te voeren. Er is een analyse uitgevoerd van de vereisten van processors om *Deep Neural Networks* (DNN's) uit te voeren. Vervolgens werd een onderzoek uitgevoerd naar de rekenintensieve workloads die nodig zijn voor het uitvoeren van DNN's, op basis van de case study van CloudNet, een DNN voor clouddetectie. Omdat bleek dat de werklast grotendeels bestaat uit matrixbewerkingen, werd de *RISC-V Vector Extension* (RVVE) voorgesteld om de uitvoering van deze matrixbewerkingen te versnellen. Het effect van het gebruik van vectorinstructies is in detail onderzocht met behulp van een *ISA-simulator* en het meten van de vermindering van het aantal instructies. In het geval van matrixvermenigvuldigingen tussen 128×128 matrices, wordt aangetoond dat vectorprocessors een potentiële versnelling tot 89 keer bieden.

Ten slotte werden vectorinstructies geïmplementeerd in *Very High Speed Integrated Circuit Hardware Description Language* (VHDL) op een baseline NOEL-V-processor van Cobham Gaisler en de verkregen vectorprocessor die werd gebruikt om een hardwareprototype op *Field Programmable Gate Array* (FPGA) te genereren dat *Technology Readiness Level* (TRL) 4 bereikte. Verder is er een gedetailleerde benchmarking uitgevoerd met specifieke kernels, waaronder kernels van CloudNet, om de voordelen van het implementeren van vectorinstructies te laten zien. Er werden verschillende configuraties overwogen met hoofdgeheugen met hoge en lage latentie, respectievelijk gebaseerd op

een *Synchronous Dynamic Random-Access Memory* (SDRAM) en *Synchronous Random-Access Memory* (SRAM) geheugen. Het gepresenteerde prototype kan een DNN voor clouddetectie ongeveer 20x sneller uitvoeren dan de baseline processor.

# List of Abbreviations

| | |
|---|---|
| AC | Average Criticality |
| ACE | Architecturally Correct Execution |
| ADR | Active Debris Removal |
| AHB | AMBA High-performance Bus |
| AI | Artificial Intelligence |
| ALU | Arithmetic-Logic Unit |
| AMBA | Advanced Microcontroller Bus Architecture |
| AOCS | Attitude and Orbital Control System |
| ASIC | Application-Specific Integrated Circuit |
| AVF | Architectural Vulnerability Factor |
| AXI4 | Advanced eXtensible Interface 4 |
| BHT | Branch History Table |
| BLAS | Basic Linear Algebra Subroutines |
| BP | Branch Prediction |
| BTB | Branch Table Buffer |
| CC | Clock Cycle |
| CI | Cell Interleaving |
| CL | Criticality Level |
| CMOS | Complementary Metal–Oxide Semiconductor |
| CNN | Convolutional Neural Network |
| COTS | Commercial-Off-The-Shelf |
| CPI | Cycles Per Instruction |
| CSR | Control and Status Registers |
| CU | Constant Utilization |
| CVF | Cache Vulnerability Factor |
| CW | Constant Workload |
| DC | Data Cache |
| DDR2 | Double Data Rate 2 |
| DDR3 | Double Data Rate 3 |
| DDR4 | Double Data Rate 4 |
| DL | Deep Learning |
| DLP | Depending on the context:<br>Data-Level Parallelism<br>Data-Level Parallel |
| DMR | Double Modular Redundancy |
| DNN | Deep Neural Network |
| DP | Double Precision |
| DRAM | Dynamic Random-Access Memory |

| | |
|---|---|
| DSP | Digital Signal Processing |
| DUE | Detected Uncorrectable Error |
| ECC | Error Correcting Code |
| EDAC | Error Detection And Correction |
| EDC | Error Detecting Code |
| EXC | Exception stage |
| EXE | Execution stage |
| F | Fetch stage |
| FC | Fully Connected |
| FD-SOI | Fully-Depleted Silicon-On-Insulator |
| FE | Front End |
| FF | Flip-Flop |
| FFT | Fast Fourier Transform |
| FI | Fault Injection |
| FinFET | Fin Field-Effect Transistor |
| FMA | Refers to both (specified when referring to just one of them):<br>Fused Multiply-Accumulate<br>Fused Multiply-Add |
| FPGA | Field Programmable Gate Array |
| FPU | Floating Point Unit |
| FRF | Floating Register File |
| FT | Fault-Tolerant |
| GCR | Galactic Cosmic Rays |
| GE | Gate Equivalents |
| GEMM | GEneral Matrix Multiply |
| GEO | Geostationary Orbit |
| GP | General Purpose |
| GPR | General-Purpose Register |
| GPU | Graphics Processing Unit |
| GRU | Gated Recurrent Unit |
| HC | High Criticality |
| HDL | Hardware Description Language |
| HP | Half Precision |
| HPC | High-Performance Computing |
| I/O | Input/Output |
| IB | Instruction Buffer |
| IC | Instruction Cache |
| ID | Interleaving Distance |
| IF | Instruction Fetch |
| IFT | Inverse Fourier Transform |
| II | Instruction Issue |
| ILP | Instruction-Level Parallelism |
| IO | In-Order |
| IOD | In-Orbit Demonstration |
| IoT | Internet of Things |

| | |
|---|---|
| IP | Intellectual Property |
| IPC | Instructions Per (clock) Cycle |
| IPS | Instructions Per Second |
| IRF | Integer Register File |
| ISA | Instruction Set Architecture |
| ISS | Instruction Set Simulator |
| IU | Integer Unit |
| L1 | Level1 |
| L2C | Level2 Cache |
| LC | Low Criticality |
| LEO | Low Earth Orbit |
| LET | Linear Energy Transfer |
| LLC | Last-Level Cache |
| LSB | Least Significant Bit |
| LSTM | Long Short-Term Memory |
| LSU | Load and Store Unit |
| M/D | Multiplier and Divider |
| MAC | Multiply-Accumulate |
| MBU | Multiple Bit Upset |
| MC | ManyCore |
| MCon | Memory Controller |
| MCU | Multiple Cell Upset |
| MD | MBU Dominated |
| MEM | Memory stage |
| MESI | Modified Exclusive Shared Invalid |
| MIMD | Multiple Instruction Multiple Data |
| ML | Machine Learning |
| MLP | Memory Level Parallelism |
| MM | Mass Memory |
| MMU | Memory Management Unit |
| MPEG | Moving Picture Experts Group |
| MSB | Most Significant Bit |
| MTTE | Mean Time To Event |
| MTTF | Mean Time To Failure |
| NLP | Natural Language Processing |
| OBC | On-Board Computer |
| OBDM | On-Board Decision Making |
| OI | Operational Intensity |
| OoO | Out-of-Order |
| OS | Operating System |
| QoS | Quality of Service |
| RA | Register Access stage |
| Rad-hard | Radiation-hardened |
| Rad-tol | Radiation-tolerant |
| RF | (scalar) Register File |

| | |
|---|---|
| RHBD | Rad-Hard-By-Design |
| RNN | Recurrent Neural Network |
| ROB | ReOrder Buffer |
| RQ | Research Question |
| RR | Register Rename |
| RS | Reed-Solomon |
| RTL | Register Transfer Level |
| RTOS | Real-Time Operating System |
| RTU | Remote Terminal Unit |
| RVVE | RISC-V Vector Extension |
| SAA | South Atlantic Anomaly |
| SBF | Single Bit Flip |
| SBU | Single Bit Upset |
| SD | SET Dominated |
| SDR | Single Data Rate |
| SDRAM | Synchronous Dynamic Random-Access Memory |
| SECDED | Single Error Correction and Double Error Detection |
| SED | Single Error Detection |
| SEE | Single Event Effect |
| SEFI | Single Event Functional Interrupt |
| SEL | Single Event Latchup |
| SER | Soft Error Rate |
| SET | Single Event Transient |
| SEU | Single Event Upset |
| SIMD | Single Instruction Multiple Data |
| SMP | Symmetric MultiProcessing |
| SNR | Signal-to-Noise Ratio |
| SoC | System-on-Chip |
| SOI | Silicon-On-Insulator |
| SP | Single Precision |
| SPARC | Scalable Processor ARChitecture |
| SpW | SpaceWire |
| SRAM | Static Random-Access Memory |
| SSO | Sun-Synchronous Orbit |
| SVE | Scalable Vector Extension |
| T&C | Telemetry and Command |
| TCP/IP | Transmission Control Protocol/Internet Protocol |
| TID | Total Ionizing Dose |
| TLB | Translation Lookaside Buffer |
| TLP | Thread-Level Parallelism |
| TMR | Triple Modular Redundancy |
| TRL | Technology Readiness Level |
| UT | Unexpected Termination |
| VBN | Vision-Based Navigation |
| VC | Vector Cache |

| | |
|---|---|
| VHDL | Very High Speed Integrated Circuit Hardware Description Language |
| VLA | Vector-Length Agnostic |
| VLSU | Vector Load and Store Unit |
| VP | Vector Processor |
| VRF | Vector Register File |
| WB | Write-Back |
| WCET | Worst Case Execution Time |
| WT | Write-Through |

# 1

# INTRODUCTION

*Blott Sverige svenska krusbär har.*
*Only Sweden has Swedish gooseberries.*

Carl Jonas Love Almqvist, "Om svenska rim" (1838)

*State-of-the-art space-grade processors lag behind their commercial counterparts because of their niched-size market (meaning smaller design teams and investments), their need for specific design solutions (e.g. tolerance against radiation effects), long qualification times and the importance of flight heritage in space applications. As a result, state-of-the-art space-grade processors are based on outdated Instruction Set Architectures (ISAs) and simple microarchitectures. The recent availability of RISC-V, a free, open and modular ISA, ignited the development of an unprecedented amount of open-source implementations targeting terrestrial applications. Future space systems could benefit from many of these developments, provided that work is done to satisfy the specific needs of processors for space (spin-in), especially in terms of fault tolerance and Technology Readiness Level (TRL).*

Parts of this chapter have been published in [1] and in [2].

1

## 1.1. BACKGROUND

Artificial Intelligence (AI) has been identified in the last decades as one of the most promising enabling technologies of future space applications [3, 4]. Deploying AI on board satellites enables intelligent On-Board Decision Making (OBDM), which for instance can help to overcome typical bottlenecks of space systems like the downlink bandwidth. Link bandwidth is a problem especially for severely resource-limited spacecraft like small satellites (e.g. it can take hours to days to downlink an image to the ground) [5]. The capability of filtering out images that do not meet certain criteria (data reduction) can greatly increase the potential of CubeSats, a highly miniaturized standard based on a satellite form factor of 10×10×10 $cm^3$, as improvements in sensors generate increasingly larger data volumes and storage space on CubeSats is typically very limited [5]. Furthermore, many proposed future missions are supposed to produce enormous volumes of data. In order to enable them, [6] suggests that either significant communication advancements or data reduction techniques are needed. However, advances in both capabilities would provide systems designers with a larger design space where better trade-offs are possible for missions producing large volumes of data. OBDM can enable also swarms [7], Active Debris Removal (ADR) by Vision-Based Navigation (VBN) [8] and in the future even cost-effective asteroid mining [9]. Furthermore, the capabilities of future scientific missions can be improved by OBDM too. For instance, the introduction of increasing degrees of OBDM capabilities to enable autonomous navigation and autonomous interpretation of data in the Mars rovers decreased the need of sending data to be processed and interpreted on Earth. Thus, the rovers were able to explore more on the surface of the planet instead of waiting for commands, overcoming also latency of operations commanded from ground [10].

Decisions can be made according to several degrees of "intelligence". In a first phase, systems were "smart" because they were provided with fixed rules defined by experts that worked for a specific application within certain expected boundary conditions. This approach is typically referred to as Digital Signal Processing (DSP). As an example, [8] describes several "tailored" algorithms for VBN. Today terrestrial applications are in a new phase where systems employ statistical learning to adapt their behavior to unexpected changes. This approach is typically referred to as Machine Learning (ML). More specifically, Deep Neural Networks (DNNs) are booming in terrestrial applications, generating a sub-field of AI known as Deep Learning (DL). This is possible thanks to the availability of large data sets from 'big data' and the availability of Graphics Processing Unit (GPU)-based hardware that enables processing of large amounts data in a reasonable timescale [11]. However, the situation in space embedded systems is quite different, as will be explained next.

The usage of terrestrial processors in space applications is not straightforward, as processors in space face unique challenges due to the effects of the space environment, like ionizing radiation causing Single Event Effects (SEEs)[1] and Total Ionizing Dose (TID) effects[2]. Radiation-hardened (rad-hard) processors for space applications typically lag more than a decade behind their commercial counterparts in terms of performance and the gap is widening every year (e.g. the commercial PowerPC-750 roadmap showed a

[1] SEEs are events caused by one highly energetic particle [12].
[2] TID effects are effects due to the accumulation of absorbed ionizing dose [13]

performance improvement of ~ 2400× from the early 1990s to 2005 [14], whereas the rad-hard version was improved by only ~ 300× [8]). This was mainly due to lower frequencies and smaller caches, both due to older technology nodes (e.g. in [8] commercial processors were based on 28 nm technologies, while contemporary rad-hard processors on 65 nm technologies) and use of Rad-Hard-By-Design (RHBD) cells. Furthermore, processors used in space are typically slower than their terrestrial counterparts due to the long qualification process for space-grade components and to a risk-averse behavior of space industry that prefers components and technologies which already are flight-proven. As a result, state-of-the-art space-grade processors are typically one or two order of magnitude less performing than their Commercial-Off-The-Shelf (COTS) counterparts [8]. Therefore, enabling OBDM in space embedded systems requires a large leap in space-grade processors to deal with algorithmic complexity, especially because satellites are typically power-constrained [8]. The analysis of state-of-the-art space-grade processors carried out with a literature review in Sec. 1.4 is the body of knowledge that will be improved in this dissertation. Before this analysis of terrestrial processors, Sec. 1.2 introduces concepts required in order to analyze the microarchitecture of processors in terms of performance and Sec. 1.3 introduces concepts to analyze the fault tolerance of space-grade processors.

## 1.2. CONCEPTS OF COMPUTER ARCHITECTURE

Comparisons of different processors are typically based on metrics involving performance. In order to understand the reasons behind differences in performance, in this section a simple model describing the performance of processors for a certain software program is presented. The time required to execute a program, $T_{ex}$, is given by:

$$T_{ex} = \frac{NI}{f_{clk}} \cdot CPI, \tag{1.1}$$

where $NI$ is the number of instructions in the program, $f_{clk}$ the clock frequency, and $CPI$ is the average number of Cycles Per Instruction [15]. Sometimes performance is expressed in terms of Instructions Per Second (IPS), where $IPS = f_{clk}/CPI$, although this metric is incomplete as it does not take into account that different Instruction Set Architectures (ISAs) require different $NI$ to execute the same program. Sometimes performance is expressed instead in terms of Instructions Per (clock) Cycle (IPC), where $IPC = 1/CPI$, which has the same limitation as IPS, and it does not take into account the improvement in terms of performance due to an higher $f_{clk}$.

The rest of this section shows how this execution time can be reduced for a given processor, employing parallelism, speculation and caching. The elements introduced in this section will allow a comparison of state-of-the-art processors for space applications with state-of-the-art general-purpose processors for terrestrial applications in Sec. 1.4.

### 1.2.1. PARALLELISM

The execution time can be reduced by exploiting parallelism in several ways. In this case, the focus is typically on decreasing the average number of $CPI$, although increasing $f_{clk}$ or decreasing $NI$ is possible too.

1

### INSTRUCTION-LEVEL PARALLELISM

One of the main tools to speedup the performance of a processor is to increase its level of Instruction-Level Parallelism (ILP), i.e. the number of instructions executed simultaneously. There are two main approaches:

- Pipelined execution: multiple instructions are overlapped in time, each of them in a different pipeline stage. Beside gains from overlapping execution of different instructions, another advantage of pipelining is the increase of the maximum allowed frequency, as the logic is broken down in shorter paths [16]:

$$f_{clk} = \frac{1}{(T/k) + T_S}, \tag{1.2}$$

  where $T$ is the propagation time trough the logic before pipelining, $k$ the number of pipeline stages and $T_S$ is the overhead in terms of propagation time introduced by the inserted sequential elements. Assuming that $T_S << T/k$, the improvement in frequency is linear with the number of stages. However, the sub-operations of a given instruction are not uniform and instructions are dependent from each other (inter-instruction dependency). Non-uniform sub-operations require balancing of the logic in the stages of the pipeline, as the maximum frequency will be determined by the slowest combinational path in the pipeline with a propagation time of $T_{max}$, where $T/k$ is replaced by $T_{max}$ in Eq. 1.2. To efficiently handle inter-instruction dependencies an increase of design complexity is required. On the other hand, a simple solution for inter-instruction dependency is stalling the pipeline until the dependencies are solved, causing large penalties in terms of performance [17] because of an increased CPI compared to an ideal pipeline without inter-instruction dependency. Inter-instruction dependency can be classified in two types:

  1. Data hazard: one of the instructions in the pipeline needs data from an instruction in later stages of the pipeline but that result isn't already available in the registers. A typical example is a load followed by an operation on the loaded value. In this case the loaded data can be directly forwarded as soon as they are ready from the previous instruction, thus minimizing the stall [15].
  2. Control (or branch) hazard: when a branch instruction is fetched, it takes some Clock Cycles (CCs) before knowing the target address and whether the branch is taken or not. In this case speculation can increase performance (Sec. 1.2.2).

  Another argument against pursuing an increase of performance by increasing $f_{clk}$ with a very deep pipeline is that an enhancement in a subsystem of the processor provides a limited improvement to the performance of the processor as a whole, as that subsystem is used only a limited amount of time. This is expressed by the Amdahl's Law [15]:

$$\textit{Improved Execution Time} = \frac{\textit{Affected Execution Time}}{\textit{Improvement}} + \textit{Unaffected Execution Time}. \tag{1.3}$$

For instance, memories are typically slower than processors. Assuming that reading and writing to the memory takes 60% of the time and improving the clock frequency of the processor by 10×, the execution time will decrease only by 36% [17].

- Multiple issue: the lower boundary for CPI of a single-issue processor is one. Multiple-issue processors deal with multiple instructions per stage, so that CPI can be less than one. There are two approaches for this:

  1. Static: the compiler analyses the software and packages instructions to execute simultaneously, when possible. This solution requires the pipeline (or at least parts of it) to be replicated. The multiple instructions launched during the same CC can be seen as one large instruction with multiple operations in parallel. For this reason, the processors that employ this approach are called Very Long Instruction Word (VLIW) processors [15].
  2. Dynamic (superscalar): in this case the processor decides during execution whether zero, one, or more instructions can be issued at a given CC [15]. A processor that can execute up to n instructions is usually called a n-way processor [15].

  To utilize more efficiently the ILP of superscalar processors, sometimes dynamic pipeline scheduling is employed, i.e. using Out-of-Order (OoO) pipelines. Dynamic pipeline scheduling chooses which instructions to execute next, reordering them to minimize stalls [15].

#### DATA-LEVEL PARALLELISM

A processor with Data-Level Parallelism (DLP) can execute an instruction on more elements of an array. Therefore, a considerable improvement in terms of performance can be obtained, as Data-Level Parallel (DLP) instructions reduce the number of instructions *NI* in a program by dealing with larger chunks of data compared to non-DLP instructions. This improvement is larger for programs where many instructions operate on regular data structures, which can be treated as vectors. However, when implementing DLP the challenge is to keep the penalty on *CPI* and $f_{clk}$ low. Two different approaches can be identified for DLP [18]:

1. Packed-Single Instruction Multiple Data (SIMD) DLP: a specific instruction is added to the ISA to exploit DLP provided by the hardware. The width is encoded in the assembly instruction. As a consequence, in order to increase performance and assure backward compatibility, also the number of instructions has to increase.

2. Vector DLP: this type of implementation can be seen as a more flexible version of packed-SIMD thanks to its time-multiplexed and Vector-Length Agnostic (VLA) approach. The VLA approach implies that the software is oblivious to the hardware vector length of a specific implementation and the same code executes using the largest parallelism possible on each platform [18–20]. This approach greatly simplifies software, especially because the width of the vectors is flexible and the ISA is independent from the level of parallelism provided by the platform.

As noted in [18], even if SIMD solutions have been successful on the consumer market, they are a suboptimal choice for an ISA. As a matter of fact, SIMD processors are usually

implemented replicating the Arithmetic-Logic Unit (ALU) and providing new opcodes[3] to the ISA in order to exploit this parallelism. This makes ISA instructions dependent on the implementation-specific degree of parallelism and new instructions must be added when a new level of DLP is required. When the architects of a packed-SIMD ISA 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 [18]. Furthermore, application code compiled for previous versions cannot automatically leverage the widened vectors of new implementations. At the same time, code compiled for wider packed-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 fringe elements of strip mine loops [18].

### THREAD-LEVEL PARALLELISM

Thread-Level Parallelism (TLP), also known as hardware multithreading, allows multiple threads[4] to share the functional units of a single processing core, switching thread when the running one is stalled. In order to do this, the processor must store the independent state of each thread. Hardware multithreading can be implemented in several ways:

- Coarse-grained: one thread runs until it is blocked by an event that creates a long latency stall (e.g. a cache miss). Hardware support for this type of multithreading is meant to allow quick switching between the threads, for instance storing the program visible state (e.g. general purpose registers and program counter) in the processing core [21].

- Fine-grained: the processor checks every cycle if the current thread is stalled or not. If stalled, a hardware scheduler will change execution to another thread that is ready to run [21]. Simpler fine-grained implementations, like the one described in [22], change thread every CC independently if the current one is stalled or not [23].

- Simultaneous: in this case a superscalar processor can issue instructions from either the same thread or from different threads [23]. The hardware thread scheduler has to chose the best instruction in order to maximize the utilization of the resources [21].

While coarse-grained multithreading is suited to avoid stalls due to a cache miss, fine-grained and simultaneous multithreading can effectively circumvent penalties due to control and data hazards [21] and can avoid the need for branch prediction. In [22] several fine-grained processing cores with different numbers of pipeline stages and numbers of hardware (HW) threads are proposed. The data in [22] shows the average number of IPC as a metric for the utilization of the pipeline for several versions of the same fine-grained implementation depending on the number of the software threads employed. It shows that even the simplest form of fine-grained implementation achieves full utilization (in this case $IPC = 1$) of its single-issue pipeline when the number of threads is

[3] Abbreviated form of "operational code", i.e. a portion of machine language that specifies an operation to be performed.

[4] From a software perspective, a thread of execution is the smallest sequence of programmed instructions that can be managed independently by a scheduler, which is typically a component of the operating system. A thread includes the program counter, the register state, and the stack [15].

greater than or equal to the number of pipeline stages. When the number of threads is smaller than the number of pipeline stages, the pipeline is underutilized.

#### PROCESSOR-LEVEL PARALLELISM (PLP)

In the last decade Moore's Law made it possible to include more processing cores in the same Application-Specific Integrated Circuit (ASIC) [24]. The most common approach to handle this parallelism is to provide a single physical address space that all the cores can share and to employ an Operating System (OS) scheduler to assign a thread to each core. This approach is called Symmetric MultiProcessing (SMP) [25]. Each processor works internally with virtual addresses that need to be translated to physical addresses. Processors communicate through shared variables in memory, with all processors capable of accessing any memory location via loads and stores [15].

General-purpose processors employ a Memory Management Unit (MMU) to translate page-based virtual addresses to physical addresses. The address table necessary for such translation is contained in the main memory and accessing it for each address is not an optimal solution. For this reason, Translation Lookaside Buffers (TLBs) are typically employed to reduce the average time per address translation in a similar manner as caches are employed to decrease the average access time to data and instructions in main memory (see Sec. 1.2.3).

SMP processors typically provide up to eight cores, as they are limited by the sequential model of the software and the use of shared resources (memories and interconnects). The impact of sequential tasks can be described applying Amdahl's Law (Eq. 1.3) to the parallelization problem [26], dividing a general problem in a parallelizable part ($p$) and a sequential part ($s$). The time required to execute the algorithm on a single core can be written as $T_{ex,1} = (s+p)\,T_{ex,1}$, where $s+p=1$. Assuming that the size of the problem remains constant with the increase of PLP, with $n$ processing cores the execution time becomes $T_{ex,n} = (s+p/n)\,T_{ex,1}$. Therefore, the speedup $S$ due to the use of $n$ cores is:

$$S = \frac{1}{s+p/n}. \tag{1.4}$$

In [26], it can be seen how, for an infinite amount of parallel processors, this leads to the theoretical maximum speedup of $1/s$. This means that, under the assumption that the size of the problem remain constant, the speedup of a program using multiple cores in parallel is limited by the time needed for the sequential fraction of the program. In an SMP platform this means that the capability of the OS to parallelize the workload is the final bottleneck. Therefore, increasing the number of processors provides only marginal improvements from a certain number of processors onward, independently from how efficiently resource sharing problems (e.g. access to shared memories) are solved.

Gustafson in [26] objected the assumption that the size of the problem remains constant when PLP is increased. Instead, he assumed that it is the run-time that remains constant, thus increasing the amount of calculations included in the problem (assuming that the workload can scale with the number of processors). Assuming that the serial part remains constant and that the parallel part can be parallelized over the $n$ processing cores, then $T^n_{ex,n} = (s'+p')\,T^n_{ex,n}$ for a parallel system (also here $s'+p'=1$). The time required by a single processor executing the workload scaled by $n$ is $T^n_{ex,1} = (s'+p'n)\,T^n_{ex,n}$.

In this case the execution time of the sequential part of the software becomes more and more negligible when the number of processors is increased, giving a linear increase with the number of processing cores, as shown below:

$$S = s' + p'\ n. \tag{1.5}$$

Following this paradigm, several manycore[5] processors, providing higher degrees of PLP, have been released (for instance with 64 cores [27]). These processors typically employ replicas of simple processors, and to parallelize the workload, each of them executes the same instruction on different data. This paradigm is often referred to as Multiple Instruction Multiple Data (MIMD) [18].

### 1.2.2. SPECULATION

Another way to decrease CPI is to use speculative execution (speculation). Speculation consists in predicting the outcome of an operation and start working assuming that the guess is right. As soon as the actual result is available, it must be checked if the prediction was right. If that was not the case, then the effect of the instructions that were executed speculatively must be reversed. For this reason, the implementation of speculation adds complexity [15]. Speculation is typically employed when a conditional branch instruction creates a control hazard in a deep and wide pipeline. For instance, in ARM Cortex-A8 (a dual-issue 14 stages processor where the branch address is calculated at the 13th stage, hence *number of issues* = 2 and *stalling penalty* = 13 CCs) if the pipeline is stalled while waiting for the branch address, then the number of potentially wasted cycles are: *number of issues* × *stalling penalty* (that in this case gives 26 wasted CCs per branch). Considering a branch predictor which predicts correctly 80% of the times in average, only an average 5.2 CCs per branch are wasted.

Data from [28] show that control instructions have low impact on parallel implementations of algorithms (e.g. making up only around 2% of the executed instructions), while they have a considerable impact on sequential algorithms (e.g. making up from 16% to 25% of the executed instructions). Branch prediction is typically transparent to software (except for variable execution times, so it can be a problem for time-determinism), as typically it is fully addressed at hardware level. Two approaches are possible:

1. Static branch prediction: the outcome of the branch is predicted before program execution. This approach is based on prediction heuristics from typical software [29] and achieves low efficiency, as it does not take into account the result of such predictions to improve them during software execution. The most popular example is the *always-taken* predictor. This technique can help with prediction of conditional branches in loops, which are more likely to be taken than not (usually more than one loop iteration is executed).

2. Dynamic branch prediction: these predictors achieve higher efficiency by learning on past behavior and exploiting temporal and/or spatial correlation. The typical

[5] Eight cores can be considered as the boundary between multicore and manycore (typically still considered multicore).

solution is a Branch History Table (BHT), a small memory indexed by the lower portion of the address of the branch instruction. The memory contains a bit that indicates whether the branch was recently taken or not [15].

Branch prediction speculates on whether a conditional branch is taken or not, but cannot speculate on the address of the target. For this reason, other forms of prediction can be employed to avoid stallings due to the calculation of the target address. The two most popular approaches, often used simultaneously, are:

1. Branch Table Buffer (BTB): a memory indexed with the lowest bits of the address of a branch instruction and holding the destination instruction address of the last time a branch instruction with the same lowest bits of the address was resolved. BTBs work well in most cases but perform poorly at returns from functions, because a function can be called from several positions within a program [15].

2. Subroutine return stack: an effective way of predicting the branch target when returning from a subroutine is to push in a stack the call address when a function call is executed and pop the return address when the return instruction is decoded [15].

### 1.2.3. MEMORY SUBSYSTEM

A corollary of Amdahl's law (Eq. 1.3) is the feasibility of improving performance by making the most common cases faster (i.e. a large part of the execution time will be affected by the improvement). The most striking example of this approach are cache memories. Caches store recently accessed data (temporal locality) and other contiguous memory locations (spatial locality), to reduce as much as possible the time spent accessing the main memory [15]. A block of data read from main memory is placed in a certain contiguous set of locations of the cache according to its associativity. The most common case is that of set-associative caches, i.e. there are a fixed number of locations where each block can be placed [15]. A set-associative cache with $n$ possible locations for a block is called an n-way set-associative cache [15]. Increasing the number of ways decreases the probability of a block to be evicted, hence reducing cache misses, but on the other hand complicates the design, as more sets have to be inspected to check whether the data is present in the cache.

Memories in processors are organized in hierarchies, as shown in Fig. 1.1. A typical memory hierarchy has Register Files (RFs) in the core, the fastest and smallest memories. At the interface between the core and the interconnect, there are Level 1 (L1) caches, divided in Instruction Caches (ICs) and Data Caches (DCs). L1 caches are considerably larger than RFs and typically require a couple of CCs to access them. Between the cores and the main memory, even larger and slower levels of cache (typically shared between the cores and other masters) are placed to achieve the desired access time/miss penalty trade-off. It should be noted that caches and layered memory architectures increase performance for the average case, while access times for the worst case get longer and time-determinism is penalized. In fact, when a cache does not contain the data, it has to fetch it from higher levels, recursively up to the main memory.

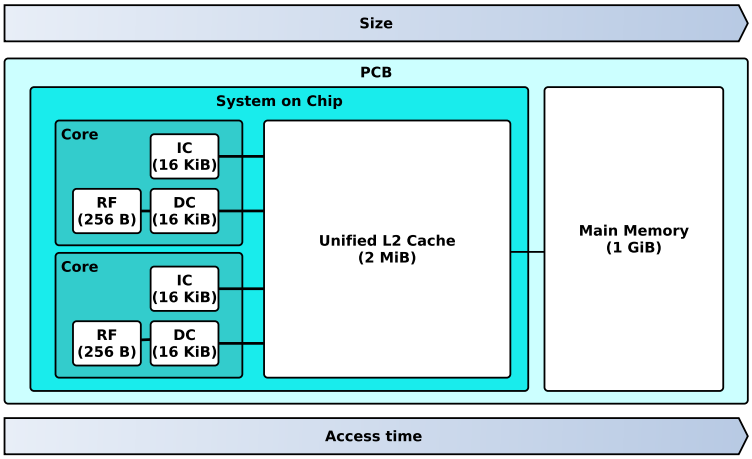


Figure 1.1: Typical memory hierarchy and memory sizes for a dual-core general-purpose processor. A similar memory hierarchy can be found in the GR740 processor [30].

#### CACHE COHERENCE

The increasing number of processing cores and the limited increase in memory bandwidth makes the memory bandwidth the bottleneck of a multicore system [31]. In a shared-memory multicore processor, data can be stored in several locations simultaneously, e.g. one copy might be residing in the main memory while other copies reside in several cache memories. When one copy of the data is modified by a processing core[6], then the copies in other caches have to be updated or invalidated. When a processing core writes data to its local cache, two different strategies can be employed [32]:

1. Write-back: the main memory is written back only when the cache line is flushed [32]. This can result in discrepancies when the same data is updated in a different way by two different processors. This strategy result in less use of bandwidth but requires more advanced cache coherence protocols.

2. Write-through: when data is written in a local cache it is also written in the main memory [32]. This simplifies the coherence issue, although a cache coherence protocol is still needed to invalidate (write-invalidate) or update (write-update) other private caches.

### 1.2.4. MEASURING PERFORMANCE

In principle, the software employed to select the best processor for a certain application should be a representative portion of the flight software. To have less-specific comparisons (and available before coding the flight software) between processors, they are typically compared using benchmarks, i.e. software intended to emulate the workload for a given type of application. The choice of the benchmark is critical, as it can unfairly favor

[6] Although the focus in this work is on processors, in general this applies also to other type of masters (e.g. interface controllers).

one implementation over the other. The most common benchmark for embedded processors is CoreMark [33], while for processors targeting personal computers the SPEC suite is more popular [34].

Historically, performance of embedded processors were assessed measuring the time required to execute part of a "synthetic" program called Dhrystone [35]. In this case performance are typically expressed in "Dhrystone MIPS" (DMIPS)[7], i.e. how many times the execution is faster compared to a reference processor (VAX 11/780) [36]. However, this program is so short that can fit in the L1 caches of many state-of-the-art processors, thus exercising only a small portion of the core infrastructure (i.e. integer unit). Furthermore, large portions of Dhrystone can be optimized by the compiler and library calls are made within the timed portion of Dhrystone. Since the library code is not part of the benchmark, it is difficult to compare results because different libraries can be used on different implementations. Using CoreMark instead of Dhrystone addresses these issues. For instance, CoreMark ensures that the compilers can not precompute the results, as every operation is based on a value that is not available at compile time. Furthermore, no library calls are made within the timed part.

The timed portion of CoreMark can be divided in three parts [33]:

1. Operations (e.g. reverse, search or sort) on linked lists[8], based on the values of the data items in the list. Including this type of workload measures how fast the processor deals with pointers and non-serial memory access patterns.

2. Operations on an input matrix (e.g multiplication with a constant, a vector, or another matrix). Furthermore, also operations on subsets of bits from each matrix item are carried out. Matrix operations can be efficiently sped up by DLP (even if they are relatively small, i.e. 12×12 [38]) and are based on (nested) loops.

3. Operations on strings (array of characters), with the goal of testing an input string to detect whether it is a number or not. Including this type of workload measures the performance of the processors when dealing with *irregular* conditional branches (i.e. different from loops).

Furthermore, the correctness of most operations is checked with a 16-bit Cyclic Redundancy Check (CRC). CRC is also included in the time measurement, as it is a common operation in software [33].

The score obtained after running CoreMark (usually for a large number of iterations) is the number of times CoreMark is executed per second. Often the CoreMark score is normalized to MHz (CoreMark/MHz), removing the dependence on clock frequency (hence on technology). However, measuring the performance in terms of CoreMark/MHz tends to penalize improvements due to the increase of the number of pipeline stages, because they increase performance mainly by increasing the maximum clock frequency (while they can even penalize the processor in terms of IPC).

[7] Although the name can cause misunderstandings, DMIPS is a relative measure of $T_{ex}$ and it is not the number of MIPS measured when executing Dhrystone.

[8] Linked lists are dynamic data structures. Each element of the linked list contains an element to the next element plus data. These type of constructs are used when the number of elements to be represented are not known at compile time [37].

1

## 1.3. DEPENDABILITY OF PROCESSORS FOR SPACE APPLICATIONS

Processors for critical applications are required to satisfy certain requirement in terms of dependability, i.e. "the ability to deliver service that can justifiably be trusted" [39]. Although in principle this definition also includes the effects of malicious threats (integrity, confidentiality), in space the focus is still on the effects of non-malicious faults, mainly due to radiation. However, regardless of the specific threats due to the space environment, processors in space have to be first of all robust against faults common to processors in terrestrial applications. For instance, simulations for a 32-nm ASIC technology show that the data propagation delay of Flip-Flops (FFs) increases less than 5% in 5 years of stress conditions due to aging [40]. This can be taken into account during design by applying larger margins on the maximum allowed frequency.

### 1.3.1. FAULTS DUE TO RADIATION

Changes in the charge stored in nodes due to particle strikes are typical faults in space processors, and they are called soft errors as they can be removed simply overwriting them with the correct value [41]. This is not the case for hard errors [42], where the distinction between fault (e.g. defective gate) and error (e.g. wrong result of a calculation) is needed for correct recovery (e.g. to replace a defective unit with a spare unit).

Despite the possibility of hard errors, soft errors due to radiation typically dominate the failure rate of processors already in terrestrial environments. In [43] the ratio of soft errors to hard errors for Static Random-Access Memory (SRAM) arrays in processors ranges from 77 to 735, and in [44] 99.36% of the errors in a SRAM array are soft errors while 0.64% are hard errors. Soft errors in space are even more predominant, as in this case charged particle strikes are more common (outside the Earth atmosphere the flux of particles is higher) and different particles are present (heavy ions and protons instead of neutrons) [45].

Furthermore, the focus in this dissertation is on faults capable of generating functional errors while faults which generate electrical failures like Single Event Latchups [46] and increase of absorbed current due to TID effects [13] will not be investigated in detail. The reason is that those are typically not addressed at microarchitectural level but at technology and electrical level instead.

#### SINGLE EVENT UPSETS (SEUS)

A Single Event Upset (SEU) can lead to a single or multiple upset. In the first case, the term Single Bit Upset (SBU) is employed. In the second case, the term Multiple Bit Upset (MBU) can be used[9]. The upset rate $\lambda_{ev}$ mainly depends on the radiation environment (including also shielding), the technology[10] and the choice of the sequential and combinational elements in the processor within the same technology. The upset rate can be either estimated with environmental models or measured on the field [48]. In the first

[9] Sometimes the term Multiple Cell Upset (MCU) is employed instead, while MBU is reserved to cases where the multiple upsets are in the same Error Detection And Correction (EDAC)-protected word.

[10] Several factors can be included in the technology. For instance, the error rate per bit on a specific technology depends on the voltage chosen (in [47] decreasing the voltage from 1.2 V to 0.8 V results in an increase of the error rate by a factor 1.5x up to 3x, depending on the radiation source). However, as shown in [47], this does not change the ratio between errors from combinational and sequential logic.

case, a standard approach is to carry out a radiation test composed of several test runs with particles with different Linear Energy Transfer (LET)[11] and measure the respective cross section[12]. Afterwards, tools like SPENVIS [50] are used to calculate the differential LET spectrum which can be obtained from the particle differential energy spectra in a certain orbit [49]. The upset rate can be then found with the following integral [49]:

$$\lambda_{ev} = \int_0^{\infty} \int_{-1}^{1} \int_0^{2\pi} f(L,\theta,\phi)\, \sigma(L,\theta,\phi)\, d\phi\, dcos(\theta)\, dL \tag{1.6}$$

where the differential flux $f$ and the cross section per bit $\sigma$ depend on the LET $L$ and the incidence and rotation angles ($\theta$ and $\phi$) [49].

Data from [51] show for a commercial 28-nm Fully-Depleted Silicon-On-Insulator (FD-SOI) SRAM a predicted in-orbit SEU rate of $4.66 \cdot 10^{-9}$ upsets/bit/day for solar minimum in Geostationary Orbit (GEO). From data in the same work, an estimation of $5 \cdot 10^{-7}$ for *worst week* in GEO and $5 \cdot 10^{-10}$ upsets/bit/day for Low Earth Orbit (LEO) can be taken (three orders of magnitude less than GEO worst conditions). Data from [52] show that considering different time spans will have different worst cases, e.g. the upset rate for the worst case of an SRAM array for one week in GEO is one order of magnitude lower than the worst case for 5 minutes, the latter reaching an upset rate of around $10^{-2}$ upsets/bit/day (similar values are given in [53], some of them even reaching $10^{-1}$ upsets/bit/day). Furthermore, the upsets are not homogeneously distributed in a certain orbit. For instance, all reboots in [54] (LEO) due to upsets happened in the South Atlantic Anomaly (SAA) and over the poles, where the level of radiation is higher due to the lower magnetic field shielding. To provide a comparison with processors in terrestrial environment, the upset rates at sea level in [55] is assumed to be $2.7 \cdot 10^{-11}$ upsets/bit/day, which is four orders of magnitude less than for the 28-nm FD-SOI in GEO (*worst week*).

The radiation environment experienced by the processor depends also on the amount of shielding, which cannot be controlled by the designer of the processor. In [52] it is shown that the reduction of upset rate due to an ideal aluminium sphere going from 0.1 mm to 2.5 mm is of 4 orders of magnitude for a 45-nm Silicon-On-Insulator (SOI) SRAM in the case of trapped protons, typical of LEO [56]. Considering an electronic box in a spacecraft brings the upset rate down of roughly another order of magnitude. However, in [52] it is shown that Galactic Cosmic Rays (GCR) are not affected by shielding depths. This causes a plateau of $8.64 \cdot 10^{-7}$ upsets/bit/day for the SRAM technology considered in [52], where adding more shielding does not improve the radiation tolerance of the part which must be addressed exclusively at semiconductor level.

In a similar manner, different technologies exhibit different upset rates in the same radiation environment. A typical RHBD SRAM memory based on a 250-nm technology has been reported in [48] to operate in GEO with an average of $1.8 \cdot 10^{-10}$ upsets/bit/day. A commercial SRAM based on 65nm bulk technology in [57] is reported to experience an average of $1.5 \cdot 10^{-7}$ upsets/bit/day in LEO, and in GEO would show an even higher

[11] The LET represents the energy loss of the particle when it travels a unit distance in the semiconductor [49]. It is typically normalized to the density of the material and given in $MeV cm^2/mg$.

[12] The device cross section for a given LET is defined as the quantity that multiplied by the particle flux produces the SEE rate of that flux of particles. It is typically given as $cm^2/device$ or $cm^2/b$ [49].

upset rate. Space-grade processors are currently based on 65-nm (e.g. GR740 [58]) or even 180-nm (e.g. GR716 [30]) RHBD ASIC technologies, while typical processors for terrestrial application are typically below 28 nm (e.g. [59]). These newer technologies are expected to be more vulnerable: when scaling from 65 nm to 14 nm the upset rate increases from around $10^{-12}$ to around $10^{-11}$ upsets/bit/day for planar bulk technologies, while it increases from $10^{-11}$ to $10^{-10}$ upsets/bit/day for FDSOI and Fin Field-Effect Transistor (FinFET) technologies [51] (all of them measured at ground altitudes). For all three types of technologies the increase happens when going beyond 28 nm, while from 65 to 28 nm the upset rate is constant or slightly decreasing.

Even in the same technology, different sequential elements composing the processor can have different upset rates. For instance, the OpenSPARC T2 in [60] (65 nm) is mainly composed of SRAM arrays optimized for density (for caches) with an upset rate ranging between $8.58 \cdot 10^{-13}$ and $1.14 \cdot 10^{-12}$ upsets/bit/day, less-dense and higher-performance SRAM arrays (for register files) with an upset rate per bit of half or less and FFs with an upset rate per bit of one-third or less compared to the SRAM array optimized for density. However, as Ref. [61] shows, this is not always the case and several technologies (especially newer ones) show the opposite situation. As a matter of fact, the ratio of the upset rate of FFs to SRAM cells in [61] is 0.44 for 130-nm, 1.96 for 90-nm, 1.75 for 65-nm and 1.15 for 40-nm technologies.

The differentiation between FFs and SRAM arrays is also required because FFs have temporal masking, which is not present in SRAM arrays. Considering an upstream sequential element connected to a downstream element through combinational logic, an upset happening in the upstream element between $t = t_{samp} - T_{prop}$ and $t = t_{samp}$ (where $t_{samp}$ is the sampling instant given by the clock and $T_{prop}$ is the time required for the correct sampling of a signal propagating from the upstream to the downstream element) will not propagate to the sequential elements downstream. A sampling factor can be defined as

$$SF_{FF} = 1 - \frac{T_{prop}}{T_{clk}}, \tag{1.7}$$

where $T_{clk}$ is the clock period for the FFs. This implies that the fraction of temporally masked errors in FFs actually increases with the frequency [47]. Despite this masking, typical models used in literature assume a constant failure rate for FFs when changing frequency [62], while more refined analyses find that there is an increase of the failure rate due to a Single Event Transient (SET) mechanism in the combinational logic between master and slave. Data provided in [63] show that this increase is very small, when considering a single FF the maximum found is $5 \cdot 10^{-15}$ errors/bit/day/MHz. Considering a design going from 100 MHz to 1 GHz, the error rate increases by $4.5 \cdot 10^{-12}$ errors/bit/day, which is of orders of magnitude less even compared to the less vulnerable technologies for space (around $10^{-10}$ upsets/bit/day). However, as mentioned in [47], testing shift registers where $T_{prop}$ is close to zero fails to take into account temporal masking, and $SF_{FF}$ is close to one for practical values of frequency. On the other hand, when testing a circuit with both sequential and combinational logic, understanding which of the two generated the error sampled in an FF to validate the temporal masking model is a daunting task. According to the model in [47], temporal masking instead can have a considerable impact. In [47] an average $SF_{FF}$ of 66.6% is given. When lowering the frequency

on the same design the sampling factor increases, until for 100 MHz the sampling factor gets to 96.66%.

Even the same type of sequential element can come in different sizes for the right performance/power/area trade-off. Data from [64] shows that FFs for a 65-nm commercial bulk technology have upset rates ranging between $1.6 \cdot 10^{-7}$ upsets/bit/day (fastest FF) and $4.1 \cdot 10^{-7}$ upsets/bit/day (slowest FF, 2.56× more vulnerable). Rad-hard (radiation-hardened) versions of the same technology have upsets rates ranging from $8.12 \cdot 10^{-8}$ to $1.82 \cdot 10^{-9}$ upsets/bit/day (2.24x increase of vulnerability with a 3x increase in drive strength). From [65] it can be seen that a rad-hard version of a FF on commercial technology can achieve a reduction of upset rate of 350×. In [47] several frequency targets (ranging from 100 MHz to 900 MHz) are set when synthesizing a processor, generating implementations with different mix of FFs. This increases vulnerability up to 10% (i.e. $RV_{FF} = 1.1$) taking the less vulnerable as reference. This increase follows a regular pattern, growing with the difference between the target frequency (e.g. 900 MHz) and the real clock frequency (e.g. 100 MHz).

The upset rate $\lambda_{ev}$ is typically assumed constant [66] (i.e. inter-arrival times of errors are independent [66]) and therefore the reliability function is exponential for each sequential element, i.e. $R_b(t) = e^{-\lambda_{ev} \cdot t}$. The use of the exponential distribution implies that the error rate of a series of elements becomes the sum of the error rates and the probability of not having an upset in the processor is $R_{SEU}(t) = e^{-SER_{SEU} \cdot t}$, where the Soft Error Rate (SER) due to SEUs is:

$$SER_{SEU} = \lambda_{ev}\left(N_{SRAM} \cdot RV_{SRAM} + N_{FF} \cdot RV_{FF} \cdot SV_{FF}\right) \tag{1.8}$$

where $N_{SRAM}$ and $N_{FF}$ are respectively the number of SRAM cells and FFs. $RV_{SRAM}$ and $RV_{FF}$ are the average vulnerability of respectively SRAM cells and FFs employed relatively to a reference sequential element with event rate $\lambda_{ev}$.

When considering MBUs, they can be measured as fraction of the total events. This means that if two events happen, one generating a SBU and one a MBU, the fraction of MBU is 50% regardless of the number of errors due to the MBU. Data from [67] show that for SRAM arrays in a 90-nm ASIC technology 95% of events cause a SBUs, 4% cause a $MBU_2$[13] and 1% cause $MBU_3$). For 65-nm SRAM arrays the situation reported in [67] is quite different: 45% are SBUs, 18% are $MBU_2$, 10% are $MBU_3$ and 27% are MBU(≥4). As a pessimistic estimation for Ultra Deep Sub Micrometer (UDSM) technologies data from [68] for a 32-nm SRAM array can be taken: in this case the fraction of SBUs is 24%, the fraction of $MBU_2$ is 52%, the fraction of $MBU_3$ is 3% and the fraction of $MBU_{\geq 4}$ is 21%.

#### SINGLE EVENT TRANSIENTS (SETS)

A single particle hitting a combinational node is able to cause a transient voltage pulse [69]. This pulse can be latched by the sequential elements downstream. A SET can be either seen as a single error or multiple errors in sequential elements by the user (e.g. software level). Even if the user is not able to distinguish between SETs and upsets, SETs have different generation mechanisms that require different redundancy techniques compared to SBUs and MBUs. As a matter of fact, SETs have additional levels of masking

[13] In this dissertation, the notation $MBU_n$ will be employed to indicate MBUs causing $n$ upsets with a single particle strike.

(electrical and logical) [70]. Furthermore, they have a different temporal masking mechanism: if the pulse reaches the sequential element outside from the sampling window, then the spike is not sampled and the error not generated. This implies that the contribution of SETs increases with the increase of the frequency. The reason is that when frequency increases, the sampling window becomes a larger fraction of the total time.

In relatively old technologies (e.g. technology nodes larger than 90 nm), SETs are not predominant as they are attenuated by large capacitance (electrical masking) and the low clock frequencies make the sampling unlikely (temporal masking) [71]. In more recent technologies instead, capacitance is reduced and the clock frequency is higher. For this reason, the probability that a spike is latched increases [71]. In [72] a comparator, an FF chain and an inverter chain are tested to compare the contribution of SETs and SEUs on a 45-nm bulk technology. The chain of inverters in [72] has a depth (12 stages) to emulate the highest electrical masking available typically in designs and accounts only for electrical and temporal masking, while the comparator also account for logical masking. As logical masking depends upon the input combination, in [72] a best, average and worst case are given. The worst case counts around twice the SETs compared to the best case. Furthermore, in [72] errors due to combinational logic (inverter chain) are less than one eighth of errors in sequential elements up to 100 MHz, around half at 500 MHz and uncertainties overlaps for 1 GHz (even if the expected value is still at half the sequential elements). The crossover frequency is around 1.5 GHz for the inverter chain and between 1.7 and 5 GHz for the comparator. However, considering that the vulnerability of FFs decreases with frequency, the contribution of sequential logic would be higher and the crossover frequency lower. This shows how increasing frequency does not necessarily increase the error rate, but certainly increases the relative vulnerability of combinational logic in the design, making optimal redundancy for low frequency not fit for higher frequencies. The SER due to SETs can be written as:

$$SER_{SET} = \lambda_{ev} \frac{A_{comb}}{A_b} SF_{SET} RV_{comb} \tag{1.9}$$

where $A_{comb}$ is the area of the combinational logic, $A_b$ the area of the reference sequential element associated with $\lambda_{ev}$, and $SF_{SET}$ is the sampling factor of SET pulses (indicating how many pulses are actually sampled by the sequential elements downstream). In [73] the overall probability of a SET being latched given a strike is 16.55% for 45 nm, 21.31% for 32 nm, 26.27% for 22 nm and 28.71% for 16 nm. A best case with $SF_{SET} = 0\%$, an average case with $SF_{SET} = 15\%$ and a worst case with $SF_{SET} = 30\%$ will be assumed. Also in this case a $RV_{comb}$ was defined, to keep into account different frequency targets that will imply the choice of different combinational elements. Data from [47] show that different timing targets (e.g. 100 MHz) can increase the failure rate of combinational logic by 2x compared to the timing target minimizing the failure rate (900 MHz), when running both implementations at the same frequency (100 MHz). It should be noted that in the case of combinational logic, as opposed to sequential elements, smaller gates are more sensitive to SETs [47].

#### ERRORS IN SRAM-BASED FPGAS

The correct behavior of processors implemented on SRAM Field Programmable Gate Arrays (FPGAs) is dependent on large configuration memories. An interesting finding

in [74] is that the percentage of bit flips in the configuration memory normalized to the resource utilization (fraction of sensitive bits in the configuration memory divided by the fraction of slices utilized in the FPGA) is roughly independent from the specific IP core (ranging from around 3% to around 6%). However, the impact of soft errors on the microarchitecture is similar to those of hard errors (e.g. stuck-at [75]).

### 1.3.2. REDUNDANCY

In order to avoid an error due to radiation to become a failure, fault tolerance solutions are employed at microarchitectural level. Fault tolerant is achieved using redundancy in different ways:

- Hardware (or spacial) redundancy: in this case (part of) the hardware is replicated. This can be done at several levels: gates, modules, components, boards.
- Temporal redundancy: operations are repeated more than once and the result compared. This typically comes with little (if the reiterations are handled in hardware and transparently to the software) or no (when the repetition is handled by the software) hardware overhead. On the other hand, the performance penalty is large, but multithreaded processors can handle multiple copies of a single thread efficiently [76].
- Information redundancy: information is stored with more bits than strictly required. In this way an error can be detected (and sometimes corrected) with less area and power overhead compared to hardware redundancy.

Redundancy typically causes penalties in terms of area, power and frequency to the design. Therefore, assessing how much redundancy is required to achieve a certain level of fault tolerance of the system during operation is crucial to avoid designs with severe area, power and frequency penalties and little or no enhancements in dependability compared to COTS processors with similar features, as not all faults will cause wrong outputs. During the design of a microarchitecture, fault tolerance can be assessed by Fault Injection (FI) [77] and observation of the effects on the behavior of the system. FI can then be employed to evaluate the vulnerability to faults in each unit of an implementation using the Architectural Vulnerability Factor (AVF), the probability that a fault in particular unit will cause an error on the outputs of the processor [78]. The overall error rate of a subsystem is then given by the SER, due to radiation in a certain environment (orbit, shielding, solar activity etc.) on a certain technology (e.g. cross sections) and the sensitive area (e.g. the number of bits), multiplied by the AVF for that particular subsystem. The following two sections the most common redundancies are described, i.e. EDAC codes (information redundancy) and hardware replication (hardware redundancy).

### 1.3.3. ERROR DETECTION AND CORRECTION CODES

EDAC codes can be classified according to their capabilities in terms of number of errors that can be detected and corrected in a single protected memory block[14], which are determined by the minimum distance ('d', i.e. the minimum number of bits that differs)

[14] In the rest of this work EDAC codes will be assumed applied to words.

between two valid words of the code ('codewords') [79]. A binary $(n, k)$ linear block code encodes words of $k$ bits using $n = k + r$ bits, with $r$ being the number of check bits [79]. Despite several codes with high correction and detection capability are proposed in literature (e.g. in [80] up to 8-error detection and 9-error correction), implementations typically employ Single Error Detection (SED) codes ([81–84]) or Single Error Correction and Double Error Detection (SECDED) codes ([82–85]), as in [80] it is shown how with the increase of the minimum distance within the codewords there is an exponential increase in overhead in terms of area and energy per access to the memory block.

SED detects all single errors in an EDAC-protected block [86]. This is often referred to as 'parity', as can be easily implemented adding a zero if the block has an even number of ones or a one if the number is odd, so that all codewords have an even number of ones. Parity is an example of Error Detecting Code (EDC). Parity is also capable of detecting every odd number of errors, while an even number of errors will generate an undetected error. Given its simplicity and low overhead, parity is sometimes used at sub-word level to detect more than one error in one word. For instance in [80] an 8 bit-interleaved parity is described, which for a 64 bit word results in 8 times the overhead in terms of check bits. This approach increases area and power overhead linearly instead of exponentially with the detection capability (even if no correction capabilities are added).

SECDED corrects all single errors and detects all double errors in a memory block [80]. The probabilities of miscorrection and detection for more than two errors in the same word depends upon the specific SECDED code employed. In [87], the (39,32) *Hsiao* code has a miscorrection probability of 59.66% for triple errors, while for the (39,32) *Odd-Weight Column* code it is 58.43%. The miscorrection probabilities for the (72,64) *Hsiao* code and the (72,64) *Odd-Weight Column* code are respectively 56.28% and 54.78%. In [88], the *Odd-Weight Column* code (39,32) miscorrects 1.7% of four errors in a word and the (72,64) *Odd-Weight Column* miscorrects 0.8% of them. These codes are examples of Error Correcting Codes (ECCs).

### LAYOUT SOLUTIONS

A way to avoid the exponential increase of overhead due to EDAC codes with increased correcting and detecting capability is to apply Cell Interleaving (CI) at layout level to deal with MBUs instead of using codes capable of detecting more than two errors [89]. In CI, memory cells that belong to the same logical EDAC-protected word are physically non-contiguous in the memory array. In this way, a single ionizing particle capable of causing multiple upsets is more likely to cause several single bit errors in different EDAC-protected words. The figure of merit of a certain cell-interleaved memory is the Interleaving Distance (ID), which indicate how many columns in an SRAM array must be involved during a particle strike to have a non-zero probability of two upsets in the same word. In [89] it is shown that an ID of 16 comes with an area increment of 32% and power increment of 25%, and can be deemed enough to avoid MBUs in most technologies (even with conservative estimations). A 100% increment in power and area for CI in an SRAM array (i.e. the same given in [89] for a 4 KiB SRAM array when increasing ID from 4 to 32) will be assumed. This value is an upper boundary for the acceptable cost of interleaving in terms of area of power for a memory array, as duplication would have a similar cost.

### SCRUBBING

In [66] a model is proposed to quantify accumulation in an EDAC-protected word, from which (assuming that the scrubbing period is small compared to the Mean Time To Event (MTTE)[15] for the accumulation[16]) the MTTE for accumulation of two errors in a word of $n$ bits for an array of $M$ words and a scrubbing period $T_s$ is:

$$MTTE = \int_0^\infty e^{-\lambda_{ev}\, n\, M\, t}\,(1+\lambda_{ev}\, n\, T_s)^{M\frac{t}{T_s}}\, dt = \frac{1}{\lambda_{ev}\, n\, M - \frac{M}{T_s}\ln(1+\lambda_{ev}\, n\, T_s)} \tag{1.10}$$

The accumulation for three SEUs in the same word can be estimated instead using the following equation, derived in a similar way as in [66]:

$$MTTE = \int_0^\infty e^{-\lambda_{ev} nMt}\left(1+\lambda_{ev} nT_s + \frac{\lambda_{ev}^2 n^2 T_s^2}{2}\right)^{M\frac{t}{T_s}} dt = \frac{1}{\lambda_{ev} nM - \frac{M}{T_s}\ln\left(1+\lambda_{ev} nT_s + \frac{\lambda_{ev}^2 n^2 T_s^2}{2}\right)} \tag{1.11}$$

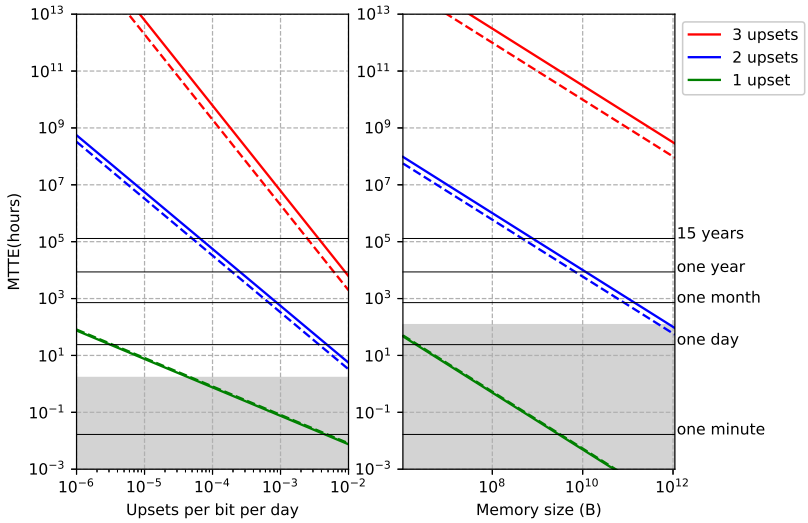


Figure 1.2: MTTE for accumulation of errors in the same word when changing upset rate (on the left: memory array size is 32 KiB and refresh rate is 10 minutes) and memory size (on the right: upset rate is 5E-8 upsets/bit/day and refresh rate is 12 hours) according to Equations 1.10 and 1.11. In both cases the impact of the word length and EDAC code is shown, i.e. solid line for (32,39) and dashed line for (64,72). In gray the range of MTTE where the models for 2 and 3 upsets in the same word are not valid for the selected scrubbing rate.

Fig. 1.2 (left) shows the $MTTE$ for one upset, accumulation of two and three upsets in the same word for extreme upset rates, assuming a 10 minutes refresh rate (which can be seen as a worst case estimation compared to realistic applications as in [90] is is shown

[15] The term MTTE is preferred to Mean Time To Failure (MTTF) in this case, to avoid confusion with the terminology that will be introduced in Sec. 3.4.1.

[16] In this work the models for accumulation of two and three bits are considered valid if $T_s < 0.1\, MTTE$.

that typical lifetime in a LLC is in the order of tens of microseconds). Even with this pessimistic assumption, accumulation is in general negligible compared to the contribution of MBUs (the ratio of MTTE of 'accumulation of two upsets' and 'one upset' is around 400 for $\lambda_{ev} = 10^{-2}$ upsets/bit/day and 2E+5 for 'accumulation of three upsets' and 'one upset'). This implies that accumulation will have negligible impact on failures due MBUs, as the latter are much more common (even considering LC, the ratio between two upsets and one upset is 24 and the ratio between three upsets and one upset is 95). The figure on the right instead shows that, even if the sensitivity of the accumulation to the memory size is the same for all events, the MTTE for large memories is small enough to contribute significantly to the failure rate. For instance mass memories like the one described in [91] have a memory scrubber to read and correct locations according to EDAC codes, therefore limiting accumulation to the scrubbing period. Furthermore, it is worth to note that, while memories with words of 64 bits perform slightly better for one upset because (72,64) is more efficient in terms of added cells compared to (39,32) (i.e. the product n×M is slightly smaller for memories carrying the same amount of bites), memories with words of 32 bits perform better for accumulation of two upsets and (by a larger margin) for accumulation of three upsets. This is intuitive, as accumulation becomes more likely when the number of bits in a word increases.

### 1.3.4. HARDWARE REPLICATION

The most common forms of hardware replications are at FF-level and at processor-level.

#### FF LEVEL

To protect the parts of the processor composed of logic, one of the most common approach is the one described in [81] for the LEON2FT. This approach uses Triple Modular Redundancy (TMR) on FFs (and will be indicated in the rest of this dissertation as FF-TMR), sampling and storing a bit on three different FFs and using a voter on the output to mask upsets and provide the correct value without any CCs of latency. To avoid common failures to the FFs in the TMR, each of the three FFs can have separate clock-trees, so that a SET in one clock-tree can be tolerated even if the data of a complete lane of thousands of registers is corrupted [81]. FF-TMR is applied also in [92], where it provides a 2.5x reduction in wrong commands/content at the outputs (used in conjunction with safe FSMs). The authors of [92] suggest that between 20% and 40% of the errors in the baseline processor are the result of SETs. As a matter of fact, SETs in the combinational logic can still be sampled by the majority of the FFs of a FF-TMR.

However, triplicating both sequential elements and combinational logic (to address also SETs) is reported to increase $T_{min}$ by 60% and area by 326% [93], which is a very high cost. To address also SETs with less overhead, [94] proposes a FF-level TMR with different delays for the three FFs (FFD-TMR) to avoid that a SET is sampled in more than a FF. The area of a FF-TMR cell is [95] is 3.47× larger than a regular FF and consumes 2.7× its power. FFD-TMR cells are instead reported to be about 6× larger than a FF in [94] and 5.2× in [96]. Furthermore, in [96] FFD-TMR cells consume between 3 to 4× the power of a regular FF, depending on the switching activity. The minimum clock period in the FFD-TMR version is 45% longer than the baseline, showing a substantially larger penalty compared to FF-TMR without delays. As a matter of fact, in [81] FF-TMR increases the

minimum period for correct execution of 8% on a 250 nm ASIC technology and the same value is given from different authors in [97] for the same processor using a 65nm ASIC technology.

Furthermore, in order to minimize the penalty in frequency, the triplicated FFs are typically placed close to each other. In this way, MBUs can cause wrong data to become majority and to be promoted to correct state, state causing data corruption [97].

#### PROCESSOR LEVEL

In [98], a processing Core-level TMR (indicated in the rest of this dissertation with C-TMR) is describes. This approach is not found to cause frequency penalties. However, in [99] a 10% penalty is reported, which shows that, even if not critical as in the case of FFD-TMR, the frequency can actually be penalized. A drawback of this approach is that errors are not masked with zero latency like in the case of FF-TMR and FFD-TMR, even if the $T_{eh}$ can be kept low enough compared to a hard or soft reset. When a discrepancy in the outputs is found, the processor takes 923 CCs to save and 909 CCs to restore the state (with caches enabled), for a total of 1,832 CCs. The time required to propagate the error to the service interface does not influence the availability, as correct operation is ensured until the error is propagated to the outputs. The propagation time has instead to be considered for accumulation, as it is possible that the data selected as 'golden' and replicated into all the three cores have latent errors that will manage to reach the outputs of the three processors completely undetected after the state is restored. However, even considering the most vulnerable design/technology combination (HE-1 without caches for HC technologies and $\lambda_{ev} = 10^{-6}$ upsets/bit/day) with the worst-case propagation time [100] (1,204 CCs at 100 MHz, i.e. 12.04 $\mu$s) used as $T_{prop}$, the probability of accumulation of two errors is negligible (8 orders of magnitude less compared to the failure rate due to the cacheless HE-1).

The situation in terms of unavailability is quite different with Core-level Dual Modular Redundancy (C-DMR) (e.g. [101]), as it is not possible to vote to chose a golden version when a mismatch is found and a soft reset is required. Another possibility is to save periodically the status of one of the core [102], but this generates substantial penalties in terms of execution time (ranging from +26% to +548%).

## 1.4. STATE OF THE ART OF SPACE-GRADE PROCESSORS

The first processors employed in space were based on ISAs specifically designed for airborne computers meant to be used in military avionic systems (e.g. MIL-STD-1750 [103]). With the introduction of proprietary commercial ISAs like x86, PowerPC and MIPS[17], the space industry could rely on software ecosystems and developments from the commercial field. In the nineties, the European Space Agency chose Scalable Processor ARChitecture (SPARC) for its ERC32 and LEON processors, as it was the only solution available at that time providing both openness and available software support [104]. Currently the European space industry (and a large part of the worldwide space community) is using Fault-Tolerant (FT) LEON processors in all ongoing and planned missions [58]. LEON is based on SPARC V8, an open and royalty-free ISA. This had compelling motivations at the time of the introduction of LEON for space [81] and the code

[17] MIPS can indicate both the Microprocessor without Interlocked Pipelined Stages ISA and a multiple of IPS.

of the non-FT version is available to the public (except for LEON4) with a GPL license. Although the LEON processor was designed for the Atmel's AT697F (2009) ASIC, it has been increasingly used within the GRLIB Intellectual Property (IP) core library as a System-on-Chip (SoC) platform. The availability of (relatively) large rad-hard FPGAs (especially with introduction of ACTEL/Microsemi RTAX series) helped this process, since the GRLIB allows components and units manufacturers can build an FPGA-based SoC with minimal manpower needed to develop hardware (HW) and software (SW).

### 1.4.1. PERFORMANCE

At the time of writing, the LEON4FT is the most powerful flight-proven European Fault-Tolerant IP core for space [30] and the GR740, a space-grade ASIC based on a quad-core LEON4FT, has been selected for ESA's Copernicus and NASA's WFIRST [105]. The main features of LEON4 are reported in Table 1.1.

Table 1.1: Features of the LEON4FT [106].

| | |
|---|---|
| ISA | SPARC V8 (32-bit) |
| ILP | 7-stage, single-issue, in-order |
| DLP | No |
| TLP | No |
| Speculation | static-branch prediction (always taken) |
| Interconnect | AHB (single-layer) |
| PLP | up to 4 (silicon-proven) |
| Cache management | write-through with write-invalidate |
| Clock frequency | 1.5 GHz (32-nm ASIC technology) |
| Linux-like OSs | Yes |

Fig. 1.3 shows how different commercial single-core general-purpose processors compare to the LEON4 and the LEON3 (year of release included within brackets) in terms of CoreMark/MHz. The collected data are not intended to prove the superiority of one implementation over another, but to suggest the opposite: the final performance of a single core processor for general-purpose applications is mainly determined by the level of ILP employed, especially considering the large uncertainties involved in the comparison of performance for processors of this complexity (e.g. compilers, different size of caches, different technology). For instance, different compilers and compiler flags can cause large differences in the final result. In [107], four different measurements (IDs: 1119, 1120, 1046 and 1050) on the Texas Instrument OMAP3530 are given, involving two different compilers (Sourcery G++ 4.4-179 and GCC 4.3.3) and two different optimization flags (-O2 and -O3). The minimum score is 12.75% less than the maximum measured score, while using different compilers with the -O3 flag leads to a reduction of 5.4% in the score. The -O3 flag activates only a subset of the possible optimization flags and other optimization flags like "-funroll-all-loops" can affect the performance too. However, all data from [107] employed in Fig. 1.3 have the -O3 flag.

Fig. 1.4 shows the strong correlation between ILP and performance for the general-purpose processors from Fig. 1.3. The values found can be used to provide an estimation

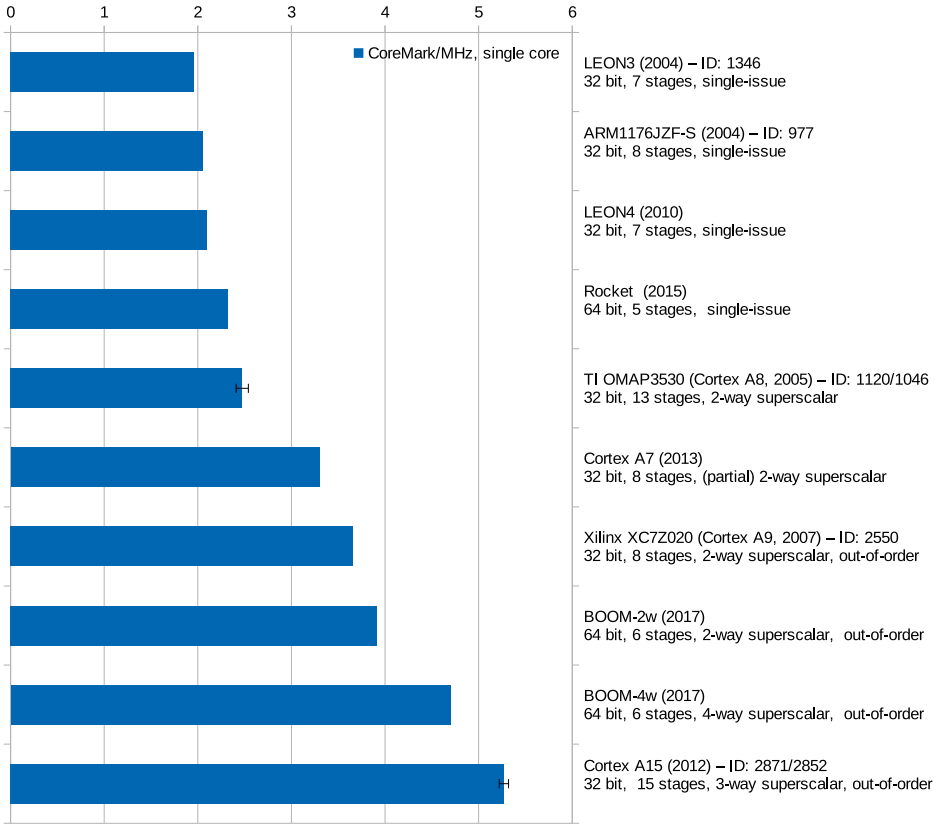


Figure 1.3: Performance of single-core processors and their main features. ID of the benchmarks is provided for data taken from [107]. Data for LEON4 is from the vendor, data for Rocket and BOOM is from [108]. Data for Cortex A7 is from [109]. Error bars are given when more results are available.

of target performance for an implementation with a certain ILP. However, this approach has some limitations, as explained below.

For instance, specific instructions can improve performances, e.g. single-cycle Multiply-Accumulate (MAC) instructions can effectively speed up the execution of CoreMark [110]. Also microarchitectural features can increase performance, like dual-ALUs, as [111] shows that substantial improvements can be obtained and [109] describes a 10-stage 2-way superscalar pipeline achieving 5 CoreMark/MHz with dual-ALUs. Furthermore, other kinds of parallelism can increase performance. For instance, the TLP employed in the MIPS i6500, a 9-stage processor with simultaneous multithreading, reaches 5.6 CoreMark/MHz for two threads [109]. In addition, Fig. 1.3 does not include the description of the degree of speculation employed in the implementations (e.g. branch prediction), but, as a general rule, the wider and deeper the pipeline, the more speculation is required to not waste a large amount of operation cycles (e.g. increasing the pipeline depth reduces IPC due to functional latency [112]). For instance, LEON4 em-

ploys a static branch prediction scheme (always taken) which exploits the fact that loops, if taken, are typically iterated more than once. To increase IPC further, the Rocket processor [113] employs a BHT for dynamic branch prediction (prediction learning from past outcomes of the branch), a BTB for dynamic branch target prediction, and a return stack to compensate the bad performance of a BTB for returns from functions (the same function can be called from several positions within a program).

Some considerations about the metric employed in the comparison are also required. CoreMark/MHz is a convenient metric to fairly compare implementations in different technologies, but neglects the increase in the maximum allowed frequency for correct operation due to shorter datapaths in deeper pipelines and measuring only the improvement due to simultaneous execution of more sub-operations from more instructions. Finally, comparing processors only using a metric evaluating performance without considering area efficiency of ILP solutions may lead to implementations too big for RHBD technologies and space-grade FPGAs, especially because area efficiency typically decreases when ILP is increased. For instance, the data from [108] show that the single-issue Rocket (comprising the core and the L1 caches) has an area efficiency of 4.64 CoreMark/MHz/mm$^2$, the 2-way superscalar OoO BOOM has 3.55 CoreMark/MHz/mm$^2$ and the 4-way version 3.36 CoreMark/MHz/mm$^2$ (all of them in 45-nm technology).

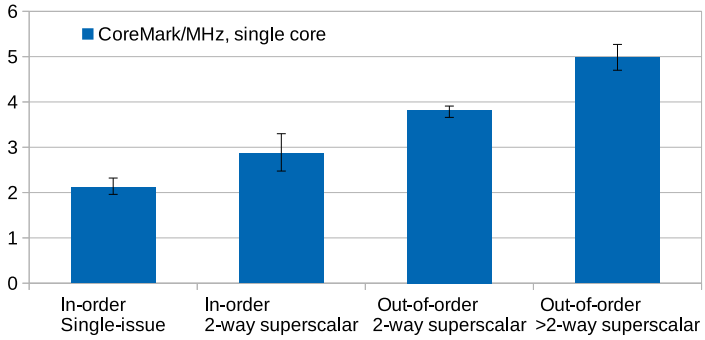


Figure 1.4: Expected performance of implementations employing different levels of ILP, from the data shown in 1.3.

Keeping in mind all these limitations, Fig. 1.4 suggests that an in-order single-issue processor is not enough to reach 3 CoreMark/MHz and that more than 2-way superscalar and OoO execution are needed to achieve 5 CoreMark/MHz. Considering other kinds of parallelism and microarchitectural solutions mentioned, either simultaneous multithreading or a deep pipeline (equal or greater than 10) plus dual-ALUs and MACs can help reaching 5 CoreMark/MHz.

It should also be noted that Figs. 1.3 and 1.4 only consider performances of single cores, while many general-purpose processors provide more than one processing core. Fig. 1.5 shows the performance enhancement due to the increase of the number of cores for several SMP implementations of the Cortex A9. The figure shows that a state-of-the-art SMP implementation can provide a 3× improvement in terms of Core-

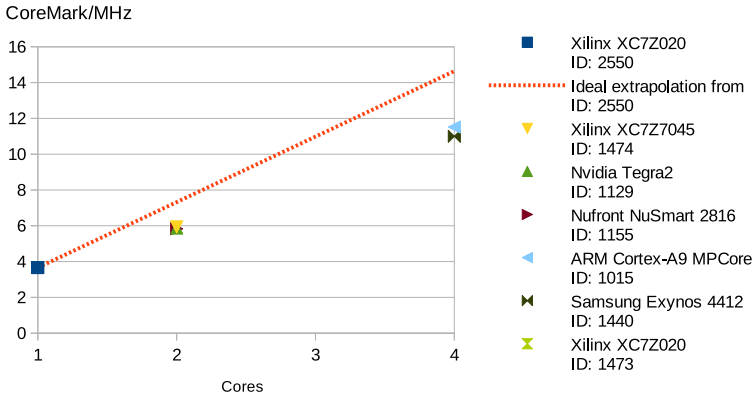


Figure 1.5: Performance in terms of CoreMark/MHz for different SMP implementations of the Cortex A9. Shown in red is the ideal enhancement obtained by multiplying the value for a single core by the number of cores. All data from [107].

Mark/MHz.

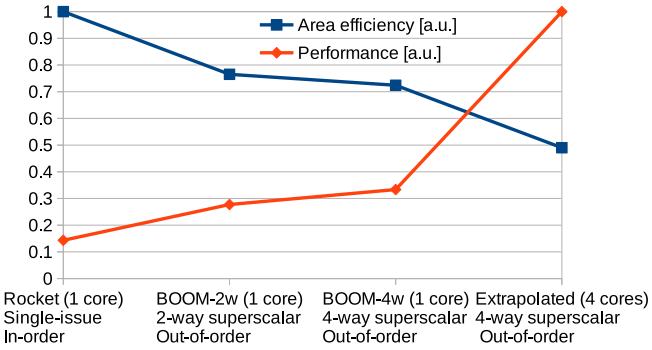


Figure 1.6: Comparison of performance and area efficiency of a general-purpose processor increasing its ILP ans PLP. Data is normalized to the achieved peak. Lines have been added to help visualize trends when moving from an implementation to another.

Fig. 1.6 show that although multicore provides a large improvement, it causes a decrease of area efficiency because of the large Level2 Cache (L2C) required. Furthermore, the problem of memory coherence in multi-core architectures becomes crucial and the choice of the coherence protocols greatly affects performance and fault tolerance. For instance, as noticed in [114], write-through protocols are inherently more robust compared to write-back protocols: in case of a single error the L1 caches can correct the error employing a parity bit and reading again the corrupted data from higher levels. In this case, only the main memory is strictly required to have correcting codes, as in write-through protocols data is available in more copies. For this reason, LEON proces-

sors use a simple invalidation write-through policy. However, write-through protocols generate large amounts of traffic. This is especially a problem in single-layer buses like AMBA High-performance Bus (AHB), where only a transaction is allowed per time. Use of switch fabric, which reduces contention allowing more than one transaction per time, allows in general a more efficient scaling with PLP, although this comes at the cost of a larger area compared to single-layer busses. Also, inter-task interference has a larger impact when employing write-through protocols compared to write-back protocols like Modified Exclusive Shared Invalid (MESI) [115].

### 1.4.2. FAULT TOLERANCE

As is clear from the discussion on cache management in the previous section, a trade-off between performance and fault-tolerance is required in space-grade processors. The FT mechanisms are often not described in detail in literature, and are only mentioned briefly in datasheets of components. Cache memories, like other memory arrays, are typically protected with information redundancy, employing EDAC codes, as shown in Sec. 1.3.3. For instance, the LEON2Ft employs two parity bits on caches [81]. There are two approaches to protect the rest of the processor:

1. Protecting the RFs and the logic (composed of FFs and combinational logic) separetly: this is the most common approach for an FT IP core for space can be found in [81], where the penalties in terms of area (Table 1.2) and frequency for a LEON2FT without Floating Point Unit (FPU) against a LEON2 on the same 250-nm ASIC technology are reported. The total overhead without considering RAM cells is instead 100% [81]. The main improvements for the FT version are TMR at FF level with separate clock trees and (32,7) BCH code on the register file. The penalty in frequency compared to the "non-FT" version is relatively small (-7.4%) and it's due to the use of TMR FFs on the critical path. Furthermore, hardware overhead due to the handling of errors in the register file is optimized by reusing hardware already available for traps. For instance, when an uncorrectable error is found in the register file of the LEON2FT an error trap is generated. Instead, when a correctable error is found the register file is written back with corrected data and then the processor jumps back to the failing instruction [81].

2. Protecting the RF and the logic simultaneously: this approach is employed in [98] and consists of replicating entirely the core, excluding large SRAM arrays (i.e. caches) that can be protected efficiently by information redundancy as shown in Sec. 1.3.3. In [98] the TCLS is described, a core-level TMR implementation of the ARM Cortex R5. The three cores share an IC and a DC.

## 1.5. MOTIVATION

Given the relatively low performance of space-grade processors and the small size of the space market compared to other embedded markets like mobile and automotive, the idea of a "spin-in" of COTS processors for terrestrial applications in space systems is becoming increasingly popular [116, 117].

Table 1.2: Comparison of area between LEON2 and LEON2FT on a 250-nm ASIC technology. The table is taken from [81].

| Module | LEON2 [mm$^2$] | LEON2FT [mm$^2$] | Area penalty |
|---|---|---|---|
| Integer unit | 0.86 | 1.61 | 87% |
| Cache controllers | 0.17 | 0.35 | 105% |
| Peripherals | 0.45 | 0.90 | 100% |
| Register File | 0.19 | 0.24 | 26% |
| Cache memory (16 KiB) | 2.42 | 2.59 | 7% |
| Total | 4.09 | 5.69 | 39% |

### 1.5.1. SPIN-IN OF DEVELOPMENTS FOR TERRESTRIAL APPLICATIONS

The term "spin-in" indicates the reuse in space systems of developments meant for terrestrial applications and this can be done at different levels of a space system:

- Subsystem: the spin-in of a full subsystem (e.g. a single-board computer) provides the maximum reuse of developments from terrestrial applications. The drawback of this approach is the total lack of specific solutions for space in the manufacturing process. Also, it leaves room for FT solutions only at software and at system level (e.g. redundant boards). Up to now this approach is limited to low-cost demonstration/university missions.

- Component: an ASIC developed for other applications is included in a space subsystem. The work in [117] shows that the 'cost of ownership' of COTS components (composed of procurement cost, cost of lot acceptance test, cost of FT mechanisms and their validation) is potentially higher than a space-grade equivalent. Therefore, [117] concludes that only the contribution of COTS components to system performance can justify their use in space avionics.

- IP core: to cope with an extremely high level of complexity, licensable and reusable blocks (IP cores) are employed during the design of a SoC. IP cores for terrestrial applications can be included as-is or after ad-hoc modifications to increase their fault tolerance in an ASIC or FPGA. In [118] it is described how an e600 PowerPC Core has been employed in a space-grade payload processor.

Even when the development of the processor starts from scratch, the designers can either chose to define their own ISA or use an ISA already available. ISAs are (sometimes expensive) IPs that enable the reuse of a certain software ecosystem (compilers, debuggers, integrated development environments). Furthermore, maintaining a software ecosystem is by far the biggest industrial challenge in the development of a processor [119].

### 1.5.2. THE RISC-V ISA

While open-source software has been around for decades, being the driving force behind most of the Internet and all of the top-500 supercomputers [120], hardware has not yet fully experienced the disruptive effects of openness, with the notable exception of

processors for space applications (mostly based on the LEON processors). Nevertheless, over the last years RISC-V has risen in popularity in terrestrial applications, drawing the attention of several universities and companies previously focusing on other open and free ISAs, proprietary ISAs or even on ISAs designed in-house (with the big drawback of having to design and maintain a software ecosystem).

RISC-V was originally developed by UC Berkeley to support computer architecture research and education oriented at hardware implementations, because a flexible, open and free ISA fit for such purpose was not available [121]. An example of the limits of previously existing free ISAs is OpenRISC, which features micro-architectural choices like branch delay slots and is not designed to be modular. Furthermore it does not cover present and future trends of embedded systems (does not support the 2008 revision to the IEEE 754 floating-point standard and the 64-bit version was not completed yet when the definition of RISC-V started) [121]. DARPA funded RISC-V in its very beginning [122] and is continuing to fund other activities related to the spin-in of open-source IPs in trustable electronic systems. The reason behind this interest is that open-source IPs and open ISAs can reduce the resources, time, and complexity required for 'secure and trusted' custom SoC design, as detailed information about open-source IP cores can be found by inspection. Also, ad-hoc improvements or modifications for security are much easier, avoiding the need of designs from scratch and thus ultimately increasing reusability [123]. The space industry can apply many of those considerations to enhance the fault-tolerance of existing open-source COTS IPs.

UC Berkeley and SiFive released in 2016 an open-source SoC generator called Rocket Chip to easily configure a SoC and automatically generate the synthesizable Verilog [113]. Since then, the RISC-V software ecosystems has matured quickly. Several ISA simulators, C compilers and debugging tools are available [124]. The availability of such ecosystem ignited the development of several open-source hardware platforms from several universities and companies. For instance, ETH Zürich and University of Bologna are working on the PULP Platform, a processing platform mainly targeted to the Internet of Things (IoT) [125]. PULP is based on several IP cores, ranging from a simple 2-stage 32-bit core to a 6-stage Linux-capable core with caches, and comprises several SoCs architectures (e.g. the single core PULPino and the multicore PULP). Also established players announced developments based on RISC-V. For instance Western Digital, a founder of the RISC-V foundation, announced that over the next few years all the processors shipped within their products will be transitioned over to RISC-V [126]. The reasons for this transition is that concerns are growing about monopolistic positions in the embedded market, as ISA owners protect their IP not allowing freely available implementations and free-market competition from other core designers, thus ultimately preventing reuse and ad-hoc designs. The adoption of a free and open ISA can thus lead to shorter time to market and lower costs from reuse.

## 1.6. RESEARCH QUESTIONS, METHODOLOGIES, AND THESIS STRUCTURE

From the description of the state of the art of space-grade processors in Sec. 1.4, it is clear that innovations are required to improve performance in space-grade processors

to enable the new applications introduced in Sec. 1.1. Furthermore, selection of fault-tolerant mechanisms in space-grade processors are typically based on heritage, without analysis of the target environment and their cost-effectiveness. Therefore, there is a need for a model to select the best redundancy approach. The novelty of these two tasks, and the wide range of problems involved in optimizing the trade-off between performance and fault tolerance, suggested that they could be explored more effectively with a PhD than with a set of specific R&D activity carried out by different companies in different contracts.

Given the considerations above, to improve the state of the art of space-grade processors, this dissertation answers the following Research Questions (RQs):

**RQ1: What are the advantages of using RISC-V in space applications?**

To answer this question, the RISC-V ISA specifications were analyzed in the first part of Chapter 2 (Sec. 2.1). The analysis consists in a comparison of the user and privileged RISC-V specifications with other ISAs employed in space-grade and in terrestrial processors.

**RQ2: How can RISC-V be employed in satellite data systems to solve key issues and enable new capabilities?**

After the advantages of the RISC-V ISA have been identified, the remainder of Chapter 2 investigates how to employ RISC-V processors in satellite data systems. To exploit modularity to its full extent, Sec. 2.2 identifies which types of processors are required in space data systems and how a RISC-V processor can be employed for each identified application. In order to identify all the types of processors required, different types of satellites with different constraints are considered, ranging from large satellites to small satellites, with a particular focus on CubeSats. Comparison of several state-of-the-art RISC-V processors against COTS and space-grade processors are provided for each target application. A critical discussion of benchmarks and metrics required for a fair comparison for each application is also provided, as ultimately the choice of the benchmark will select the best processor. Therefore, selecting the most representative benchmark for each application is paramount. From this analysis, several processor 'profiles' (including the suggested RISC-V subsets, microarchitectures, reference implementations and expected performance) to address a wide range of on-board application, from microcontrollers to high-performance processors for OBDM are proposed. Finally, in Sec. 2.3 a roadmap to bring available RISC-V processors from IP cores for terrestrial applications to space-level component is proposed.

**RQ3: How to design a fault-tolerant high-performance processor for space?**

After having described how to exploit the modularity of the RISC-V ISA, in Chapter 3 the focus is on the main advantage that comes with openness for space processors, i.e. the possibility to analyze its dependability as a white box instead of using a black-box approach (the only possible with proprietary processors). In order to do this, a model to evaluate the dependability of open processors is built starting from what is already available in literature (part of it already presented in Sec. 1.3 The model is then complemented with own contributions (e.g. error models, technology space, optimization function, analysis of redundancy and expected in-orbit behavior) and it is employed to identify the most vulnerable parts of a processor for space applications. Furthermore, considerations on different microarchitectures and environments are presented.

**RQ4: How to enhance the performance of next-generation processors for satellite data system architectures?**

After having identified in Chapter 2 processors for OBDM as the most promising type of processors to enable new system-level capabilities and the type of processors that would benefit the most from research and design exploration, an in-depth analysis of the workloads of machine learning algorithms is carried out in Chapter 4. In particular, the focus is on DNNs for image analysis and anomaly detection, as these are identified as the most impacting applications of machine learning on-board satellites. Each algorithm is decomposed in kernels, and for each of them the effectiveness of DLP is investigated, analyzing Operational Intensity (OI), problem size and memory traffic.

After this analysis, in Chapter 5 vector processors are proposed as a solution for this type of workload. The specific challenges of designing a vector processor are also highlighted, with a focus on the most critical elements of the microarchitecture (e.g. vector register file, memory hierarchy) and to the scalability with the amount of DLP. Furthermore, specific aspects related to dependability are investigated, like the use of EDAC codes. Finally, the Spike ISA simulator has been employed to compare the performance in terms of number of instructions of vector and scalar implementations of the kernels of a relevant DNN for image analysis.

Then, in Chapter 6, a novel general-purpose baseline RISC-V processor for space applications (NOEL-V) will be extended adding DLP (RISC-V standard Vector extension) to increase performance and efficiency of the processor for the OBDM workloads analyzed in Chapter 4, according to requirements formulated keeping into account the knowledge built in the previous chapters. Verification of correct operation of the vector processor in Chapter 6 will be carried out comparing the output of the kernels for the scalar and the vector processor. Verification of the requirements is then carried out, implementing the vector processor on a FPGA board and benchmarking it against the baseline processor on the same FPGA to prove that the requirement on the speedup is met. Furthermore, the vector processor is compared with other RISC-V implementations targeting the OBDM profile.

Finally, in Chapter 7 the main contributions of this dissertation to the body of knowl-

edge identified in Sec. 1.4 are described, highlighting innovations and their implications on future works.

# 2

# DEFINITION OF RISC-V PROCESSORS NEEDED IN SPACE DATA SYSTEMS

*Il futuro è di chi lo sa immaginare.*
*The future belongs to those who can imagine it.*

Enrico Mattei

*This chapter analyzes the RISC-V Instruction Set Architecture (ISA), focusing on the implications of its modularity and openness. Then, it identifies present and future needs in space data systems and proposes to address them with RISC-V processors. In order to satisfy different applications with contrasting requirements in satellite data systems, four different types of processors are identified: 1) low-area/low-power microcontrollers, 2) On-Board Computers (OBCs), 3) general-purpose processors for payloads and 4) enhanced payload processors for On-Board Decision Making (OBDM). Several solutions based on RISC-V are proposed and compared to proprietary Commercial-Off-The-Shelf (COTS) solutions and to space-grade solutions. The aim of this chapter is also to showcase the unprecedented number of open-source implementations and models relevant to space applications that were developed in a relative short time (2016-2018) on the RISC-V ISA.*

Parts of this chapter have been published in [1].

## 2.1. ANALYSIS OF THE RISC-V ISA

The following subsections analyze the RISC-V ISA to identify the advantages of RISC-V processors compared to processors based on commercial ISAs (e.g. ARM) and ISA currently employed for space processors (e.g. SPARC).



### 2.1.1. MODULARITY

The RISC-V manual is structured in two volumes, one for the user-level ISA and the other describing the privileged architecture with three privilege levels (User, Supervisor and Machine mode) [127]. An implementation can employ just the User mode, the User and the Machine mode (when security is a concern), or all of the three modes to support Linux-like OSs. The Hypervisor (H) mode (designed to support Type-1 hypervisors) has been removed and the encoding space reserved, as the RISC-V community is focusing on hypervisor support via an extended Supervisor mode suitable for both Type-1 and Type-2 hypervisors[1] [127] .

The user-level ISA is defined as a base integer (I) ISA, which must be present in any implementation, plus optional extensions to the base ISA [121], shown in Table 2.1. The integer base is restricted to a minimal set of instructions operating on 32 General-Purpose Registers (GPRs), providing a baseline around which more customized ISAs can be built. Alongside the general-purpose registers employed to store operands and results of instructions, Control and Status Registerss (CSRs) are also defined. As opposed to GPRs, each of these register has a specific function, i.e. configuring the status of the processor and/or allowing the software to know the status of the processor [121]. A subset of the integer base (E) can optionally be implemented for processors targeting small 32-bit microcontrollers, with 16 general-purpose registers. The standard defines a "general" subset (G) as the set of extensions typically required for general-purpose computing systems. Whereas other ISAs are treated as a single entity, which changes to a new version as new features and instructions are added over time (e.g. ARMv7 and ARMv8 [130]), RISC-V aims at keeping the base and the standard extensions unchanged over time, and instead plans to add new instructions and new features as further optional extensions. This helps in terms of software reuse, providing straightforward backwards compatibility. The base of RISC-V is similar to the original RISC developed in the Berkeley RISC project [131], but the whole ISA is updated to account for new trends and needs of the embedded systems market, since it is new (the baseline ISA has been ratified in 2019 [132]) and, being an open standard, allows open discussion on what must be included in the standard. For instance the standard defines 32-bit (RV32), 64-bit (RV64) and even 128-bit (RV128) address space variants, and provides features like a 16-bit compressed instructions extension (C) to increase performance, code density and power efficiency. The standard atomic instruction extension (A) adds instructions that atomically read, modify, and write memory for inter-processor synchronization us-

[1] Hypervisors provide isolation between different instances of OSs running for instance in multicore processors. Isolation is currently being investigated in space applications to improve safety behavior, i.e. to avoid that the failure of an application running on an OS may impact the execution of another application running in a different OS on the same processor [128]. A type-1 hypervisor (bare-metal) runs directly on the host's hardware. A type-2 hypervisor (hosted) resides above a conventional host operating system and provides a full set of virtual hardware resources to the above guest OS [129].

ing load-link/store-conditional instructions instead of compare-and-swap instructions, thus avoiding the ABA problem[2] [133] affecting the compare-and-swap instructions and allowing a straightforward use of modern crossbars[3] that do not support locked accesses, like Advanced Microcontroller Bus Architecture (AMBA) Advanced eXtensible Interface 4 (AXI4). Furthermore, RISC-V is little-endian, allowing straightforward integration with the most popular state-of-the-art embedded infrastructures and proprietary architectures (e.g. ARM).



Table 2.1: User-level standard extensions [121].

| Subset | Name |
|---|---|
| Integer | I |
| Integer Multiplication and Division | M |
| Atomics | A |
| Single-Precision Floating-Point | F |
| Double-Precision Floating-Point | D |
| Quad-Precision Floating-Point | Q |
| Decimal Floating-Point | L |
| 16-bit Compressed Instructions | C |
| Bit Manipulation | B |
| Dynamic Languages | J |
| Transactional Memory | T |
| Packed-SIMD Extensions | P |
| Vector Extensions | V |
| User-Level Interrupts | N |

### 2.1.2. SIMPLICITY

One of the implications of the modularity of RISC-V is that it was possible to keep the base ISA "I" simple, without the need to target any application in particular. As a matter of fact, employing instructions which are fit for a specific application makes the hardware implementation inefficient for other applications. As a result, even if the base subset I is similar to the original RISC developed in the Berkeley RISC project [131, 134] in the eighties, it has been greatly simplified. For instance, the RISC-V ISA does not require condition codes and register windows [121]. Another example is that, while SPARC V8 specifications (which are similar to the original RISC project [135]) prescribe the use of a combined hardware/software technique to deal with branch hazards called "branch delay slots"[4], RISC-V leaves the choice of the scheme to deal with branch hazards completely to the implementation (see examples in Sec. 1.2.2).

[2] ABA is not an acronym. It indicates the issue arising from the fact that a compare-and-swap instruction will not realize that A has be changed to B and then back to A, assuming that A remained unchanged [133].

[3] The term crossbars is used to describe an interconnect capable of connecting several masters and slaves without contention as long as multiple transaction do not compete for the same master or slave.

[4] In this case the compiler fills (or not, if nothing is available) the pipeline with useful operations while waiting for the branch resolution.

### 2.1.3. EXTENDABILITY

RISC-V allows both standard and non-standard extensions (the latter are defined outside the specifications). Although non-standard extensions are in principle possible for all ISAs, non-standard extensions require modifications in the software toolchain. In general this is not technically and/or legally possible for proprietary ISAs.

However, non-standard custom extensions for space applications will not be considered further in this work as a solution, as space is a niche market and often the lack of software support for processors in space is the main issue. Reusing the toolchain supported by a large community is the most perceived advantage of the introduction of RISC-V in space systems.

However, in other fields the situation is quite different. For instance, Xuantie-910 (a RISC-V processor from Alibaba, designed for high-performance computing) includes non-standard extensions to arithmetic, bit manipulation, load and store, TLB and cache operations [136]. In addition, there are extensions for the MMU to support page-based memory attribute management and for the interrupt controller to support permission control. However, also in this case compatibility with the standard toolchain is deemed as a desired feature, so the core can be configured to disable the non-standard extensions [136].

The RI5CY processor, designed mainly for IoT applications, provides instead non-standard instructions for bit manipulation, auto-incremental load and store, MAC, hardware looping and packed-SIMD operations [137].

### 2.1.4. OPENNESS

Regardless of the level where the spin-in takes place, the main advantage of the spin-in of a commercial processor is the reuse of the software ecosystem and of the experience of a much larger community working on terrestrial applications. For instance, a bug in a compiler or in an IP core can be found more easily than in the case of a processor only used in space applications. The advantage is even stronger when this community is built around an open and free ISA, as everyone can contribute to advancing the state of the art. For this reason, wide adoption by industry and academia is crucial. These considerations were critical for the adoption of SPARC by ESA in the nineties [104]. However, nowadays SPARC lost momentum in terrestrial applications, mainly because of the predominance of the proprietary x86. New markets with different requirements in terms of energy efficiency (e.g. mobile devices) have led to the success of proprietary ISAs from ARM [138]. Nevertheless, RISC-V has risen in popularity in recent years. The spread of an open and free ISA like RISC-V already enabled a vast field of research activities for terrestrial application (e.g. security, AI, etc.), as accessing proprietary architectures is costly and limits what can be done with a certain product or within a certain research activity. The availability of a software ecosystem supported by a large open community ignited an unprecedented amount of developments, with several announcements and/or releases of open-source implementations.

The rest of this section describes how openness can help enhancing two crucial aspects of space systems: security and fault tolerance.

### SECURITY

DARPA funded RISC-V in its very beginning and is continuing to fund other activities related to the spin-in of open-source IPs in trustable electronic systems [123]. The reason behind this is that open-source IPs and open ISAs can reduce the cost, time, and complexity required for secure and trusted custom SoC design, as detailed information available with open-source IP can be found by inspection. Ad-hoc improvements or modifications for secure and trusted application are much easier, avoiding the need to design everything from scratch and focusing only on the critical part and thus ultimately increasing reuse [123]. The work in [139] is an example of how the openness of RISC-V enables academia to work on systematic approaches to eliminating attack surfaces instead of fixing specific security holes and provide strong security guarantees against cache timing and memory access pattern attacks. Another example is [140], which proposes an extension of RISC-V against temporal and spatial memory attacks.

### FAULT TOLERANCE

Many of the considerations on open-source IP cores and security can be applied by the space industry to enhance the fault tolerance of existing open-source COTS IP cores.

When selecting a processor for satellite data systems, typically two choices are available: either a space-grade processor with long flight heritage and well-characterized behavior (e.g. LEON processors [30]), or a proprietary COTS processor employed as a black box sometimes after adequate radiation test [101, 141]). The latter is preferred to the former when the performance required cannot be met with space-grade processors [117], which typically lag behind their commercial counterparts in terms of performance [14]. The recent availability of open-source IP cores for terrestrial applications allows for a better understanding of their vulnerability, avoiding black-box characterization (typical of proprietary COTS components) and allowing a trade-off between the two approaches. A better modelling of the inner working of processors can both help choosing the best IP core and its configuration. For instance, in [142] the lack of public Register Transfer Level (RTL) models (typical of proprietary processors) is identified as the main issue when trying to characterize the effects of upsets in a microarchitecture (mainly because it is not possible to estimate the exact number of sequential elements). Furthermore, the authors of [143] suggest that the failure rate measured with particle beam experiments is much larger than the one estimated with FI due to undisclosed proprietary parts of the real physical hardware platform compared to the virtual platform where the FI was carried out. Once the vulnerability of a processor is estimated, it can be reduced employing redundancy. Redundancy typically comes with significant area, power and performance overhead. Therefore, assessing its cost-effectiveness is crucial. However, the amount and type of optimal redundancy can change drastically depending on the requirements in terms of dependability (i.e. reliability, availability, safety [39]) and performance, as well as on the target radiation environment. For instance, the focus of the standard ISO 26262 [144] for automotive applications is on functional safety. For this reason, several ASICs for automotive employ two processors executing instructions in lockstep, so that errors can be detected comparing the outputs of the two replicas and the processors are restarted in case of mismatch [101]. A similar approach can reduce availability, as for instance even benign differences at the outputs of the processors will cause a reset. Furthermore, as long as the safety requirements are met, availability is not

a primary concern in automotive. This is not the case for space applications, as dependable processors in space are expected to provide a certain service without interruptions over a certain span of time, hence the focus is instead on availability. For example, in the case of a telecommunication satellite in GEO the time span of a mission could be more than 15 years in which the whole space system is expected to provide a certain service 99.9% of the time [145]. This implies an even tighter requirement on the availability of the On-Board Computer (OBC). Furthermore, when the processor is intended for usage in space, the presence of ionizing radiation makes the occurrence of soft errors far more likely than on ground and the amount of redundancy must be carefully evaluated as power and area available in space data systems are typically very limited. On the other hand, loss of performance in space data systems can be easily tolerated in most cases (e.g. in Sec. 2.2.1 it will be shown that processors with low performance are enough to control large satellites) . In High-Performance Computing (HPC) the constraints are the opposite, as the amount of loss in terms of performance that can be tolerated is typically very limited [146].

## 2.2. PROCESSORS FOR SATELLITE DATA SYSTEMS

Throughout the eighties and early nineties space computer systems were limited by the very low processor performance. For instance, the first European single-chip processor was the ERC32, providing 14 MIPS at 20 MHz [58]. Driven by Moore's Law, the number of transistors on a single chip and the maximum clock frequency increased significantly over the years, improving system performance and enabling new computing possibilities.

The period between 1990-2010 was characterized by centralized computing that revolved around an OBC and non-intelligent (i.e. without software) Remote Terminal Units (RTUs), as described in [147]. Improved connectivity and process improvements in the period from 2010 enabled a shift towards decentralized computing, with a proliferation of connected intelligent devices and sensors, along with a shift to localized computing. The latter allows processing to occur closer to the data, effectively improving latency, bandwidth and energy use. The distribution of intelligence throughout the whole satellite, as shown in Fig. 2.1, is based on processors tailored to several power consumption, processing power, size and real-time requirements. Time-deterministic, low performance, low power, low area microcontroller processors (Sec. 2.2.2) can be employed in RTUs. The payload processor instead can either be just a general purpose processor (Sec. 2.2.3) or including an accelerator capable of enabling OBDM (Sec. 2.2.4). A payload module with an enhanced payload processor capable of making decisions requires less data bandwidth (or in the best case only the low-speed T&C link) to downlink only selected and meaningful data and/or the results of on-board processing.

Finally, the focus of the space industry in recent years shifted from large GEO satellites to small (< 500 kg) Low Earth Orbit (LEO) satellites (especially CubeSats) [148, 149].

The following subsections show how RISC-V solutions compare with state-of-the-art space-grade and COTS processors that fit best for each application in the architecture described in Fig 2.1.

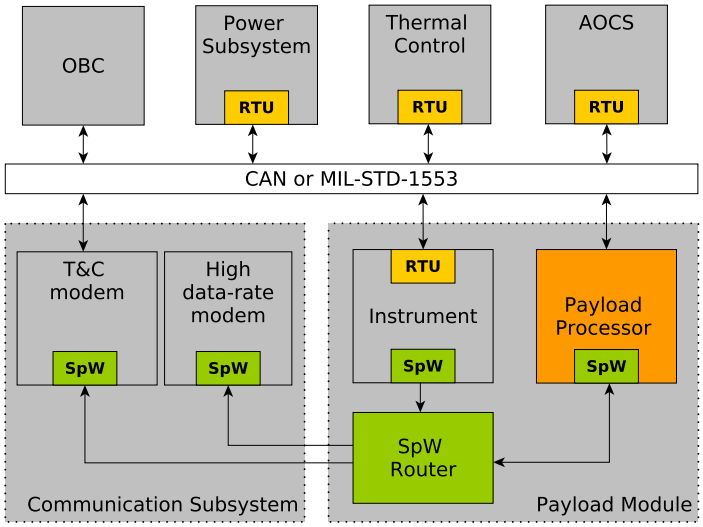


Figure 2.1: Distributed satellite data system architecture for a medium-large spacecraft based on processors with different target applications. The figure does not include any redundancy schemes and only one instrument connected to the SpaceWire (SpW) router is represented (when more instruments are present they can be connected in parallel to the one represented).

### 2.2.1. ON-BOARD COMPUTERS

OBCs are employed to execute tasks like attitude and orbit control, telecommands execution or dispatching, housekeeping telemetry gathering and formatting, on board time synchronisation and distribution, failure detection, isolation and recovery [150].

One of the most severe constraints for OBCs for CubeSats is the available power. As reported in [151], a 1U CubeSat has a total of 1-2 W available and a 3U has 5-6 W. On the other hand, a 6U CubeSat reaches around 20 W [152]. Given these constraints on power, OBCs for small CubeSats (1-3U) are based on simple single-core processors. This type of microarchitecture achieves low performance, as shown in Table. 2.2, ranging from 1 to 2 CoreMark/MHz[5]. As a reference, multi-core processors for mobiles reach around 16 CoreMark/MHz [153] and multi-core/multi-thread processors for Personal Computers around 40 CoreMark/MHz [154]. The peak absolute performance can be estimated multiplying the microarchitecture performance by the maximum achievable frequency, which depends on the technology (e.g. technology node) and type of Integrated Circuit (IC) employed, i.e. ASIC or FPGA. Here the gap becomes even larger, as clock frequencies in space applications are typically lower (in the order of hundreds [155] or even tens [156] of MHz instead of GHz [154]).

Small satellites are moving from short demonstrator missions to longer, more-critical missions [157]. Although the cost of replacing a failed SmallSat in LEO is relatively low compared to a classical large satellite in GEO, the dependabilty of the single spacecraft

[5]Although it is preferable to measure performance with CoreMark, these simple implementations are often still benchmarked with Dhrystone.

is required to increase. Therefore, radiation tolerance becomes more important and will move the CubeSats from being heavily reliant on COTS to use radiation-tolerant (rad-tol) or rad-hard parts [157]. For instance, there is a new trend of using fault-tolerant processors designed for space application. An example is the AAC Clyde Sirius OBC in [156], based on the LEON3FT, one of the most popular FT processors in space applications [30]. Furthermore, the Sirius OBC is different from the other OBCs considered in Table 2.2, as it is implemented on a FPGA with TMR on all FPGA FFs and EDAC codes on memories. One of the aims of these features is to allow deep space exploration missions with CubeSats [156].

Table 2.2: Performance of some OBCs in the PC104 board format (9.0 cm × 9.6 cm, the de-facto standard for CubeSats), measured with CoreMark and/or Dhrystone (DMIPS), depending on the data available. Values for power refer to peak (p), average (a) and minimum (m) consumption. Data from [155, 156, 158–161].

| OBC | Processor | Norm. Perf. | Abs. Perf. | Power [W] |
|---|---|---|---|---|
| NanoMind a3200 | AT32UC3C | 1.5 DMIPS/MHz | 91 DMIPS | 0.17 (a)<br>0.9 (p) |
| iOBC | AT91SAM9G20 | 1.1 DMIPS/MHz<br>1.5 CoreMark/MHz | 440 DMIPS<br>612.9 CoreMark | 0.4 (a) |
| Sirius | LEON3FT | 1.4 DMIPS/MHz<br>1.8 CoreMark/MHz | 70 DMIPS<br>90 CoreMark | 1.3(a)<br>0.8(m)-2(p) |

However, the RHBD solutions employed in [155] increase power consumption. Also, the power consumption of the OBC depends on the technology and on the use of the peripherals (i.e., interfaces and external memories), e.g. ranging from 0.8 to 2 W for the Sirius OBC [160]. Although the minimum consumption is below the total power available for a 1U CubeSat, such a large part of the power available shows that it is difficult to employ RHBD processors in 1U CubeSats, while in a 3U CubeSat they could be employed instead (the maximum consumption is 33%-40% of the total power available).

To compare the examples given for SmallSats to OBCs for large, high-reliability spacecraft, the OSCAR [162] from Airbus has a mass of 5 kg and is larger than many SmallSats (230×160×200 $mm^3$). Its peak power consumption is 15 W. On the other hand, OSCAR is expected to last 10 years in LEO and 15 years in GEO, and it is designed to be SEU[6] tolerant and Single Event Latchup (SEL)[7] immune [162]. However, it should be noted that the computational capabilities required for an OBC for a large spacecraft are on the same order of magnitude (1.8 CoreMark/MHz [161], 86.4 CoreMark at 48 MHz [162]). This shows that the gap between OBCs for SmallSat and OBCs for large spacecraft lies more in resilience to radiation effects than in performance.

Table 2.3 shows that OBCs for medium-large satellites have dramatically improved in terms of power (more than 6× reduction), mass (4× reduction), dimensions (5× reduc-

[6] A SEU happens when a ionizing particle changes the value stored in a single or more sequential elements.
[7] A SEL happens when a ionizing particles generates a low-resistance path between power and ground.

Table 2.3: Improvements of OBCs achieved with the advance of the state of the art of SoCs [162–167].

| OBC | GOCE OBC | GAIA OBC | OSCAR |
|---|---|---|---|
| Year | 2009 | 2013 | 2014 |
| Manufacturer | TAS-I | RUAG-S | AIRBUS |
| CPU | ERC32 | LEON2FT | LEON3FT |
| ISA | SPARCV7 | SPARCV8 | SPARCV8 |
| Frequency [MHz] | 24 | 80 | 80[8] |
| Abs. Perf. [MIPS] | 17 | 65 | 68 |
| Norm. Perf. [IPC] | 0.71 | 0.81 | 0.85 |
| Pipeline stages | 4 | 5 | 7 |
| Power [W] | ≤ 90 (avg) | ≤ 40 (avg) | 15 (peak) |
| Mass [kg] | 21 | 16 | 5 |
| Dimensions [mm$^3$] | 470×272×332 | 420×270×276 | 230×160×200 |



tion)[9] and absolute performance[10] (4× improvement) since the introduction of SoCs in space. For instance, OSCAR is based on SCOC3, a SoC which includes also TM/TC control and CCSDS standard, eliminating one board with respect to the former generation of OBCs [167]. However, the improvement in computational speed was mainly due to the increase of the pipeline stages and frequency and not to the microarchitectural efficiency of the processors, as the average number of IPC has increased only by 20%. The main reason is that OBCs mainly deal with real-time tasks leveraging Real-Time Operating System (RTOS) like RTEMS. Even the newest ones are based on single-core single-issue in-order processors like the LEON2FT and the LEON3FT, because higher performance processors employ microarchitectural features (e.g. superscalar execution, out-of-order execution, speculation, multicore, several level of caches) that pose challenges to the development of real-time software, especially during Worst Case Execution Time (WCET) [168]. As a matter of fact, software standards for on-board software (e.g. [169]) require a schedulability analysis, which in turn requires a WCET analysis of each task [170]. An example of WCET for an OBC is given in [170].

The previous considerations on the performance/time-determinism trade-off for the microarchitecture of the OBC suggest that the performance of the whole data system can be increased by distributing software for data acquisition/processing and simple control applications in other spacecraft subsystems rather than using higher-end processors for OBCs. This approach is also suggested by the fact that sensors, actuators and payloads are inevitably distributed throughout the satellite.

Another reason behind the use of simple microarchitectures in OBCs is that "command and control" algorithms run by OBCs have a very low OI, i.e. the number of op-

[9] The data provided in Table 2.3 for power, size and weight refer to a single board with a single computer (processor, memories, etc.). Typically two of them are employed (to overcome common-mode failures on a single board), using an additional board to handle the redundancy at board-level.

[10] Absolute performances in Table 2.3 are given in IPS. Therefore, a fair compariso is possible only between implementations of the same ISA, as it counts only the number of instructions per second without evaluating their impact on the execution times. As SPARCV8 added multiply and divide instructions to SPARCV7, the improvements of SPARCV8-based OBCs over the one based on SPARCV7 is actually underestimated.

erations executed per byte read from the memory. For instance, considering a program reading two numbers and storing their sum (all of them represented with 32 bit), the OI is limited to 1/12. Higher-end applications, like image processing, have instead a much larger operational intensity, typically from one to three orders of magnitude larger (See Chapter 4). For this reason, while the latter are effectively sped up by increasing the computing capabilities of the processors (e.g. increasing parallelism), the former are essentially limited by the memory bandwidth. Sec. 2.2.4 introduces the 'roofline' model as a tool to distinguish between these two kinds of algorithms and to evaluate when an increase in computing capabilities is actually needed for a specific algorithm on a specific platform.

For all the aforementioned reasons, a RISC-V substitute of the LEON3FT (when used as main OBC processor) would require a similar microarchitecture, achieving similar performance with the drawback of giving up long flight heritage and large amount of OBC software legacy for the LEON processors. On the other hand, a RISC-V substitute could leverage a larger software ecosystem in the future for new developments, building on a much larger user base for (e.g.) the maintenance of the SW toolchain.

The Rocket processor is the RISC-V implementation closest to the LEON3 and a more detailed comparison is given in Sec. 2.2.3, where multicore versions of these processors are evaluated as low-end general-purpose processors for payloads.

### 2.2.2. MICROCONTROLLERS

RTUs are usually present on medium-large size spacecraft. In the past, the RTU was typically a "non-intelligent unit", i.e. without software [147, 171]. However, there is a trend to have at least a low-performance microcontroller in the RTU in order to implement locally autonomous acquisition and control sequences, to gather the analogue and digital telemetry from sensors and units (e.g. temperature), to calibrate raw measurements, to provide the conditioning for analogue sensors, to control the reaction wheels, gyros, star trackers, sun sensors, propulsion and other units in the Attitude and Orbital Control System (AOCS), to control solar array drive mechanisms, to distribute power to heaters, to distribute (in some cases also downconverting) power to active loads, and control and monitor payload units [171]. For instance, [172] describes a satellite architecture employing RTUs containing an RTAX-based microcontroller. This trend led to development of the DPC from Thales Alenia Space [173] and of the GR716 from Cobham Gaisler [30], both funded by ESA and based on 180-nm RHBD ASIC technology, as well as several other rad-tol microcontrollers.

The use of RTUs also increases modularity, as the RTU can be replicated and the pin-out and software modules of the OBC are less dependent on the specific satellite. Furthermore, discrete observation signals with direct connections to the OBC imply penalties in mass. The use of RTUs connected to the OBC with a bus (e.g. CAN or MIL-STD-1553) reduces cable mass of the satellite and the pin count of the OBC. An example is the Delfi-C3 satellite, where moving from a centralized to a decentralized architectures based on microcontrollers resulted in a mass saving of 1% [174].

In [175] a plug-n-play platform for small satellites development based on several sizes of the RTU is proposed, containing:

- nanoRTU (32×32×6.2 $mm^3$, 0.2 W) for temperature sensors, electrical power switches, magnetic torque control, active heat control, solar arrays and batteries;
- microRTU (70×30×10 $mm^3$, 1.5 W) for propulsion control, mass memories and Telemetry and Command (T&C) communication.



For this type of applications, where limited performance is required, general-purpose processors (described in Sec. 1.4) are usually considered not suitable due to high power consumption, large area, presence of caches and speculation. The ARM Cortex-M0+, a 32-bit 2-stage single-issue processor, is an example of a state-of-the-art proprietary implementation employed for terrestrial applications with similar requirements. In the past, ESA has evaluated the ARM Cortex M0+ for a ProASIC3-based microcontroller both without modifications and applying FF-TMR together with safe finite state machine encoding (and reset for unexpected states), employing automated tools [92].

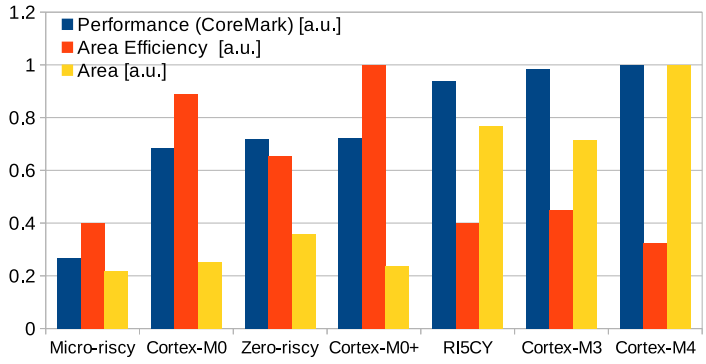


Figure 2.2: Performance, area and area efficiency for several implementations targeting microcontrollers. The data is derived from [110], except for the performance of the Cortex-M3 and Cortex-M4 (from the vendor and [107]).

In [110], several RISC-V processors from the PULP platform are described and compared to state-of-the-art ARM cores, Cortex-M0+ included. Fig. 2.2 shows comparisons in terms of performance (CoreMark/MHz), area (kGE) and area efficiency (CoreMark/MHz/kGE) of the Zero-riscy (2 stages, based on the RV32EC subset), Micro-riscy (2 stages, based on the RV32IMC subset), RI5CY (4 stages, based on the RVC32IM subset plus non-standard extensions), ARM Cortex-M0+, ARM Cortex-M0 (3 stages, an older implementation targeting the same applications of the Cortex-M0+), ARM Cortex-M3 (3 stages), and ARM Cortex-M4 (3 stages). Each parameter is normalized to the maximum value found from the comparison. Comparison between ARM cores and cores from [110] in terms of power are not reported here, as they require power measurements with the same technology, operating voltages and clock frequency. Instead, area can be easily compared using Gate Equivalents (GE), which allows a technology-independent measure of the area. Furthermore, area efficiency is even more relevant for space processors, as a larger area typically implies a larger probability of soft errors. This aspect will be investigated in further detail in Chapter 3.

The considered microcontroller implementations can be binned in three groups according to performance: the low-end implementations (Micro-riscy), the mid-end implementations (Zero-riscy, ARM Cortex-M0, ARM Cortex-M0+) and the high-end implementations (RI5CY, ARM Cortex-M3, ARM Cortex-M4).

The mid-end implementations generally achieve high area efficiency in terms of CoreMark/MHz. While the ARM Cortex M0+ has the highest area efficiency, it is remarkable that new developments can be based on an open and free implementation (like Zero-riscy) developed by academia with similar performance as state-of-the-art proprietary processors.

The Micro-riscy is optimized for less intensive calculations, defined in [110] as "pure control code", as opposed to the CoreMark based on arithmetic calculations. The SoC described in [176] uses the Micro-riscy for power management. In [121], the use of RV32E instead of RV32I is expected to save 25% of area, while Micro-riscy saves more (38.6%) compared to the Zero-riscy mainly because it removes also the M extension (no hardware multiplier and divider) [110]. It should be noted that while this leads to a large improvement in terms of area, it causes large penalties in terms of CoreMark/MHz performance (-62.7%) with smaller improvements in power consumption (-10.9%). The main reason is that power consumption is dominated by the pre-fetch buffer accessing the memory that remains the same in both designs [110]. For this reason, the power efficiency of the Micro-riscy is significantly lower than the one of the Zero-riscy in terms of CoreMark/W (-57.5% at 0.8 V and 55 MHz). It should be noted that RISC-V specifications allow the M extension to coexist with E [121] (this would increase substantially area efficiency and performance or Micro-riscy), while they do not allow F extension to coexist with E because the area of a FPU is typically so large that the saving in area due to 16 general-purpose registers instead of 32 would be marginal.

Both Micro-riscy and Zero-riscy employ the C extension to reduce code size[11]. Another extension which would improve performance of these implementations is the RISC-V extension for bit manipulation (B), planned in the current specifications [121]. The B extension would reduce considerably code size and increase performance of typical operations on single bits in embedded software, by providing for instance instructions to read, write, clear or set a certain subset of bits from a register.

RI5CY includes the extension described in Sec. 2.1.3. It is an implementation targeting control applications requiring DSP capabilities, similarly to the ARM Cortex M4 (which also add DSP extensions to the ARMv7 ISA). RI5CY is 6.1× faster than the Micro-riscy for the 2D-convolution (2D-conv) integer kernel[12] employed as a benchmark in [110]. While the area efficiency for the RI5CY is lower than the one of the Micro-riscy when running CoreMark, RI5CY is 2.83× more area-efficient when running the integer kernel. DSP kernels sometimes require floating-point operations. For this reason, the ARM Cortex M4 can optionally include a FPU (Cortex-M4F), and in [176] the RI5CY has been extended with the F extension.

[11] The RISC-V specifications [121] claim that 50%-60% of the RISC-V instructions in a program can be replaced with C instructions, resulting in a 25%-30% code-size reduction. In [177] similar code sizes for RV32EC and ARM Thumb-2 are reported when considering the SPEC CPU2006 benchmark suite. As the frequency of instructions and the number of registers employed vary with the application, [178] defines a non-standard extension to reduce code size for a specific application.

[12] Common low-level operations in linear algebra programs are usually called "kernel" operations.

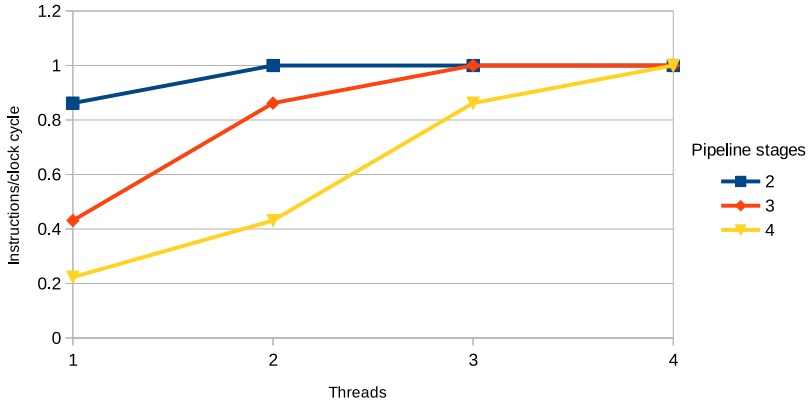


Figure 2.3: IPC for Klessydra when running basic integer kernels. The IPC has been calculated multiplying the MIPS (from [22]) by the clock period, for several pipeline lengths and number of threads. Lines have been added to help visualize the increase of performance with the number of threads.

Defining the optimal number of pipeline stages from the number of IPC given in [110] is not possible, as the three cores are based on different ISA subsets. A more homogeneous set of cores is Klessydra [22], a Very High Speed Integrated Circuit Hardware Description Language (VHDL) model of a RISC-V processor (RV32IM) designed for the PULPino SoC with a configurable pipeline of 2, 3 and 4 stages and supporting up to 4 hardware threads with a simple fine-grained implementation that changes thread every CC (independently from whether the current one is stalled or not) [23].

Fine-grained multithreading for microcontrollers is interesting for two reasons: it allows an efficient handling of SW techniques to increase fault tolerance employing redundant threads [179] and it increases the average number of IPC without the penalties on time determinism due to speculation, as can be seen in Fig. 2.3. As a matter of fact, some processors provide branch prediction to increase the number of IPC, which must be disabled if time-determinism is required. The work in [180] shows instead how to deal with real-time applications and fine-grained multithread on microcontrollers.

IPC is a figure of merit of how much a certain pipeline is used over time and should not be confused with absolute performance: pipelines with more stages may have worse IPC but still shorter executions times because they can achieve higher frequencies and employ more instruction-level parallelism (even if a larger part of it is wasted compared to shorter pipelines). For instance, the 2-stage version achieves an absolute performance of 63 MIPS with two threads (being fully-utilized), while the 3-stage version with two threads achieves 92 MIPS (being utilized at the 86.2%).

However, as it can be seen in Fig. 2.3, the longer the pipeline, the worse is the penalty in terms of IPC if no branch prediction or multithreading is present. The absolute performance of the processors shown in Fig. 2.3 for a single thread is 54 MIPS for the 2-stage version, 46 MIPS for the 3-stages version and 30 for the 4-stage version [22]. This shows that, for time-deterministic implementations without speculation, a short pipeline (2-3

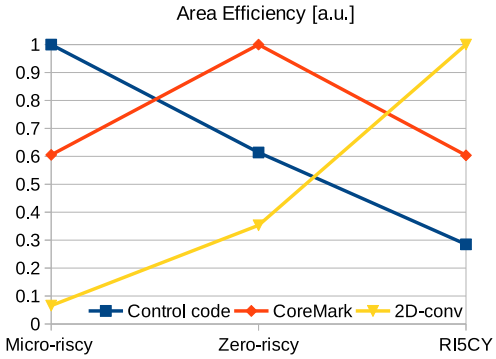


Figure 2.4: Comparison of Micro-riscy, Zero-riscy and RI5CY for different benchmarks (data is from [110]). For each benchmark, data is normalized to the achieved peak. Lines have been added to help visualize trends when moving from an implementation to another.

stages) is the optimal choice also in terms of absolute performance.

The analyzed implementations for microcontroller applications provide interesting insights on the importance of the benchmarks in selecting the most efficient implementation. In Fig. 2.4 the area efficiency of Zero-riscy, Micro-riscy and RI5CY is shown for different benchmarks (representing different target applications).

### 2.2.3. GENERAL-PURPOSE PROCESSORS

Fig. 2.1 shows a payload processor taking care of processing and sending data through the high-data rate modem. Data coming from the instrument can be either stored in the Mass Memory (MM) before its transmission to ground, or processed (off-line) by the payload processor after the data is stored and before the transmission, or processed (on-line) by the payload processor before storing the data in the MM.

While for OBCs RTOSs like RTEMS [156] and FreeRTOS [159] are employed, there is a trend of using Linux in payload processors [181], as in this case often there are no hard real-time requirements for the payload software. The use of Linux allows for an easier software development, because of the availability of drivers, e.g. by using Secure Digital (SD) cards via SATA, and other software modules already available, or to communicate via Transmission Control Protocol/Internet Protocol (TCP/IP) [181]. For these reasons, typically general-purpose processors, already discussed in Sec. 1.4, are employed.

An example of a general-purpose processor for space is the GR740 [30], a quad-core LEON4FT processor originating from ESA's European Next Generation Microprocessor initiative [115] (see discussion in Sec. 1.4).

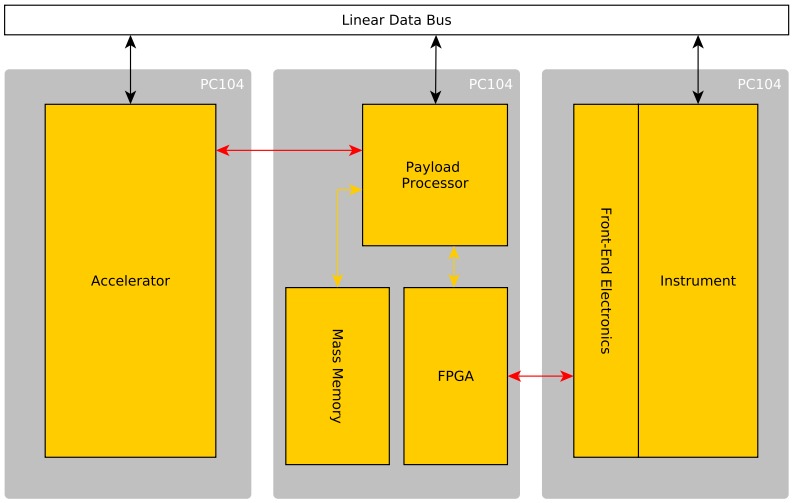


Figure 2.5: Example of use of payload processor with accelerator and FPGA to interface the instrument, similar to the design described in [182]. The linear data bus (typically I2C) connects this subsystem to the rest of the CubeSat (not shown here). In red, point-to-point data busses. In yellow, local busses on the PCB.

### 2.2.4. PROCESSORS FOR ON-BOARD DECISION MAKING

Fig. 2.5 shows an example of on-line processing based on a general-purpose payload processor and an accelerator to enable OBDM[13], similar to the design of Hyperscout-2 [182]. Hyperscout-2, launched in 2020, is the first satellite to demonstrate the feasibility of applying on-board artificial intelligence with a DNN [182, 183]. It is a 6U CubeSat for Earth Observation (EO) with a hyperspectral camera. A DNN is employed to classify images according to the percentage of cloud cover and prioritising their downlink according to their information content [183]. In order to do this, a Myriad 2 [184] is employed as an accelerator to run a 17-layer Convolutional Neural Network (CNN) on the images coming from the instrument (with a total board consumption of 1.8 W during inference [183]). Based on this In-Orbit Demonstration (IOD) mission, a PC104 board (with latch-up protection and components with flight heritage) targeting power requirements below 5 W is being developed [185], which is below 25% of the power typically available in a 6U CubeSat. Using a parallel processor as accelerator to execute on-board compute-intensive workloads that are common in DNNs for Artificial Intelligence, like GEneral Matrix Multiply (GEMM) [186], is an important trend in space embedded systems. In these processors, parallelism is either obtained implementing instructions that operate on more elements of a vector, with DLP and/or PLP.

In [8] several types of platforms (general-purpose processors described in Sec. 2.2.3, manycore DSPs, FPGA-accelerated processors and GPUs) are compared with regard to several benchmark algorithms representative of VBN for space applications (another example of OBDM in space applications). All the devices are based on 28-nm technologies

[13]The role of the FPGA in this design is just to interface the front-end electronics of the instrument.

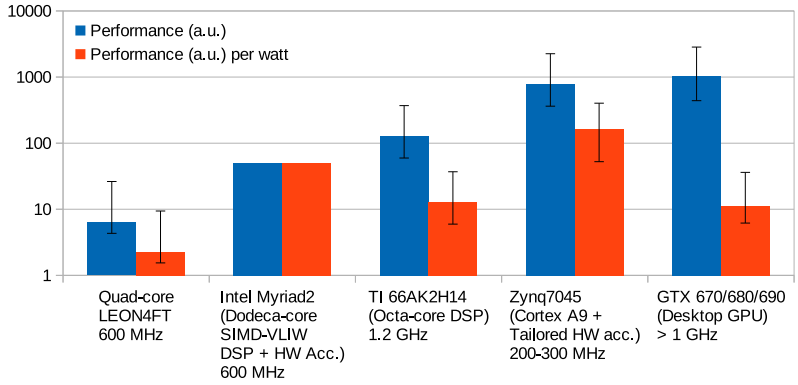


Figure 2.6: Comparison of five different platforms in terms of performance and energy efficiency. All data is derived from [8] and "a.u." stands for "arbitrary unit".

for fair comparison, except the LEON4FT which is based on a larger technology node. The results are reported in Fig. 2.6. Error bars have different meanings depending on the platform: for the LEON4FT the uncertainty stems from the fact that the performance is extrapolated from a single core running at lower frequency; for the 66AK2H14 it stems from slightly different results found in literature; for Zynq it originates from the fact that acceleration resulted in various acceleration factors compared to general-purpose processors depending on the specific algorithm and finally for the desktop GPUs it stems from different chip models [8].

Fig. 2.6 shows that high-end GPUs provide the highest performance but with the worst energy efficiency of all the analyzed platforms. Data from [8] also show that GPUs for mobile devices have performance 13 times lower than the ones for desktops without substantially increasing energy efficiency. Even worse, the estimated best-case energy efficiency of mobile GPUs is even lower than the one of desktop GPUs.

On the other hand, Fig. 2.6 shows that the highest energy efficiency is achieved by processors employing custom FPGA-based on-chip accelerators tailored to the specific application. However, given the niche-sized market for space applications, developing a custom hardware accelerator for each algorithm is too expensive as the cost cannot be spread on large volumes as it is possible for instance for mobiles. The hardware acceleration described in [8] is obtained by connecting the accelerator to the bus/crossbar so that no change to the ISA is required (as is typically done on Zynq platforms). The extendability of RISC-V enables ISA application-specific extensions to increase performance in a specific application for tightly-coupled accelerators, as described in [187]. Also this approach has a high cost, especially because it makes reuse of the software ecosystem more difficult and requires forking from the standard toolchain. An approach to solve this issue is to provide software support for an ISA extension defining a generic interface for accelerators, as the Rocket Custom Co-processor (RoCC) included in the Rocket core [188].

The low performance of a general-purpose processor, like the quad-core LEON4FT, is

easily explained by the fact that this kind of processor relies mainly on ILP and includes little or no DLP. As a matter of fact, the second best platform found in [8] in terms of energy efficiency is the Myriad 2 [184], a manycore SIMD processor. The Myriad 2 processor contains 12 processing cores, each of them being capable of operating simultaneously on integer and floating operands of 128 bits each (potentially from 2 elements of 64 bits each to 16 elements of 8 bits each) [184]. The 66AK2H14 has been included in the comparison to show that a similar choice can also lead to a different performance/energy efficiency trade off. For less demanding applications, parallel processors with lower computational capabilities can be employed. An example of manycore SIMD processors based on RISC-V targeting ultra-low power applications is the PULP platform [125], which provides an array of eight RI5CY cores with two shared FPUs plus a Zero-riscy for control (both of them described in Sec. 2.2.2). The work in [176] describes an ASIC implementation (Mr. Wolf) of the platform on a low-power 40-nm technology, reporting a peak performance of 1 Single Precision (SP)-GFLOPS and 7 GOPS, a maximum energy efficiency of 18 SP-GFLOP/J and 30 GOP/J and a maximum frequency of 450 MHz. Another ASIC implementation of the same platform, the GAP8 processor contains 8 RI5CY processing cores for computations (and a simple processing core for control)[189], each of them capable of executing operations on integer operands of 32 bits (1 elements of 32 bits, 2 elements of 16 bits or 4 elements of 8 bits each [137]). The lack of support for floating point instructions, together with lower computational capabilities (from 345.6 GOP/s [190] to 20 GOP/s [189]), allows the GAP8 to have a peak consumption of 75 mW [189], which is around an order of magnitude lower than the Myriad 2 (800 mW [184]).

Given the comparison between packed-SIMD and vector ISA extensions in Chapter 1, RISC-V dropped the initial proposal of a packed-SIMD standard extension (P)[14] and proposes a vector (V) extension (in the process of being standardized), derived by the development of the Hwacha co-processor [191]. The V extension is intended to be flexible and reconfigurable for different data types and sizes on the run, with the goal to support both implicit auto-vectorization in (OpenMP) and explicit SPMD (OpenCL).

In [18], a manycore MIMD implementation composed of an array of simple RISC-V implementations (not containing any SIMD features) with 4, 8, 16 cores is compared against a Rocket with a Hwacha coprocessor with 1, 2 and 4 vector lanes. The results show that, for similar area, the vector lane solution generally performs better and with lower power consumption. The main reason behind this is that the multiple cores in a MIMD array execute copies of the same instructions across multiple data elements. This approach leads to lower area and power efficiency, as the instruction fetch mechanism is one of the most expensive components of a processor (as already noted in Sec. 2.2.2). An example of MIMD implementations is the Epiphany series. The Epiphany-III has 16 cores and achieves 1 GHz on 16-nm technology, with a declared peak performance of 32 SP-GFLOPS and energy efficiency greater of 16 SP-GFLOP/J [192]. The Epiphany-IV with 64 cores achieves 800 MHz with a declared peak performance of 102 SP-GFLOPS and an energy efficiency greater than 51 SP-GFLOP/J [193]. Finally, the Epiphany-V core with 1024 cores reaches a declared peak performance per CC of 2048 Double Precision (DP)-FLOP/CC [194]. Another manycore implementation is the 16-nm Celerity Soc [188], con-

[14] However, there is still an interest in packed-SIMD fixed-point operations for use in the integer registers of small RISC-V implementations that may lead to the standardization of a P extension.

taining five Rocket RV64G cores, an array of 496 RV32IM cores and a Binarized Neural Network (BNN) accelerator coupled using RoCC. The authors of [188] acknowledged the key role of the RISC-V ecosystem in enabling a small team of junior graduate students to design and tape-out a complex SoC in just nine months. However, they report also to have faced some challenges including limited documentation, lack of reference implementations in an industry standard hardware description language, and the lack of a stable release schedule across the entire ecosystem [188].

A dual-core SMP implementation of Rocket with a single-lane Hwacha (version 3) per core is described in [191] (Raven-3), taped out on a 45-nm process and achieving 1.3 GHz. Performance is measured with a DP floating-point matrix-multiplication kernel (reported to achieve 78% of peak performance). The values found are 16.7 DP-GFLOP/J at 250 MHz and 0.65 V, 1.72 DP-GFLOPS at 550 MHz and 0.8 V (3.13 DP-FLOP/CC), and 4.03 DP-GFLOPS (peak performance for the kernel) at 1.3 GHz and 1.2 V (3.10 DP-FLOP/CC). While Hwacha is a coprocessor to be coupled to a Rocket core, in [195] a stand-alone data-parallel processor (LACore) based on a non-standard extension of RISC-V is described. Even if it was not validated with an ASIC implementation, [195] provides a full benchmarking of the implementation using hand-written optimized versions of the HPC Challenge (HPCC) Benchmark kernels to compare the LACore implementation against a baseline single-issue RISC-V implementation without DLP (both in gem5). The absolute performance of LACore and the speedups obtained against the scalar RISC-V baseline are reported in Table 2.4. The frequency is 3 GHz for the scalar part and 1 GHz for the data-parallel part of the pipeline for LACore and 3 GHz for the RISC-V baseline. LACore substantially increases performance of all the kernels in the HPCC compared to the scalar RISC-V implementation, except for the RandomAccess (as reported in [196]) because it does not imply vectorial operations but measures the peak capacity of the memory subsystem while performing random updates to the system memory (to evaluate the performance of a processor to deal with large and segmented data sets) [197]. The other kernels in HPCC are: HPL (solving a linear system of equations), *dgemm*[15] (real matrix-matrix multiplication), STREAM (which measures sustainable memory bandwidth and the corresponding computation rate for simple vector kernel like Triad[16]), PTRANS (which exercises the communications where pairs of processors communicate with each other simultaneously) and FFT (a complex one-dimensional Discrete Fourier Transform). The efficiency of DLP for matrix operations is confirmed from the data in Table 2.4. As a matter of fact, [195] provides an estimation of the area penalty of 2.53× for LACore over the RISC-V baseline. This means that LACore improves area efficiency up to 31.77× for the *dgemm*, but only up to 1.67× for the FFT. Even if there is no established benchmark suite for OBDM of applications, as opposed to what CoreMark is for embedded applications and SPEC for desktops, the performance of these processors are generally benchmarked like in [195], measuring the number of (FL)OPS or (FL)OP/CC for several kernels.

Performance of a certain kernel in terms of (FL)OP/CC on a certain platform can be

[15] This is a *gemm* kernel operating on DP floating point elements. Analogous subroutines are defined for different data types and the first letter represents the data type (e.g. *sgemm* for simple precision and *igemm* for integers).

[16] The operation executed is $z \leftarrow \alpha x + y$, where $z$, $x$ and $y$ are vectors and $\alpha$ a scalar.

Table 2.4: Absolute performance and relative speedup of LACore compared to a baseline scalar RISC-V pipeline. For each benchmark values for the peak performance (p) and for the largest tested workload (w) are given, depending on the size of the matrix for *dgemm* (p and w:1024) and for PTRANS (p:$2^6$, w:$2^{10}$), and the length of the vector for FFT (p:$2^{12}$, w:$2^{20}$) and STREAM Triad (p:$2^{12}$, w:$2^{20}$) [195].

| Benchmark | DGEMM [DP-GFLOPS] | FFT [DP-GFLOPS] | PTRANS [DP-GFLOPS] | HPL [DP-GFLOPS] | STREAM Triad [GB/s] |
|---|---|---|---|---|---|
| LACore | ~ 16 (p,w) | 1.88 (p)<br>0.55(w) | ~ 0.6 (p)<br>0.15 (w) | 2.55 (p)<br>1.52 (w) | 103 (p)<br>7.25 (w) |
| speedup | 80.4x (p,w) | 4.23x (p)<br>2.1x (w) | ~ 3.5x (p)<br>1.52x (w) | 4.67x (p)<br>5.88x (w) | 13x (p)<br>5.2x (w) |

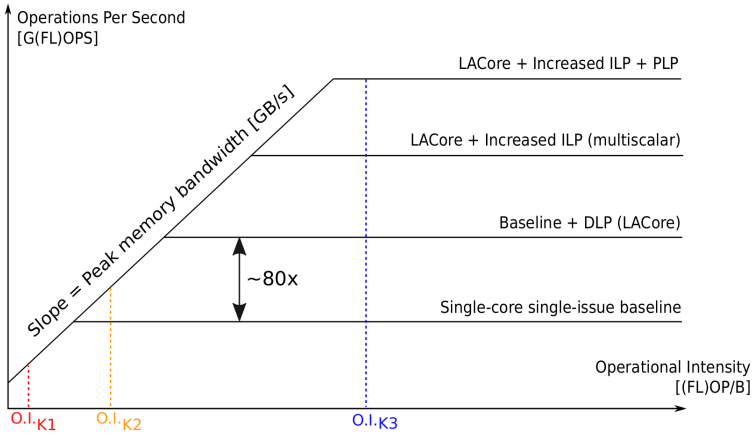


Figure 2.7: Example of roofline models for four platforms employing incremental levels of parallelism for three different kernels (K1, K2 and K3) with different OIs (incremental improvements not to scale).

related to its OI using the "roofline" model [198], with OI being the number of (FL)OP per byte read from off-chip memory for a certain kernel. In this model, a platform is represented by a slope given by the memory bandwidth and a "ceiling" representing the peak computational power, as shown in Fig. 2.7. Fig. 2.7 adds a higher level of ILP (pipeline capable of executing more DLP instructions at the same time, typically require HW replication of the DLP module) and multiple cores (PLP) to LACore as incremental enhancements. The performance of a kernel in Fig. 2.7 is given by the intersection of the vertical coloured line given by its OI and either the slope or the flat "ceiling" of a certain platform. However, the measured GFLOPS are typically lower than the ones at the intersections because of non-idealism not taken into account in this simple model. For instance, this model assumes that memory traffic and computations can be overlapped in time, and this is not always possible.

Kernels intersecting the ceiling are limited by computing capabilities and their performance can be enhanced with additional levels of parallelism (like the kernel K3 is for each of the four platforms presented in Fig. 2.7 and the kernel K2 is only marginally and only for the addition of DLP). Kernels limited by the slope are constrained by the memory bandwidth and therefore are not improved by higher levels of parallelism (e.g. the

incremental use of DLP does not speedup the kernel $K_1$). When considering LACore and the roofline model in Fig. 2.7, RandomAcces is an example of algorithms like K1, FFT an example of kernel like K2 and DGEMM a kernel that can be assumed like K3. DGEMM is typically compute-bound for all sizes and reaches about 95% of the peak performance (flat line) [199].



## 2.3. RISC-V ROADMAP FOR SPACE APPLICATIONS

### 2.3.1. RISC-V PROFILES

In Sec. 2.2 different implementations have been analyzed for each type of processor identified in satellite data systems. From this analysis, several "profiles" of RISC-V processors are defined in Table 2.5, focusing on the features (ISA subsets and non-standard extensions, ILP, PLP), suggested benchmark, expected performance (for each type of processor several level of performance and/or DLP/PLP approaches are proposed) and reference silicon-proven implementations available.

The three profiles identified for microcontroller applications of RISC-V (uC) are essentially derived from the three IP cores presented in [110] for the IoT. The uC-LE is for "pure control" applications (e.g. power management and latch-up protection of COTS components), the uC-ME is for control applications requiring more calculations (e.g. AOCS units) and the uC-HE for sensors producing larger amounts of data (e.g. cameras). Although the LE profile could be benchmarked with the CoreMark, "pure control code" (e.g. a non-preemptive task-scheduler and a driver to interact with peripherals) like the "Runtime" employed in [110] is much more representative of real applications for this profile. As the uC-HE can be seen as a step towards OBDM processors (a single-core, performance can be also measured in terms of (FL)OP/CC and a pessimistic estimation of 2 (FL)OP/CC is reported by dividing the GOPS performance when running integer kernel on the Mr. Wolf [176] by the number of cores.

For uC processors, short pipelines are preferred to achieve high IPC without speculation to avoid penalties on time-determinism. These processors also target deterministic interrupt handling for time-critical applications, with fixed and minimized latency for interrupts. The RISC-V architecture model for managing exceptions, interrupts and registers is simple and suitable for implementations of processors with deterministic timing behaviors (e.g. no windowed overlapping registers, overflow/underflow to be managed).

The expected performance in terms of CoreMark/MHz is similar or greater than the ones expected from the OBCs because shorter pipelines lead to higher IPC. However, lower maximum frequencies in general lead to lower absolute performance compared to microarchitectures for OBCs. Additionally, OBCs require support for RTOSs.

The features and the expected performance of the four profiles of General-Purpose (GP) processors stem directly from Figs. 1.4 and 1.5, with the LE version being a multicore extension of the OBC profile with relaxed real-time constraints. The RV64GC subset is proposed for this profile (as it is the subset targeted at the moment by the Linux ports for RISC-V[17]) and all the three levels of privileges described in Sec. 2.1.1 are re-

[17] While atomic instructions are generally needed especially to boot Linux on multicore processors, a smaller subset would be strictly required, as for instance the need of the C extensions is a requirement introduced by the Linux ports available at the moment of writing.

quired. These processors are intended to support general-purpose OSs but sometimes also RTOSs for hard-real time processing. Development of real-time software becomes more critical by increasing ILP and speculations, thus the different profiles give several possible time-determinism/performance trade-offs, along with area/performance and power/performance trade-offs. The use of the N extension can be evaluated, as delegating some interrupts to user-level processes reduces the overhead both in terms of number of cycles and in terms of cache misses [200].

To enable OBDM, several PLP/DLP approaches for the OBDM profiles are proposed. All of them are related to silicon-proven implementations and for each of the profiles a range of expected FLOP/CC is given. For the OBDM-ManyCore (MC)-LE the lowest expected performance is taken from Mr. Wolf, which has 8 cores and two shared FPUs. Assuming an ideal improvement of a factor 4× for 32 cores and assuming an ideal 4× increase of performance with an FPU per core, an optimistic estimation of 32 FLOP/CC for a 32 cores version can be proposed. An expected performance of 16 FLOP/CC is taken for a more conservative estimation. This estimation is half the peak FLOP/CC given for the 16 core Epiphany in [192]. From this discussion a conservative derating factor of 0.25× can be deduced. For these reason, the target performance of the MC-HE are multiplied by 0.25 compared to the peak values given in [194] for a conservative estimation. For OBDM-Vector Processor (VP) the maximum value found for Raven-3 is divided by 1.5 to estimate performance for single core and multiplied by 2 to have estimation for four cores, in accordance with Fig. 1.5. The manycore profiles can be seen as MIMD extension of the microcontroller profiles, while the VP profile is an extension of the SMP GP processors with the addition of tightly-coupled coprocessors. The HPCC has been selected as reference benchmark suite as it contains popular kernels (e.g. GEMM).

All the metrics employed (e.g. CoreMark/MHz) to estimate the expected performance in Table 2.5 have been chosen to be technology-independent and frequency-independent, so that penalties for fault-tolerant features required in space (like the ones described in [81]) will not cause lower expected performances compared to the "non-FT" reference versions, even if they will cause lower frequency (therefore lowering the absolute performance, e.g. the absolute CoreMark score) and larger area (for instance reducing the number of cores that can fit in a certain chip, thus limiting PLP).

### 2.3.2. FROM THE RISC-V ISA TO FLIGHT-PROVEN PROCESSORS

Sec. 2.2 analyzed several silicon-proven IP cores based on RISC-V. However, a RISC-V based IP core validated for the space environment with a long history of flight heritage like LEON2FT is not available yet. This section describes the different models employed during the development of a space-grade processor. Different models of a processor are typically used depending on the purpose during the development of a system (some of those are more fit to develop software, some others to explore design space of an hardware implementation, some others to estimate performance and some others to synthesize a FPGA or ASIC implementation) and on the Technology Readiness Level (TRL) of the implementation (for instance before tapeout the most mature hardware model is the FPGA prototype).

Table 2.5: Features and expected performances of RISC-V profiles. LE stands for Low-End, ME for Middle-end, HE for High-End and VHE for Very-high-end. IO stands for in-order, OoO out-of order, $n$-w for $n$-way superscalar and SI for single-issue. NSE stands for Non-Standard Extension.

| **Profile**<br>Reference Impl. | **ISA Subset**<br>(optional) | **ILP** | **PLP (N)** | **Benchmark** | **Target Performance** |
|---|---|---|---|---|---|
| **uC-LE**<br>Zero-riscy | RV32E(M)C<br>(B) | 2-3 st.<br>SI IO | 1 | Control code<br>CoreMark | 1 CoreMark/MHz |
| **uC-ME**<br>Micro-riscy/Cortex-M0/M0+ | RV32ICM<br>(B) | 2-3 st.<br>SI IO | 1 | CoreMark | 2.4 CoreMark/MHz |
| **uC-HE/OBDM-LE-1**<br>RI5CY/Cortex-M3/M4(F) | RV32IMC(F)<br>(DSP NSE or V) | 3-5 st.<br>SI IO | 1 | CoreMark<br>Kernels | 3 CoreMark/MHz<br>2 (FL)OP/CC |
| **OBC/GP-LE-1**<br>(LEON3FT/Rocket) | RV32GC | 5-7 st.<br>SI IO | 1 | CoreMark | 2 CoreMark/MHz |
| **GP-LE-N**<br>(LEON4FT/Rocket) | RV64GC<br>(N) | 5-7 st.<br>SI IO | 1-4 | CoreMark | 2-6 CoreMark/MHz |
| **GP-ME-N**<br>ARM Cortex-A7/A8 | RV64GC<br>(N) | 8-13 st.<br>2-w IO | 1-4 | CoreMark | 3-9 CoreMark/MHz |
| **GP-HE-N**<br>BOOM (2-w)/Cortex-A9 | RV64GC<br>(N) | 6-8 st.<br>2-w OoO | 1-4 | CoreMark | 4-12 CoreMark/MHz |
| **GP-VHE-N**<br>BOOM (4-w)/Cortex-A15 | RV64GC<br>(VN) | 6-8 st.<br>>2-w OoO | 1-4 | CoreMark | 5-15 CoreMark/MHz |
| **OBDM-MC-LE-N**<br>Mr. Wolf/Epiphany-III | RV32IM(FD)C<br>(DSP NSE) | 3-5 st.<br>SI IO | 8-32 | Kernels<br>(gemm) | 2-16 (FL)OP/CC |
| **OBDM-VP**<br>Raven-3 | RV64IMA(FD)CV | 5-7 st.<br>SI IO | 1-4 | Kernels<br>(gemm) | 2-6 (FL)OP/CC |
| **OBDM-MC-HE**<br>Epiphany-IV/V | RV64IM(FD) | 3-5 st.<br>SI IO | 64-1024 | Kernels<br>(gemm) | 32-512 (FL)OP/CC |

Models of processors can be classified in simulation and hardware models. Different models provide different levels of abstraction, generating a large spectrum of possible accuracy/speed points for simulation models and cost/representativeness of the flight model for hardware models. Furthermore, hardware models (when available) can be employed to speedup parts of simulation. For instance during software development and hardware prototyping bugs may be triggered on the order of billions of cycles. While on a hardware model executing a billion of cycles takes a dozen of seconds, on a RTL C++ model generated from Chisel 1 billion cycles takes 8 hours [108]. On the other hand, RTL models provide a level of observability usually not achievable with hardware prototypes.

#### SIMULATION MODELS

Simulation models are reported below with a critical discussion on their accuracy, performance and space relevance. The results of this discussion are reported in Table 2.6.

- Instruction Set Simulator (ISS): once the required standard extensions and privilege levels are defined, the profile and the software can be evaluated in an ISS like Spike (the "golden reference" of the RISC-V ISA), QEMU and RV8 [201]. Microarchitecture features and memory behavior are not represented, so the accuracy in terms of performance and fault tolerance is very limited. From the data provided in [201] a penalty factor of 5x over native x86 code can be taken for these simulators. Assuming an x86 processor running native code at 100 GIPS, a performance of 20 GIPS is then expected.

- Event-based simulators: an example is gem5, which provides several possible trade-offs of accuracy/speed in between ISA simulators and cycle-accurate simulations [202] by providing some microarchitectural features (a minimal single issue model, an improved version representing timings of memory, a more detailed pipelined in-order model, and a pipelined OoO model), memory models and emulation or execution of system-level services [203]. Several RISC-V implementations are described in literature employing gem5 for the good accuracy/speed trade-off of this solution during design exploration. For instance, [195] uses gem5 models to compare different architectures instead of using real hardware to remove variables as process technology, clock speeds ad cache configuration in order to compare the processor architectures only. In [204] gem5 is about 40 times faster than Chisel C++ RTL models in terms of IPS.

- Virtual prototypes: an example of virtual prototype for LEON processors is described in [205]. This platform is intended to enable fast full-system simulation, low-level SW development and design-space exploration for multi-processor SoCs. It is based on state-of-the-art design specification languages (such as SystemC/TLM2) and with three abstraction levels: loosely-timed, approximately-timed and RTL. Different levels of abstraction can be employed for each part of the system, according to the required accuracy/speed trade-off. In [206] a RISC-V virtual prototype is described, and in [207] a RISC-V virtual prototype is reported to achieve performance in the range of 30 to 220 MIPS (depending on the accuracy).

- Speed-optimized Cycle-Accurate (CA) models: including implementation-specific and microarchitectural features to estimate correctly timings. Space industry heavily relies

on this tool to develop software in a faster way and to provide WCET analysis. For instance, the use of a cycle-accurate model of LEON (TSIM) is key to provide a light weight, scalable and cost-efficient solution for integration and validation of satellite subsystems and payload equipment containing embedded software [208]. The vendor reports TSIM to reach 60 MIPS on a Intel i7-2600K PC at 3.4 GHz, which is close to the MIPS reported for the OBC in third column of Table 2.3 (-13%). However, the situation gets worse when the hardware prototype works at higher frequencies (e.g. the GR740 is reported to have a maximum frequency of 250 MHz).

- Hardware Description Language (HDL) models (also referred to as IP core or RTL model): this is a crucial model for an implementation, as it is also employed to synthesize the processor for FPGAs and ASICs. RISC-V is often associated with the Chisel HDL because Rocket, the reference RISC-V implementation, is written in Chisel. Chisel has a remarkable potential as a tool to create parameterized architectures, enabling and facilitating academic research [209]. From the Chisel model both a Verilog RTL and a C++ RTL model can be generated. [209] reports that the C++ models is around 8× faster than a simulation with commercial tools on the Verilog RTL considering both compile time and run time, when booting an OS on a Rocket. However, the performance of C++ RTL models generated from Chisel are in the order of some kIPS. The space industry lacks of competence and experience in designing with high-level description languages and modifying the auto-generated Verilog is not a viable option, as modifying autogenerate non-human readable code is risky and damages maintainability. Furthermore, autogenerated code may be not acceptable for dependable applications, as there is is no established design flow in the industry and an additional layer in between hardware description and implementation is added. IP cores for space are typically in VHDL, as it is historically widely adopted by the European space industry [210]. However, as opposed to Chisel which has not been adopted by industry yet, new possibilities like SystemVerilog could also be taken into account as industry and academia heavily relies on this language for terrestrial applications. For instance, the PULP platform is written in SystemVerilog, which greatly simplifies verification and this is becoming a sensible problem for space industry as VHDL was fine to verify simple IP cores but shows its limits as the complexity of the IP cores needed in space increases.

  From the point of view of space applications, IP cores can be classified in two categories:

  – COTS IP cores (or "non-FT"): all the RISC-V IP cores analyzed in Sec. 2.2 are at this level, where specific features for fault-tolerance dimensioned for space applications are not present. LEON processors have "non-FT" versions which can be considered at a similar level of fault tolerance compared to COTS IP cores.

  – FT IP cores: this type of IP cores employs solutions to increase their fault tolerance specifically design for the space environment. An example (LEON2FT) is given in Sec. 1.4.2.

  The differences between the two types of IP cores are becoming less pronounced over time, as larger cache memories with smaller technologies tend to have considerable

SER even in terrestrial applications.

- Netlist: after synthesis a netlist is generated. This is the most accurate simulation model available and can be used for instance also for static timing analysis.



Table 2.6: Simulation models and their features. Speed, accuracy and space relevance are compared using a coarse scale from Very High (VH) to Very Low (VL). A finer scale using "+" and "-" is used to indicate smaller differences.

| Sim. Model (example) | Speed | Accuracy | Space Rel. | Typical use |
|---|---|---|---|---|
| ISS (Spike) | VH | L | M | ISA reference model |
| Modular platform (gem5) | M+ to M- | M- to M+ | L | Design exploration (processor arch.) |
| Virtual Prototype (SoCRocket) | H to M | M- to M+ | M | Design exploration (SoC level), SW dev. |
| Speed-optimized CA (TSIM) | H | H | H | SW development |
| High-level description (Chisel) | L | VH- | L | Parametric and fast design |
| RTL (VHDL/SystemVerilog) | L | VH | H | Design, functional simulation, synthesis |
| Netlist (Verilog) | VL | VH+ | H | Gate-level simulation (e.g. timings) |

### HARDWARE MODELS

This section describes the different types of hardware models typically available for a processor in space applications.

- FPGA: RISC-V processors for space should be portable to a large spectrum of FPGAs, like it is possible today with the LEON processors. Characteristics of FPGAs employed in the development of space systems vary depending on the purpose of the model (e.g. prototype vs flight system). FPGAs can be classified according to the technology employed in the configuration memory and according to whether they are qualified for use in space applications or not:

  - COTS SRAM-based (e.g. Xilinx Spartan6, Xilinx Kintex UltraScale and Xilinx Artix7): these are the most popular type of FPGAs in terrestrial applications, providing the best solution for rapid prototyping and evaluation because of their mature toolchain, large capacity and availability of several FPGA-specific IP cores from the vendors. However, they have typically high power consumption (a design based on one of the latest COTS FPGAs dissipates 4 W [211]) and the configuration memory is vulnerable to SEUs [212]. Typically hardware and information redundancy are required in the design of a SRAM FPGA, along with scrubbing of the configuration memory to avoid error accumulation in the configuration memory, which comes at a great cost in terms of power consumption (e.g. +265% in [213]). However, SRAM-based FPGAs require an additional level of protection when operating in space compared to an ASIC because large part of the sensitive area is composed of the configuration memory. For the Ultrascale series, Xilinx provides the SEM IP core for SEU detection and correction in the configuration memory with additional FI capabilities to allow the user to evaluate the dependability of the final solution [214].
  - COTS flash-based (e.g. Microsemi IGLOO2, Microsemi ProASIC, Microsemi PolarFire, Microsemi SmartFusion2): although they can leverage a smaller user-base

compared to SRAM FPGAs, Flash-based FPGAs seem the best fit for space application, because of their lower power consumption [215] and immunity to upsets in the configuration memory [216]. Furthermore, in [215] it is shown that a design dissipating 23.70 mW on a COTS Flash-based Microsemi SmartFusion2 dissipates 245 mW (10x) on a COTS SRAM-based Xilinx Artix-7. However, as these FPGAs are not explicitly validated for the space environment, other problems may arise, like SETs in critical parts of the SoC (e.g. in Phase-Locked Loops [217]), SELs, low-dose TID failures, relatively high SEU in the user memory. For instance, [218] reports the results of heavy ions and proton tests for SEE of an IGLOO2. Several instances of "high current" status triggered by particles (suspected SELs) are reported (while for instance the space-grade RTG4 is SEL immune [219]).

- Space-grade SRAM-based (e.g. Xilinx Virtex 5QV): these devices are typically well characterized in terms of effects of radiation and provide meaningful improvements over COTS SRAM-based ([220] claims a 1000× improvement in SEU hardness of the cells of the configuration memory). However, SRAM cells of the configuration memory and user memory are still sensible to SEUs and scrubbing of the configuration memory is then required. For instance, in [220] 3.80E-10 upsets/bit/day in GEO at an altitude of 36,000 km are predicted for the configuration memory. Furthermore they are still more power-hungry than their flash-based counterparts, as for instance the power consumption of the RHBD Xilinx Virtex-5QV SRAM-based FPGA is in the range of 5–10 W, while the power consumption of the comparable rad-tol Microsemi RTG4 is in the range of 1-4 W [8].
- Space-grade flash-based (e.g. Microsemi RTG4): flash cells are inherently immune to SEUs, but user memory is still vulnerable to SEU. This means that the same level of fault tolerance employed for ASICs can be considered sufficient. In [219] the tolerance to TID effect of the RTG4 is compared with the one of SmartFusion2 and a significant improvement is found, with a TID tolerance tested up to 160 krad. No SEL were observed for a LET of 103 (MeV $cm^2$)/mg at 100°C and the SER of the sequential elements (FFs with TMR) is at least three orders of magnitude better than the simple FFs in the SmartFusion2. No MBUs were observed in the block RAM of the RTG4 because of interleaving.
- Space-grade antifuse (e.g. Microsemi RTAX-S): the configuration is fixed, so radiation cannot change configuration memory at all. This comes with large penalties in terms of frequency and capacity compared to space-grade flash-based FPGA. The vendor reports the RTAX-S to achieve a frequency 3 times lower than the one of the RTG4 and to have a capacity in terms of logic elements 7.5 times smaller.

- ASIC prototype: a silicon-proven processor is typically seen as a mature processor which can be evaluated for inclusion within systems. In space the situation is more complicated, and for this reason an ASIC demonstrator is even more important considered the additional difficulties in the use of FPGAs.
  - Commercial technologies: while in the past ASICs based on commercial technologies were susceptible to destructive SEE events (SEL and Single Effect Hard Errors [221]), newer SOI technologies are immune to SEL [222] and typically susceptible to

hard failures only for TID. On the other hand, many sub-20-nm ASICs employ FinFET technology, which has an increased SEL vulnerability compared to bulk Complementary Metal–Oxide Semiconductor (CMOS) technologies with the same voltage [223]. The SER caused by SEUs and SETs must nevertheless be evaluated. An example of an ASIC prototype based on commercial technology was the AT697E, an ASIC prototype of the LEON2FT from Atmel based on a 180-nm CMOS process.

  - RHBD technologies: in Europe many ASICs are based on the RHBD DARE library for the UMC 180-nm CMOS technology [224] and the newer C65SPACE (65 nm) by STMicroelectronics [225]. The AT697F was based on the ATC18RHA (180 nm) radhard library, providing an increase in resilience to TID effects from 60 to 300 krad compared to the AT697E.

- Space-grade component: the space qualified version of a processor typically employs a ceramic and hermetic package, extended temperature range (from -55 °C to 125 °C) and extended qualification flow (according to QML-V and QML-Q space grade flows [226]). Radiation performance may vary:

  - The rad-tol ATmegaS128 has a LET threshold for SEL of 62.5 (MeV $cm^2$)/mg at 125 °C and TID tolerance up to 30 krad [227].
  - The rad-hard AT697F (from the same vendor) is reported to have a LET threshold for SEL of 95 (MeV $cm^2$)/mg at 125 °C and TID tolerance up to 300 krad [228].

- Flight-proven component: according to [229] a component has achieved flight heritage after two years of nominal performance under nominal mission conditions. The AT697 flew for the first time in 2008 and remained fully functional for more than 5 years, achieving flight heritage, which was key for broad market acceptance of the LEON2FT [230]. Together with the LEON processors, also IP libraries like GRLIB have been proved successfully in space. Therefore integration in GRLIB would enable RISC-V processors to reuse all the interfaces already available for LEON-based SoCs with flight heritage like SpaceWire, MIL-STD-1553B, CAN, etc.

Given the discussion on the models adopted by the space industry, a roadmap to bring a "non-FT" or COTS IP core (e.g. LEON2 or Rocket) into a flight-proven ASIC component (e.g. AT697F) is given in Table 2.7, summarizing with examples which steps can be taken. A more detailed roadmap may also evaluate intermediate hardware models (e.g. space-grade FPGAs and ASIC prototypes) for flight.

## 2.4. PRIORITIZATION AND SYNERGIES

The work required to answer RQ3 and RQ4 for each identified profile is more than can be handled in a single PhD. For this reason, it was necessary to exploit synergies with parallel developments, and selecting the most promising profiles for each RQ. During this PhD, Cobham Gaisler was working on the development of the NOEL-V Platform. This platform covers the OBC profile and all the GP and uC profiles identified in this chapter, with some novelties that will be described in Sec. 6.1. Cobham Gaisler released open-source the non-FT version of the NOEL-V in their GRLIB IP Core library approximately

Table 2.7: Simulation and hardware models from a COTS IP core to a flight-proven solution.

| Model | Strengths | Weaknesses |
|---|---|---|
| COTS IP core (LEON2, Rocket) | Many available, large user base | Fault tolerance to be assessed (and improved) |
| FT IP core (LEON2FT) | Designed keeping into account SEEs | Larger area and power consumption, lower maximum frequency |
| SRAM FPGA (LEON3FT on Virtex-5/5QV) | Large capacity and rapid prototyping (COTS versions) | Config. memory vulnerable to SEU (even for space-grade versions) |
| FLASH/Antifuse FPGA (LEON4FT on RTG4) | Configuration memory not vulnerable to SEU | Less capacity (compared to COTS SRAM FPGAs) |
| ASIC proto. (comm.) (AT697E) | High performance, large capacity, Process is immune to SEL (SOI) | Risk of destructive SEEs (bulk/FDSOI) and/or low TID failures |
| ASIC proto. (RHBD) (AT697F before qualification) | Representative of final performance and radiation-hardness | Larger area, lower frequency compared to commercial technologies |
| Space-grade comp. (AT697F after qualification) | Mature solution (TRL=8) | More expensive than ASIC prototypes, later on the market (~ years) |
| Flight-proven comp. (AT697F after flight) | Proven solution (TRL=9) | At least 2 years needed for flight-proven (more than two-years old technology) |

halfway this PhD (July 2020), while the FT versions are expected to be proprietary like for LEON processors. Therefore, the most critical areas to research on during this PhD work were identified to be on the one hand the analysis of the vulnerability of the GP profiles as a case study to provide a methodology to turn a open-source non-FT IP Core into a FT IP core in a cost-effective way, and on the other hand the extension of these profiles to cover also the OBDM profile with the NOEL-V platform. The final design of the NOEL-V platform extended to address the OBDM profile has been delivered to Cobham Gaisler, and it is planned to be included in GRLIB. Furthermore, part of the work identified in this chapter has been carried by master students under the supervision of the PhD candidate. The results of these activities will be mentioned in Chapters 3 and 4, when RQ3 and RQ4 will be addressed.

## 2.5. SUMMARY

Several profiles of processors based on RISC-V, which are expected to improve satellite data system architectures compared with state-of-the-art processors for space, are identified. This chapter discussed how the open nature and modularity of the RISC-V ISA allow designers to implement the optimal microarchitecture for different applications, ranging from low-performance/low-power microcontrollers to OBCs to processors for payload applications to high-performance processors for machine learning and digital signal processing. The large number of RISC-V implementations described, some of them with very different target applications, also shows how RISC-V allowed an unprecedented amount of research in a relative short time on a single ISA. Furthermore, studies on the same open ISA can be easily employed to compare and evaluate microarchitectures to solve currently unaddressed needs in space systems. The introduction of RISC-V in space will contribute to providing a range of alternatives to proprietary solutions to enable new capabilities in satellites data systems, as concerns are growing about monopolistic positions in the embedded market. This chapter provides knowledge to choose proprietary solutions when strictly needed to achieve a certain level of performance or efficiency and choose several degrees of openness when possible, en-

abling the next generation of space data systems to be based on an effective mix of the two approaches. A flexible roadmap to bring RISC-V processors from COTS IP cores to flight-proven components is proposed, based on the models typically employed in the development of space processors.

# 3

# Cost-Effective Redundancy to Mitigate the Effects of Soft Errors

.

*Yet mark his perfect self-contentment, and hence learn this lesson,*
*that to be self-contented is to be vile and ignorant,*
*and that to aspire is better than to be blindly and impotently happy.*

Edwin Abbott Abbott, Flatland (1884)

*This chapter provides a comprehensive framework to assess efficacy and cost-effectiveness of redundancy to increase dependability of the microarchitecture of processors for space applications. The focus is on soft errors, which dominate the failure rate of processors in space. Error, failure and propagation models are proposed, starting from information available in literature, already presented in Chapter 1. These models are employed to estimate the failure rate due to soft errors for processor profiles identified in Chapter 2. This detailed white-box analysis is possible only for open-source Intellectual Property (IP) cores. In this work it will be applied to several open-source IP cores based on the RISC-V Instruction Set Architecture (ISA). For these case studies, relevant types of redundancy described in literature for space processors will be evaluated in terms of their cost-effectiveness and expected in-orbit behavior.*

Parts of this chapter have been published in [231] and [232].

## 3.1. INTRODUCTION

This chapter develops a *comprehensive* framework at processor-level[1] to assess and mitigate the soft error vulnerability of processors in a *cost-effective* way. This means that the cost of the redundancy required for such mitigation is evaluated and compared to the reduction in soft error vulnerability obtained with them.

The need of this framework comes from the fact that works in literature typically describe in great detail specific aspects of the vulnerability of specific hardware structures and how to address soft error vulnerability of specific units in a processor (e.g. register files [234], data [80] and tag [235] array in caches). This sub-processor approach is dictated by the extensive work required to build a relevant test setup and to the number of experiments required to get meaningful statistics. In this chapter those works will be complemented by combining their results together, using them to develop a comprehensive framework that the reader can reuse and readapt to its own designs or when evaluating an open-source IP core.

Moreover, all the aspects related to the microarchitecture will be developed further and a model will be established. This will give more insights on how to evaluate open-source IP cores and how to enhance their dependability in a cost-effective way. This framework will help to evaluate the redundancy to be implemented on a processors, as fault-tolerant processors are presented without justifying the choice of their redundancy, like in [81] and [98]. Even if a few comprehensive frameworks [236, 237] were proposed in recent years (2016-2019), the present framework differs for two reasons:

1. it is presented step by step to the reader (see Table 3.16), without using proprietary black-box tools. In this way, the reader can therefore implement the framework for their own designs and contribute to its extension in a straightforward way.

2. it has a wider scope, as it includes definition of threat models from the space environment, and considerations on availability and validation.

## 3.2. OUTLINE

To introduce the problem, the first part of this chapter follows the error from its generation to the occurrence of the service failure (as shown in Fig. 3.1). In Sec. 3.3, first an error model is proposed (Sec. 3.3.1), then the effects of the defined error models are analyzed up to the service interface (Sec. 3.3.2), and finally the application-dependent effects of errors at the service interface are analyzed (Sec. 3.3.3).

The second part of the chapter follows instead the steps of a typical design flow for a fault-tolerant processor. In Sec. 3.4 a quantitative model to identify the most vulnerable units of processors is presented (Sec. 3.4.1) and it is applied to four different processor designs (Sec. 3.4.2). Sec. 3.5 then analyzes several types of redundancy and discusses their cost-effectiveness. Sec. 3.6 discusses aspects related to validation and in-orbit expected behavior. Finally, Sec. 3.7 summarizes the main findings of this chapter.

[1] That is, including caches but excluding peripherals, interconnects, interfaces, off-chip memories and main memory. However, processors are typically included in a SoC together with peripherals and memories. To further extend this framework, the reader can refer to [233], which estimates the impact of other subsystems of SoCs.

## 3.3. MODELLING THREATS

Fig. 3.1 shows how threats[2] interact with a processor. A failure is a deviation from the expected behavior of the service provided at the service interface [39], and it is caused by one or more deviations from the correct state of the system (errors). The cause of the error is called fault [39].

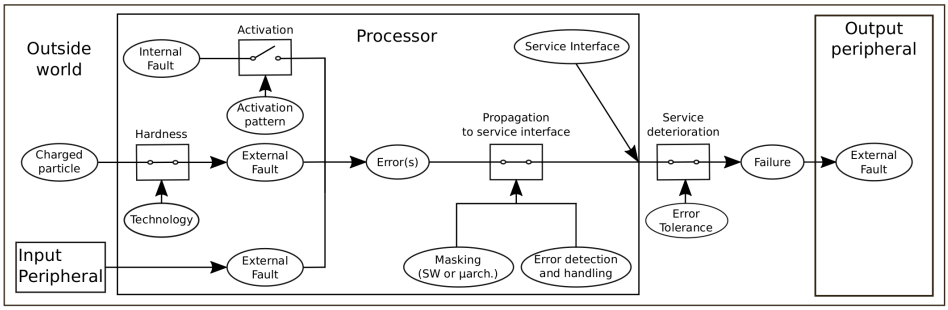


Figure 3.1: Typical interactions of threats with a processor providing a service to an output peripheral.

### 3.3.1. FAULT AND ERROR MODELS

Given the discussion on the errors generated by radiation in Sec. 1.3.1, the *SER* of the processor will be estimated as $SER = SER_{SEU} + SER_{SET}$, which can be rewritten as:

$$SER = \lambda_{ev} \cdot N_{eq} \tag{3.1}$$

where $N_{eq}$ is the number of reference sequential elements that would produce the same SER given a certain $\lambda_{ev}$, i.e. the event rate. Comparing this equation with Eqs. 1.8 and 1.9 yields:

$$N_{eq} = N_{SRAM} \cdot RV_{SRAM} + N_{FF} \cdot RV_{FF} \cdot SV_{FF} + \frac{A_{comb}}{A_b} \cdot SF_{SET} \cdot RV_{comb} \tag{3.2}$$

In Table 3.1 the parameters of the proposed model for 5 different types of technologies are reported: Low Criticality (LC), MBU Dominated (MD), High Criticality (HC), SET Dominated (SD) and Average Criticality (AC) . The effect of the fraction of MBUs on the final failure rate will be taken into account as described in Sec. 3.4.3. However, it should be noted that for MBUs it is not possible to define worst cases and best cases and this will be always so for each type of redundancy explored in the following sections. Therefore, as a metric to define the best, average and worst case in Table 3.1, the total percentage of MBUs is considered. This is the reason why, for instance, the percentage of $MBU_{(\geq 4)}$ is greater for AC than for HC.

The parameters in Table 3.1 describe a three-dimensional space of technologies, as shown in Fig. 3.2. Four of the selected technologies (LC, MD, HC, and SD) are edges of a solid in this space and one is the average case (AC). As a matter of fact, technologies not

[2] In [39], the term 'threats' refers to faults, errors, and failures.

Table 3.1: Parameters of the error models for space processors (data derived from [47, 67, 68, 73]) for different types of technology defined in Sec. 3.3.1: Low Criticality (LC), Average Criticality (AC), High Criticality (HC), SET Dominated (SD) and MBU Dominated (MD). The values are taken from the discussion in Sec. 1.3.1.

| Technology | LC | AC | HC | SD | MD |
|---|---|---|---|---|---|
| $SF_{SET}$[%] | 0 | 15 | 30 | 30 | 0 |
| $SF_{FF}$[%] | 97 | 82 | 67 | 67 | 97 |
| $SBU$[%] | 95 | 45 | 24 | 95 | 24 |
| $MBU_2$[%] | 4 | 18 | 52 | 4 | 52 |
| $MBU_3$[%] | 1 | 10 | 3 | 1 | 3 |
| $MBU_{\geq 4}$[%] | 0 | 27 | 21 | 0 | 21 |
| $MBU_{even}$[%] | 4 | 45 | 73 | 4 | 73 |



only affect $\lambda_{ev}$ (rate of events), but with the relative contribution of SEUs, SETs and MBUs (quality of events) they also determine which redundancy is more effective. The rest of the edges of the solid are defined considering only a finite range of $\lambda_{ev}$ ($10^{-12} - 10^{-6}$), identified according to average values experienced during several missions (Sec. 1.3.1). The choice to not consider extreme conditions such as *worst week* and *worst 5 minutes* in GEO is due to the analysis in Sec. 1.3.3, which shows that the probability of having more than one upset in a single EDAC-protected word is negligible for relatively small memories (e.g. < 2 MiB).

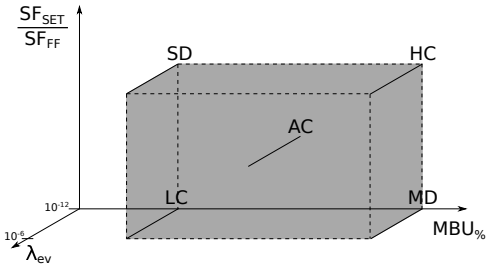


Figure 3.2: Technology space considered in this work, delimited by dashed lines. 'Edge' and 'average' technologies in black solid lines.

### 3.3.2. ERROR PROPAGATION TO THE SERVICE INTERFACE

Errors generated by a fault not masked at the technology level can be masked during their propagation to the service interface (even when not considering redundancy) at microarchitectural level (e.g. the error does not influence the behavior of the processor) and at software level (e.g. an error which affects a bit in an unused instruction or is used only by a dynamically dead instruction[3]), as shown in Fig. 3.3. When the error is masked, the application terminates normally and output pins (and files) do not differ

[3] A dynamically dead instruction is an instruction which outputs are not used by any other instructions and that does not actually influence the output of the processor. [238]

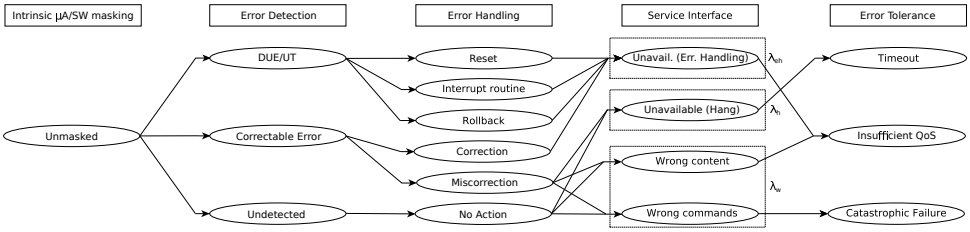


Figure 3.3: Propagation of errors to the service interface and effects on the system service. The failure rates $\lambda_{eh}$, $\lambda_h$ and $\lambda_w$ identified at the service interface will be introduced in Sec. 3.4.1.



from the fault-free execution[4].

When redundancy (see Sec. 1.3.2) is employed, along with the intrinsic microarchitectural and software masking, error detection and handling are also possible. The possible outcomes of error detection and handling are:

- Correctable error: the error detection and handling mechanism proceeds to correct the error (correction). However, when more errors than expected are present, the correction can be wrong (miscorrection [87]).
- Detected Uncorrectable Error (DUE): the error detection and handling mechanism is not able to correct the error, although it is able to detect the error and to prevent it from propagating to the service interface [240]. The reaction to a detected DUE (e.g. rollback) may cause penalties in terms of availability.
- Unexpected Termination (UT): its effect on the error propagation is the same as a DUE, but it is typically caused by the OS and software [241] instead of hardware. For instance, a process may terminate abnormally due to built-in protections (memory access violation, kernel panic, and arithmetic exception) triggered by an anomalous behavior [242].
- Undetected: in this case the redundancy employed fails to detect the error during its propagation and no action is taken.

### 3.3.3. SERVICE INTERFACE AND ERROR TOLERANCE

The system service defines the service interface at which the service is to be provided and which outputs of the software (e.g. variables directly mapped to a failure) and hardware (e.g. signals to other subsystems) will be able of propagating the errors. An error, when propagated to the service interface, can generate wrong data, wrong commands or unavailability of the system (Fig. 3.3). The unavailable state can be further broken down into a state where the unavailability is due to the intrinsic vulnerability of the processor (i.e. hang) and a state where it is due to error handling.

[4] In [239], masked cases are instead classified in two different categories depending on whether the final architectural status is different from a fault-free execution (referred to as *Output Not Affected*) and those where it is the same (referred to as *Vanished*).

### INTRINSIC ERROR TOLERANCE

In many works, wrong data and wrong commands on the output are both assumed to be a failure, calling this Silent Data Corruption (following the terminology of [240]). However, this is not always the case, as some services are inherently tolerant to wrong data at the service interface. In [243] a system is defined as error tolerant with respect to a service, if the system produces *acceptable* results to the end user according to a certain Quality of Service (QoS) even when errors are propagated to the outputs of the system. The system fails due to insufficient QoS instead when the QoS is below a certain threshold ($QoS_{thr}$). For instance, in a system providing edge detection for images, the QoS is defined in [244] as the peak Signal-to-Noise Ratio (SNR) when comparing the corrupted and correct images and $QoS_{thr}$ is set to 10 dB. More complex services have a more complex definition of acceptable quality. For instance, in Moving Picture Experts Group (MPEG) video encoding, to exploit the similarities among contiguous frames, there are three types of frames: *I* frames (which contain all the information to be decoded on their own, providing low compression rates), *P* frames (which require data from previous frames to decode, achieving higher compression rates compared to I frames), and *B* frames (which can exploit both data from previous and following frames to be decoded, achieving even higher compression rate compared to B frames) [244]. In general, the loss of *B* and *P* frames can be compensated by the decoder, while the loss of an *I* frame will result in a substantial quality degradation. In [244] a frame is considered bad if the SNR (compared to the correct frame) is more than 2 dB for *I* frames, 4 dB for *P* frames and 6 dB for *B* frames. The QoS in [244] is then defined as the percentage of good frames and $QoS_{thr}$ is then set to 10% of bad frames. An example of even more complex service is inference for image classification. In this case the $QoS_{thr}$ is defined as the difference in confidence of the top ranked element compared to the top ranked element of the fault-free execution [245]. In addition, the concept of QoS is introduced also for the catastrophic failures, which in this case is when the top ranked element differs from the golden execution. As a matter of fact, a differentiation is done between the case where the top ranked element is at least a 'good candidate' (i.e. one of the first 5) in the fault-free execution and the case where it is not.

In [244] it is shown that in order to fully exploit the concept of error-tolerance, control operations (defined as those which can change the control flow in the software and therefore potentially generate wrong commands at the outputs) must be identified and protected. Catastrophic failures are avoided both for the Smallest Univalue Segment Assimilating Nucleus *(SUSAN)* algorithm (edge detection) and *MPEG* (MPEG encoding) when errors are not injected in control operations (while some other benchmarks have catastrophic failure rates up to 19% even when errors are not injected in control operations). When control operations are protected, more than 100 errors per second had to be injected in *SUSAN* to show any frame loss due to the SNR being too low. *MPEG* had instead about 2% loss at 10 errors per second. Both the error rates injected for *SUSAN* and *MPEG* are pessimistic for space applications, as the error rate for processors in space is several order of magnitude lower (in Sec. 3.4.2 the maximum SER found is around three errors per day at the highest upset rate considered). *MPEG* crashes disabling protection for control operation, while for *SUSAN* disabling protection leads to very poor fidelity of the output. This can be attributed to the relatively small number of control instruc-

tions (less than 9%) in *SUSAN* compared to the higher percentage in *MPEG* (around 50%) [244].

#### EXPLICIT ERROR TOLERANCE

Once models of failures at the service interface are defined, explicit techniques of error tolerance can be employed. One of the most commonly used is the watchdog timer, namely a counter that if not periodically reset by the processor will reset the processor itself [246]. This is represented in Fig. 3.3 with *Timeout* and it is based on the simple model of *Hang* of the processor at the service interface. However, more complex models can be employed, and in [246] also a smart watchdog is proposed. Similarly, in [247] a symptom-based mechanism is employed to reduce the failure rate by 20x over a baseline design without explicit error tolerance.



## 3.4. MODELLING THE VULNERABILITY OF PROCESSORS

Once the models for the threats are defined, the following step is to build a model to identify the most vulnerable parts of the design. A common model found in literature is the AVF decomposition [55].

### 3.4.1. AVF DECOMPOSITION

In order to take into account the masking effects due to software and microarchitecture, in [78] the AVF of a unit is defined as the probability that a fault in that unit of the processor will cause a failure at the outputs of the processor. For this reason, the AVF depends on which event of those described in Sec. 3.3.3 are considered as failures. In this work, the definitions of failures as indicated in Fig. 3.3 (at the service interface) will be employed.

The rate of occurrence of a failure $f$ for the unit $i$ can be modelled as:

$$\lambda_{i,f} = SER_i \cdot AVF_{i,f}. \tag{3.3}$$

For a correct execution, all the units of the AVF decomposition are required to not propagate an error to the outputs of the processor. As a result, units in an AVF decomposition can be thought as a series of components in a reliability block diagram [55]. Assuming that the masking is uniform (therefore not changing the distribution of events) and assuming that failures in different components are independent of each other, the total reliability is given by the product of the reliability of the units composing the processor. The processor-level failure rate $\lambda_f$ for the failure $f$ is then given by:

$$\lambda_f = \sum_i SER_i \cdot AVF_{i,f} = SER \cdot AVF_f \tag{3.4}$$

As $SER = \lambda_{ev} N_{eq}$, the effects of failures on a service for space applications (relatively high $\lambda_{ev}$ and low $N_{eq}$) can be sometimes compared to the effects on services for application with lower $\lambda_{ev}$ and higher $N_{eq}$ (e.g. servers) [55]. Eq. 3.4 can also be written as

$$\lambda_f = \lambda_{ev} \cdot \textstyle\sum_i \hat{\lambda}_{i,f}, \tag{3.5}$$

where $\hat{\lambda}_{i,f} = N_i \times AVF_{i,f}$ is the failure rate normalized to the upset rate per bit. For failures causing wrong outputs or data, the failure rate $\lambda_w$ (Fig. 3.3) is enough to estimate their effect on the service[5].

The impact on the service interface of a certain type of failure $f$ causing unavailability is also determined instead by the duration of the unavailability $T_{u,f}$ each instance of this type of failure causes. If a system is unavailable for a total $T_{\text{Unavailable}}$ during a certain time period $T_{\text{Mission}}$, the unavailability is then defined as:



$$U = \frac{T_{\text{Unavailable}}}{T_{\text{Mission}}}, \tag{3.6}$$

and the availability as Availability $= 1 - U$. Different types of events causing unavailability can be observed:

- Timeout ($\lambda_h$): these events are due to residual $AVF_u$ not protected by redundancy. In the remainder of this chapter it will be assumed that they are addressed employing a watchdog timer that triggers a hard reset (power cycle) when it expires. An order of magnitude for $T_{u,h}$ can be found in [248], where it is assumed to last 5 minutes, as extensive checking (e.g. memory) is required.

- UT ($\lambda_{eh,ut}$): in this case an error handling mechanism causes a UT of a process. When a process is terminated, a possible solution is to use an interrupt service routine for diagnostic and restart of the process. These have typically lower impact than a restart of the whole processor. The work in [249] shows that a process can be restarted with a latency on the order of 10 ms.

- DUE in data without valid copies ($\lambda_{eh,sr}$): in this case (e.g. errors in Write-Back (WB) caches) a DUE requires at least a soft reset (i.e. ending the current processes and booting again). From the work in [250], a penalty of 45 s can be assumed for a soft reset, composed of end time and boot time.

- Rollback to an up-to-date value ($\lambda_{eh,rb}$): when the corrupted data is available in the most up-to-date value, the loss in terms of availability is minimal. For instance, in case of a DUE in a Level1 (L1) cache with Write-Through (WT) policy the data can be read from the L2C, with a penalty of a cache miss [251]. As can be seen in [251], 150 CCs can be taken as a pessimistic estimation for a cache miss and even in this case, assuming a clock frequency of 100 MHz, the penalty is in the order of microseconds (which is in most cases negligible).

- Correction ($\lambda_{eh,c}$): the latency in this case is very short. For instance, the LEON2FT checks the EDAC code on the (scalar) Register File (RF) during the execution phase, writes back errors in the RF with the correct value, flushes the pipeline and restarts from the instruction that reads the operand with the error [81]. This procedure causes typically minimum penalty in terms of stalling (in this case just 5 CCs).

[5] Sometimes, instead of the failure rate, the MTTF is employed to indicate how often a failure will happen on average. The use of an exponential reliability function simplifies further the calculations, as $MTTF_w = 1/\lambda_w$.

- Device-specific rollback ($\lambda_{eh,ds}$): some devices save the old status to rollback to it in case of DUE [102] or they compare the output of three processors and restore the correct status from one of the golden replica [98]. In these cases the penalty in terms of availability is implementation-specific. This aspect will be discussed further in Sec. 3.5.

The unavailability due to each type of events $f$ can be expressed as:

$$U_f = \frac{N_{u,f} \cdot T_{u,f}}{T_{Mission}} = \frac{(T_{Mission} \cdot \lambda_{u,f})\, T_{u,f}}{T_{Mission}} \tag{3.7}$$

where $N_{u,f}$ is the number of times the events $f$ happened during the mission and $T_{Mission}$ is the total mission time. Therefore, the unavailability of the processor considering all the possible sources $f$ of unavailability is:

$$U = T_{u,h} \cdot \lambda_h + \sum_F T_{u,eh,f} \cdot \lambda_{eh,f} = \lambda_{ev} \cdot \hat{U} \tag{3.8}$$

### VULNERABILITY IN TIME: ACE ANALYSIS

More insight can be gained on the meaning of the AVF by considering how AVF is estimated in [78], i.e. considering the bits required for an Architecturally Correct Execution (ACE). A bit is an ACE-bit when changing its value will cause the error to propagate to the service interface and it is an un-ACE bit otherwise. A bit typically changes from ACE to un-ACE and vice versa during program execution, as shown in Fig. 3.4 for bits in a location of the RF.

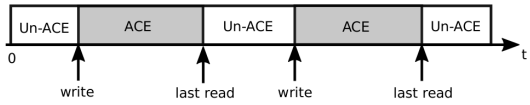


Figure 3.4: Fraction of time bits a location in the RF is in ACE (gray) and un-ACE (white). Between *write* and *last read* an arbitrary number of reads can happen.

At any instant in time, the AVF can be expressed as the number of ACE bits in a structure $N_{ACE_i}$ over the total number of bits in the structure $N_i$:

$$AVF_i(t) = \frac{N_{ACE_i}(t)}{N_i}. \tag{3.9}$$

The average AVF can then be defined as the average number of ACE-bits in a certain timespan. Using Little's law [78], the average number of ACE-bits within a structure (e.g. instruction buffer or execution unit) can be written as the product of the arrival rate (bandwidth $B_{ACE_i}$) of ACE bits and the average time of persistence in the structure (latency $L_i$):

$$AVF_i = \frac{N_{ACE_i}}{N_i} = \frac{B_{ACE_i} \cdot L_i}{N_i}. \tag{3.10}$$

For instance, when considering hardware structures storing or executing instructions, the rate of arrival of ACE bits is given by the number of IPC. The average time these

bits spend in the structure depends on the functionality of the block, which may store it for a long time (e.g. memory or buffers) leading to high AVFs or for shorter times (e.g. execution units) leading to lower AVF. Furthermore, for functional units like ALU, Eq. 3.10 shows that the more frequently they are used and the longer is the latency of the operation, the more vulnerable they are. For memories, it shows that the longer the average lifetime and the higher the memory utilization, the higher the AVF is.

### 3.4.2. IMPACT OF THE MICROARCHITECTURE ON THE FAILURE RATE



In Chapter 2 an overview on RISC-V ISA was presented and as well as the proposed approach of how to employ it in space data systems to address present and future needs. In this roadmap, several 'profiles' of processors were proposed. In this section, four General Purpose (GP) profiles will be analyzed from the point of view of dependability as case studies for the proposed model: GP-LE-1, GP-LE-4, GP-HE-1 and GP-HE-4. However, "GP" is usually omitted in the rest of this chapter, as only GP processors are considered. The LE-4 can be seen as an implementation equivalent to the state of the art of space-grade components (single-issue, in-order pipeline, quad-core like the GR740 [30]), while the HE-4 can be seen as a possible future space-grade processor. These configurations will be represented by the Rocket (LE) and the BOOM processor (HE) where FI was carried out in [242]. Therefore, for units in Tables 3.2 and 3.3 values of AVFs from [242] are employed. However, to provide a more comprehensive comparison of the contribution of each block in a realistic design, estimations for one IC, one DC and one FPU per core as well as for the L2C (one shared among the cores in LE-4 and HE-4) are also included. For the Floating Register File (FRF), as a pessimistic estimation, the same value of the Integer Register File (IRF) of the Rocket is employed, as data from [100] shows for FRF similar contribution to the failure rate compared to the IRF. When considering the functional part of the FPU, [252] shows that in average (over different benchmarks) only 1.76% of errors in FPUs reach the FPU output[6].

Table 3.2: AVF (from [100, 242, 252]) and $N_{eq}$ for LE-1 (without caches), decomposed in Integer Register File (IRF), Multiplier and Divider (M/D), Instruction Buffer (IB), rest of the Integer Unit (IU), Control and Status Registers (CSR), Floating Register File (FRF) and Floating Point Unit (FPU).

| LE-1 | IRF | M/D | IB | IU | CSR | FRF | FPU |
|---|---|---|---|---|---|---|---|
| $AVF_w$ | 3.3% | 0.2% | 0.5% | 2.4% | 5.9% | 3.3% | 1.0% |
| $AVF_h$ | 1.0% | 0.1% | 0.3% | 4.4% | 8.2% | 1.0% | 0.2% |
| $AVF_{eh,ut}$ | 12.2% | 0.4% | 1.1% | 4.9% | 4.3% | 12.2% | 0.6% |
| $N_{eq,LC}$ | 2.65E+3 | 2.17E+2 | 9.9E+1 | 1.1E+3 | 1.2E+3 | 2.8E+3 | 1.6E+3 |
| $N_{eq,AC}$ | 2.65E+3 | 5.72E+2 | 1.4E+2 | 1.7E+3 | 1.4E+3 | 2.8E+3 | 5.0E+3 |
| $N_{eq,HC}$ | 2.65E+3 | 9.27E+2 | 1.8E+2 | 2.2E+3 | 1.6E+3 | 2.8E+3 | 8.5E+3 |

[6] Further data from [252] shows that AVF for control modules in the FPU is 8.9% while datapath modules have a 1.43%. The large percentage of area dedicated to the datapath in a FPU explains the low average value. Also, this is a pessimistic estimation for the AVF of a FPU in a processor as the service interface is taken at the output of the FPU and not at the output of the processor, thus neglecting the masking effect of the rest of the processor to errors coming from the FPU. These data do not differentiate between types of failure so it is assumed that the breakdown is similar to the one of the ALU in the HE-1 in terms of $AVF_w$, $AVF_h$ and $AVF_{eh,ut}$.

Table 3.3: AVF (from [100, 242, 252]) and $N_{eq}$ for HE-1 (without caches), decomposed in Integer Register File (IRF), Register Rename (RR), Instruction Fetch (IF), Instruction Issue (II), Load and Store Unit (LSU), ReOrder Buffer (ROB), Branch Prediction (BP), Arithmetic-Logic Unit (ALU), Control and Status Registers (CSR), Floating Register File (FRF) and Floating Point Unit (FPU).

| HE-1 | IRF | RR | IF | II | LSU | ROB | BP | ALU | CSR | FRF | FPU |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $AVF_w$ | 1.9% | 2.4% | 2.6% | 2.4% | 1.5% | 1.2% | 0.8% | 1.2% | 3.9% | 3.3% | 1.0% |
| $AVF_h$ | 1.0% | 3.3% | 1.0% | 3.1% | 2.4% | 2.4% | 1.5% | 0.4% | 0.2% | 1.0% | 0.2% |
| $AVF_{eh,ut}$ | 8.7% | 5.7% | 7.3% | 0.9% | 3.7% | 0.8% | 0.1% | 0.7% | 5.4% | 8.7% | 0.6% |
| $N_{eq,LC}$ | 4.5E+3 | 2.9E+3 | 4.1E+3 | 7.1E+2 | 2.1E+3 | 1.1E+3 | 2.8E+3 | 1.9E+3 | 1.3E+3 | 3.4E+3 | 4.8E+3 |
| $N_{eq,AC}$ | 6.4E+3 | 4.1E+3 | 5.6E+3 | 9.8E+2 | 2.6E+3 | 1.2E+3 | 3.0E+3 | 3.7E+3 | 1.5E+3 | 4.3E+3 | 7.5E+3 |
| $N_{eq,HC}$ | 8.4E+3 | 5.2E+3 | 7.1E+3 | 1.2E+3 | 3.1E+3 | 1.4E+3 | 3.1E+3 | 5.5E+3 | 1.8E+3 | 5.1E+3 | 1.0E+4 |

Table 3.4: Features of the cache subsystem common to LE and HE (data derived from [253]). 'Pref.' stands for 'prefetcher', WB stands for Write-Back.

| Unit | Size | Block size | Associativity | Policy | Prefetching |
|---|---|---|---|---|---|
| DC | 32 KiB | 64 B | 4-way | WB | stride pref. |
| IC | 32 KiB | 64 B | 4-way | read-only | pref. |
| L2C | 1 MiB | 64 B | 16-way | WB | w/o pref. |

Table 3.5: AVF (from [253]) and $N_{eq}$ (the same for all technologies) for caches. LE-1 and HE-1 have one DC and one IC each. LE-4 and HE-4 are obtained replicating 4 times the respective single-core version and adding a L2C.

| Caches | $DC_{WT}$ | $DC_{WB}$ | IC | $L2C_{WB}$ |
|---|---|---|---|---|
| $AVF_w$ | 5% | 8.8% | 0.5% | 0.5% |
| $AVF_h$ | 1.3% | 2.5% | 5% | 0.6% |
| $AVF_{eh,ut}$ | 2.9% | 4.3% | 5.2% | 1.7% |
| $N_{eq}$ | 5.14E+4 | 5.7E+2 | 2.0E+5 | 2.4E+6 |

For all the profiles the same cache configuration is employed (i.e. the baseline of [253]) that is reported in Table 3.4 and with AVF values reported in Table 3.5. This will provide the reader with an estimation of how the same size of caches influences the failure rate in different designs (even if higher performance processors may employ larger caches). However, in Sec. 3.4.2 models and considerations on scaling of cache size will be provided. For simplicity, in this section only data arrays will be considered, neglecting the contribution of tag arrays in caches. Even if tag arrays in [254] are reported to have higher AVF than data arrays[7] (as for instance they have on average an AVF 2.76× higher than data arrays in DC), they typically are smaller (around 7 KiB, i.e. around 9 times smaller than the data array). Therefore, not including tag bits in the model can be expected to underestimate the vulnerability of caches by around 20% according to Eq. 3.4. Furthermore, using values for caches of a processor with a different ISA does not impact AVF of caches in a significant way, as shown in [255] where the AVF of caches for two different ISAs (ARM and x86) for 10 programs from the MiBench benchmark suite

[7] Also [235] shows a high value for tag arrays (32.5%).

have small differences[8].

Furthermore, the same average values of AVF are assumed for single and quad-core versions of the same design. As a matter of fact, [256] investigates the changes in AVF in a dual-core processor where each core is running a different thread and it shows that AVF is roughly the same compared to a single core (the change in AVF is within ±2% of the value of the AVF for a single core).

Estimations of $N_{eq}$ are obtained with syntheses on Design Compiler [257] on a 65 nm bulk commercial technology targeting 100 MHz and using the code available to the public of the Rocket processor[9] and of the BOOM processor[10]. However, as it was not possible to have access to the memory compiler of the ASIC technology (as it is often the case), the $N_{eq}$ of caches is estimated using CACTI [258].

It can be noted from Fig. 3.5 and 3.6 that caches are the most vulnerable units in processors, even considering technologies with high SER from combinational logic. Most of the units have a similar relative contribution to $\lambda_w$ and $U$, except the IC which has a similar impact compared to L2C in terms of unavailability but lags behind more than one order of magnitude in terms of $\lambda_w$. Most of the units increase their failure rate when moving from LC to SD. However, for a few of those (those with higher percentage of sequential elements like BP), the failure rate decreases due to FF temporal masking (as shown in [47]). Furthermore, microarchitectures impact the failure rate much more in terms of $N_{eq}$ than in terms of AVF. As a matter of fact, the maximum ratio between two different designs in terms of $N_{eq}$ with the same type of technology defined in this section (cacheless LE-1 and the HE-4) is around 100 for each technology, while the maximum ratio of AVFs found in literature due to different microarchitectures is around 4× (in [259]).

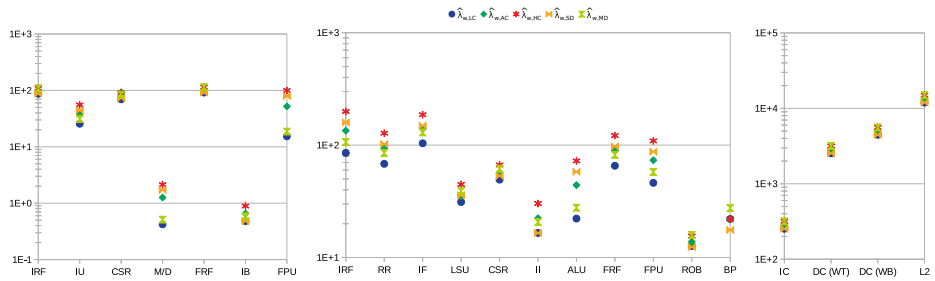


Figure 3.5: Normalized failure rate for wrong outputs $\hat{\lambda}_w$ for LE-1 (cacheless), HE-1 (cacheless) and caches (from left to right). Calculations based on Eq. 3.4.

### DESIGN EXPLORATIONS

In [260] the effect of the processor width and of the number of functional units (e.g ALU and FPU) on the AVF of the functional units is investigated but no clear correlation is found. Looking at data from the literature for IRF and caches (e.g. [253]), two models

[8] Intuitively, this is more true for L2C (-4%) and DC (+5%), while the difference is slightly larger for ICs (+24%), as instructions are ISA-specific [255]

[9] https://github.com/chipsalliance/rocket-chip.git

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

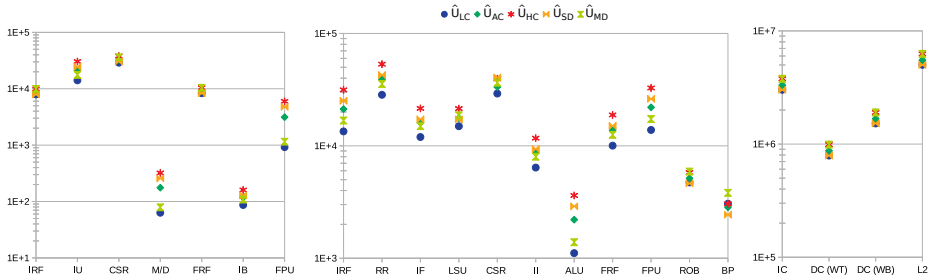


Figure 3.6: Normalized unavailability $\hat{U}$ for LE-1 (cacheless), HE-1 (cacheless) and caches (from left to right). Calculations based on Eq. 3.8.

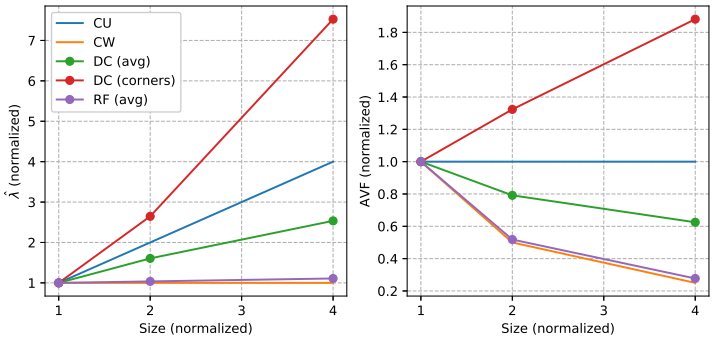


Figure 3.7: Effects of size on $\hat{\lambda}$ (normalized to the $\hat{\lambda}$ of the smallest size considered) and AVF for 2× and 4× increases (based on [253]). For DC the average of the benchmarks (*avg*) is indicated together with a superlinear case (for the benchmark *corners*).

of scaling of the failure rate can be defined for an array of sequential elements based on Eq. 3.10, as shown in Fig. 3.7:

1. Constant Workload (CW): the workload for the array remains constant while increasing the size of the unit, meaning that the failure rate remains constant and the AVF decreases by the same factor as the size was increased.

2. Constant Utilization (CU): the relative utilization of the array remains constant while increasing the size of the array, meaning that the AVF remains the same and the failure rate increases of the same amount the size was increased.

As shown in Fig. 3.7 some units show a behavior similar to CW (RF), some lay in between CU and CW (DC on average in [253]) and other units increase their utilization when their size is increased (DC for the *corners* benchmark in [253]) and in this case they are said to show a "superlinear" behavior (as done in [261]).

The results in [262] confirm the increase of failure rate of the DC when increasing its size. However, in this case the behavior shown is superlinear, as increasing its size by 16× (from 16 KiB to 256 KiB) increases its failure rate by 21×. Interestingly, they also

show that increasing the size of DC by 16x has an effect on the failure rate of L2, which decreases by around 2x. The work in [263] highlights how cache arrays typically exhibit a superlinear behavior when the cache hit rate increases with the increase of the size (e.g. for the *FFT* and *matrix multiplication* benchmarks), while if the cache hit rate remain constant they typically show a CW behavior. An explanation for this is presented in [261] and reported in Fig. 3.8 (left). If we consider a program that reads the variable *A*, then the variable *B* and then again the variable *A*, in a large cache, it is more likely that both *A* and *B* will reside in the cache. For this reason reading *B* does not cause a cache miss and line *A* is not evicted. In a small cache instead, reading *B* is more likely to cause a cache miss and a replacement of *A* with *B*, thus reducing drastically the fraction of time the location stores ACE-bits. The mechanism described happens for both WT and WB policies, while in Fig. 3.8 (right) it is also shown a mechanism specific of WB caches. As a matter of fact, in WB caches dirty lines also exist and those are always ACE, as they will be eventually written back to main memory. Fig. 3.8 (right) shows a program which writes *A* and then reads *B* and then does not act on the location until the end of the program, when the dirty lines will be written back. Also in this case, a small cache which substitute *A* with *B* can reduce the fraction of time the location stores ACE-bits considerably.

The previous discussion shows also that the write policy influences the AVF of the L2C: in [253] a value of 7% can be taken for a WB L2C (in [255] a similar value is given) and 4.2% for a WT L2C (1 MiB), which implies almost double the SER due to the L2C.

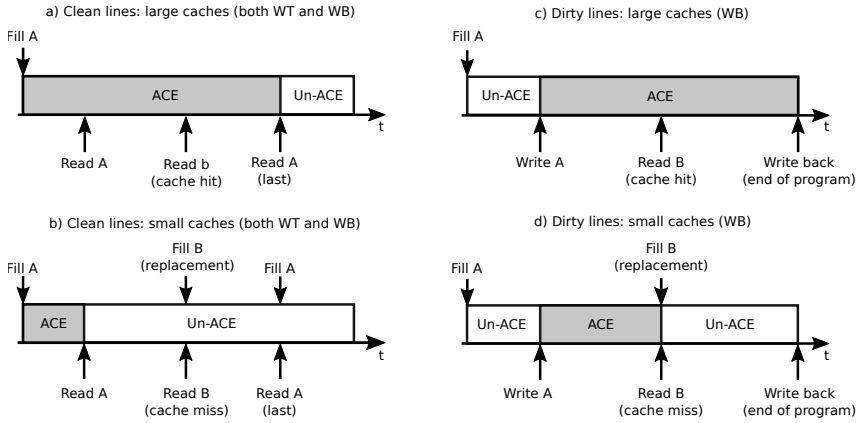


Figure 3.8: Examples of superlinear behavior for a location in DC storing the variable *A*, similarly to [261].

Furthermore, as shown in [253], the AVF of the DC is in the majority of the cases insensitive to the associativity (5 benchmarks out of 8), while some benchmarks (*djpeg* and *smooth*) exhibit a steep variation of AVF for a specific number of ways, and only one (*search*) shows an increase of AVF with the number of ways. IC instead decreases its AVF when the number of ways is increased [253]. Adding prefetches to the DC leaves substantially unchanged the AVF, while removing prefetchers for IC reduces the AVF, which becomes, on average, 0.67x the baseline AVF [253].

### 3.4.3. IMPACT OF OTHER FACTORS ON THE FAILURE RATE

Several factors impact the failure rate. Fig. 3.9 summarizes these factors indicating how large is the maximum value compared to the minimum value found of the failure rate when varying a certain factor. The impact of technology (and of the environment) and of microarchitecture was already assessed in Sec. 3.3.1 and Sec. 3.4.2 respectively. The remainder of this subsection quantifies the impacts of other factors.

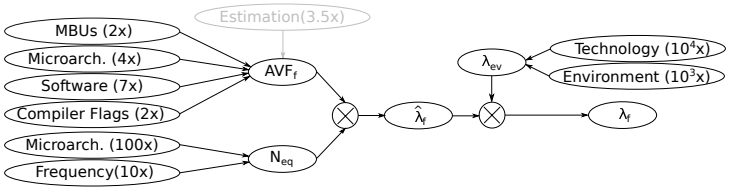


Figure 3.9: Factors impacting the failure rate and their relative impact (maximum value divided by the minimum value of the failure rate when varying a certain factor). The values were estimated as explained in Sec. 3.4.3. *Estimation* is in gray because it does not affect the failure rate on the field.

#### DEPENDENCE ON PERFORMANCE AND COMPILER FLAGS

The work in [264] observes a fuzzy correlation between AVF and each performance metric considered (i.e. IPC, branch prediction rate and several cache miss rates) across several SPEC2000 benchmarks. However, in [265] the use of performance throttling is proposed to lower the overall AVF of a processor. Acting both on pipeline resources and cache miss rate, a failure rate reduction up to 35% is achieved.

Another way to improve performance is to employ specific compiler flags. In [266], GCC with several combinations of optimization flags for the MiBench benchmark suite are compared in terms of performance and AVF. It is shown that the optimal set of flags for AVF decreases the AVF by around 9% compared to -O3 and of 8% compared to -O2. Among the optimization levels, in [267] -Os is found to be better than the -O0 for around 75% of the benchmarks (on a total of 25 benchmarks considered), while the lowest AVF in average is obtained with -O2. Recent work [268] shows that the ratio of the AVF obtained with the worst and best sets of optimization flags is around 2×.

#### DEPENDENCE ON SOFTWARE

Error masking in a processor, like performance, depends on the software employed. For this reason, it is crucial that the set of programs employed during the estimation of the AVF is representative of the final application or is general enough to represent a wide spectrum of applications. Most works in literature use the SPEC benchmark suites [242, 259], others use EEMBC suites [100] and others MiBench ([253, 266]) for its similarities to the SPEC benchmarks in terms, for instance, of instruction mix. As a matter of fact, instruction mix of the software can have a significant impact on the failure rate of certain units of the processor. For instance, data from [269] show that moving from benchmarks with low fraction of FPU instructions like *AMG2006* and *UMT* (0.03 and 0.1 per CC) to those with high fraction of FPU instructions like *LINPACK* (0.64 per CC) increases the failure rate of the FRF by 50x.

The variation of AVF on a microarchitecture employing different programs depends also on the microarchitecture itself. For instance, the work in [259] shows how, while for an In-Order (IO) "small core", the AVF ranges from 8% to 16% (2x maximum increase) for the benchmarks of the SPEC CPU2006, for an OoO "big core" it ranges from around 8% to around 29% (3.63x). Furthermore, the OoO core has for every benchmark a higher AVF compared to the in-order processor (except for *gobmk*). In [259] it is also shown the Cycles Per Instruction (CPI) stack[11] for each benchmark, highlighting that there is no simple rule to predict whether a workload has high or low AVF[12]. According to [259], the benchmarks with low AVF have low vulnerability because of their high number of branch mispredictions and instruction cache misses. The benchmarks with high AVF show instead a more complex behavior. Some benchmarks (e.g. *milc*) are memory-intensive: a load operation accessing main memory typically blocks the head of the ReOrder Buffer (ROB), which causes the ROB to fill up. This leads to a significant increase of ACE bits while servicing the memory operation[13]. However, some memory-intensive benchmarks (e.g., *mcf* and *libquantum*) have low AVF because of branch mispredictions. Other high-AVF benchmarks (e.g. *zeusmp*) are compute-intensive: high IPC and high Memory Level Parallelism (MLP)[14] is achieved by having high occupancy in various queues. Some benchmarks with high AVF instead experience resource stalls because of DC misses, L2C misses, limited Instruction Level Parallelism (i.e., chains of dependent instructions) which cause the ROB and issue queues to fill up with instructions. Data in [242] show a different trend. In this case, the AVF values of the OoO core are smaller than those of the in-order core for every benchmark, and this is also true for the only two benchmarks that are considered in both works (*bzip* and *gcc*) [259]. Also, the increase in ratio between the minimum and the maximum AVF found is much lower: from 12.4% to 20.6% for the in-order (1.66×) against 7% to 12.3% (1.76×) for the OoO processor. When considering caches, in [253] the AVF for the baseline DC ranges from around 3% to around 23% (7.66×).

### DEPENDENCE ON THE FRACTION OF MBUS

In [142] more complex error models compared to the Single Bit Flip (SBF) of [242] are employed to investigate the effects of MBUs on the AVF. The most relevant result is that the AVF value saturates on average around 3 upsets per strike, with an increase of around 10% compared to the value found with the SBF model on average and with a peak of around 25%. To take into account this effect, the AVF employed in Eq. 3.4 will be then $\overline{AVF} = \alpha_{MU} \cdot AVF$. In the remainder of this chapter $\alpha_{MU} = 1$ will be assumed for technologies with a low fraction of MBUs (LC and SD), $\alpha_{MU} = 1.1$ for technologies with an

[11] A CPI stack quantifies the fraction of cycles spent doing useful work, 'lost' cycles because of resource stalls, branch mispredictions, instruction cache misses, Last-Level Cache (LLC) misses and main memory accesses [259].

[12] It should be noted that the ACE states of caches are not evaluated in this case, as caches are assumed to be protected.

[13] This mechanism is only relevant to OoO processors and does not happen in in-order processors. This explain why the ranges are different.

[14] MLP is the average number of useful long-latency off-chip accesses outstanding when there is at least one such access outstanding [270]. Also in this case, this is a mechanism typical of OoO processors, as simulation results in [270] show that a moderately aggressive OoO issue processor improves MLP over an in-order issue processor by 12-30%.

average fraction of MBUs (AC) and $\alpha_{MU} = 1.25$ for technologies with a high fraction of MBUs (HC and MD). The impact of MBUs on the failure rate is limited, as the ratio between the minimum and the maximum value of AVF changing the number of upsets per strike reported in [142] is around 2x. Also in [269], the maximum ratio found when injecting one and four upsets is 2x.

#### UNCERTAINTY DUE TO THE ESTIMATION METHOD EMPLOYED

AVF was originally defined with an ACE analysis implemented on a microarchitectural simulator [78]. In [271], the ACE approach is found to underestimate on average fault masking by about 3.5x compared to FI. The causes identified for such overestimations are: the limited information on the bits (when it cannot be determined whether a bit is in a ACE or un-ACE state, it is assumed in ACE-state to prove that requirements can be met); limited size of time windows to analyze dead instructions; and Y-bits[15]. The conclusion in [271] is that ACE analysis can be refined until a theoretical threshold is reached, after which is not possible to further reduce conservatism because of Y-bits. However, before this theoretical limit for ACE analysis is reached, ACE analysis becomes intractable due to the increase of complexity. The authors of [273] reject this point of view, arguing that while FI on RTL may provide a more accurate AVF by modeling all low-level masking effects, much of this can be accounted for at the performance level by identifying and modeling those masking effects that significantly impact the AVF and that the Y-bit effect is on the order of 14% on the AVF and that it can be addressed with a more refined ACE analysis [271]. While most of the extended microarchitectural simulators are not available to the public, a modified version of the gem5 simulator capable of assessing soft error vulnerability [274] is available[16].

FI requires a large number of experiments and either a working hardware platform or an RTL model that can be simulated. However, the results in [275], regarding a dual-core processor design consisting of around 350k sequential elements, show that randomly selecting more than 2.85% elements (10k) for injections provides only marginal improvements in terms of reduction of uncertainty (the standard deviation when considering 10 different groups of FFs saturates).

To inject errors also in microarchitectural resources of a hardware prototype, hardware support is needed. For instance, in [242] faults are injected in a FPGA prototype with an extra XOR at the input of each FF of the processor. The host processor within the FPGA decides which FF to inject (and at which CC) and sends the command to the fault injector connected through a crossbar. Without a similar hardware support, errors could be injected only via software in architectural resources. Another possibility is to simulate an RTL model and inject errors during the simulation, changing the value of a specific signal [276].

### 3.4.4. LIMITATION OF THE MODEL

Although the use of the AVF decomposition as introduced in Sec. 3.4.1 is common [55], beside reasonable assumptions as the independence between errors and the uniform

[15] Y-Bits are bits that can alter the course of execution in the processor without causing a failure, for instance branches for which the behavior of the application is unaffected by whether the particular branch instance is taken or not. Around 40% of the executed branches in SPECint2000 are Y-branch [272].

[16] https://github.com/MPSLab-ASU/gemV

masking, there are some limitations in its capability of assessing the vulnerability of processor units.

#### SUB-UNIT VULNERABILITY

The AVF decomposition does not provide insights on the homogeneity in terms of vulnerability of a certain structure. The work in [100] provides instead also data for sub-unit vulnerability. In this case, the sequential elements of each unit are grouped in Criticality Levels (CLs), depending on the percentage of times an error in that sequential element propagates to the outputs. For instance, CL0 means that a fault in that element never causes a failure and CL5 indicates that a fault in that element always causes a failure. This classification may allow selecting redundancy more efficiently. For instance, a memory array with a large fraction of CL0 sequential elements can be protected more efficiently with selective information redundancy [277, 278] or partial hardware redundancy [279]. While in [100] a conservative approach is adopted which defines an FF as critical if it is critical at least for a benchmark, data from [47] suggest that a considerable part of the critical FFs remains the same in 8 programs from MiBench (85 out of the top 200 vulnerable FFs of each benchmark), and only a minority (around 13% for each benchmark) are critical in only a single benchmark. For instance, [277] notes that only a few "long-lived" registers (10% for the IRF) are responsible for 40% of the total vulnerability time of the IRF. Based on this consideration, it is proposed to use a cache smaller than the RF to store the ECC of physical registers and replace check bits of "short-lived" registers with those of the "long-lived" ones.

#### PROPAGATION TO SPECIFIC SIGNALS AT THE SERVICE INTERFACE

The AVF decomposition does not take into account to which signal of the service interface the errors will propagate. In [100] it is found that errors manifest only in 65% of the outputs, with almost 80% of these errors manifesting in only 20% of the ports.

#### PROPAGATION TIME

The AVF decomposition also does not take into account how long it takes for the propagation of the error to the service interface. Data from [100] show that all of the processor components (without considering caches) have a minimum error manifestation time equal or less than 7 CCs, whereas the average error manifestation time for an error in the processor is 1,204 CCs. The worst propagation time is 153,287 CCs (for the logic responsible for branch prediction). Errors propagate more quickly when they directly affect the processing data (e.g. ALU and FPU), while storage units like RFs have instead longer propagation times [100]. The FRF has longer average propagation times (2950 CCs) compared to the IRF (370 CCs), mainly because of more matrix operations and longer latency of the FPU compared to the ALU. The exception are some long-life integer variables, such as indexes in iterative loops (propagation time on the order of the thousands of CCs) [100].

#### ERROR ACCUMULATION

The AVF decomposition assumes that the software is composed of program loops of period $T_L$ each [55]. In order for the AVF decomposition to produce a negligible error, the product $T_L \cdot \lambda$ must be small [55]. This means that AVF decomposition is valid when a

small number of soft errors occur in a loop iteration. In [280] a model to overcome this limitation is proposed, however it is much more complex. Nevertheless, for each unit a failure rate $\lambda_i = DF \cdot SER_i$ can be associated, where $DF$ is a more general derating factor. Therefore, the final result of such more detailed model is a failure rate for each unit in the design like those in Figs. 3.5 and 3.6, on which the same procedures to apply and validate redundancy can be followed as done in Sec. 3.5 and 3.6.1.

## 3.5. EVALUATION OF REDUNDANCY



Given the possible large overheads caused by redundancy (see Sec. 1.3.3 and Sec. 1.3.4), the concept of cost-effectiveness is introduced in [277] (where a proposed technique is compared to others in terms of area, power and performance overhead) and in [279] (where area and power overhead are considered). To provide a metric for this concept, the following cost function $C$ is defined:

$$C = \alpha \frac{\Delta T_{ex}}{T_{ex}} + \beta \frac{\Delta A}{A} + \gamma \frac{\Delta P}{P} + \delta \frac{\Delta \lambda_w}{\lambda_w} + \epsilon \frac{\Delta U}{U} \tag{3.11}$$

where $\alpha$, $\beta$, $\gamma$, $\delta$, and $\epsilon$ are arbitrary weights depending on the target of the design and express the weight of:

- The term $\Delta T_{ex}/T_{ex}$ is the relative increase in the execution time for a (set of) program(s) employed to evaluate performance or the execution of a certain task. The performance model in Eq. 1.1 will be employed[17]. However, an increase in $T_{clk}$ may be partially compensated by a decrease in CPI due to the fact that the frequency of the memory remains the same and for this reason the penalty is less than proportional to the loss in $T_{clk}$ of the processor. Furthermore, latencies in case of DUEs or corrections are not considered in the loss of performance, as they are not frequent enough to cause degradation in performance (as opposed to increase of latency even when there are no errors).
- The terms $\Delta A/A$ and $\Delta P/P$ indicate respectively the relative increase in terms of area and power of the whole processor (keeping the same target and operative frequency).
- The quantities $\Delta \lambda_w/\lambda_w$ and $\Delta U/U$ account for the percentage of errors detected by the redundancy, i.e. its coverage.

To show how weights can affect the optimal choice, two opposite cases will be taken into account:

1. Focus on dependability ($C_d$): $\alpha = 0.25$, $\beta = 0.5$, $\gamma = 0.5$, $\delta = 1$, $\epsilon = 1$. As these weights make increases in terms of unavailability and failure rate very costly and decreases in terms of performance less important, this can be seen as the set of weights used in the cost function employed during in the design of an OBC for command and control operations.

[17] It should be noted that increases in CPI are actually more expensive than increases in $T_{clk}$ as the increase in CPI implies that the FT processor is not functionally equivalent to its COTS counterpart even when errors are not detected.

2. Focus on performance ($C_p$): $\alpha = 1$, $\beta = 0.5, \gamma = 0.5$, $\delta = 0.25$, $\epsilon = 0.5$. As these weights make decreases in terms of performance expensive and increases in terms of failure rate tolerable (while increases in terms of unavailability are considered to be in the middle), this can be seen as the set of weight used for the cost function employed during the design of a payload processor for high-performance onboard processing.

In order to quantify the effect of redundancy on $\Delta\lambda_w/\lambda_w$ and $\Delta U/U$, let us consider for brevity only errors in the sequential elements (analogous equations can be derived changing the *seq* indices into *comb*, and *SEU* into *SET*), the $\Delta\hat{\lambda}_{w,seq}$ of a certain redundancy technique can be determined with the equations below:



$$\hat{\lambda}_{w,seq,nored} = \sum_i AVF_{w,i} \cdot N_{eq,SEU,i} \tag{3.12}$$

$$\hat{\lambda}_{w,seq,red} = \sum_i AVF_{w,i} \cdot \left(N_{eq,SEU,i} + \Delta N_{eq,SEU,i}\right) \cdot P_{undet,seq,i} \tag{3.13}$$

$$\Delta\hat{\lambda}_{w,seq} = \hat{\lambda}_{w,seq,red} - \hat{\lambda}_{w,seq,nored} = \sum_i AVF_{w,i}\, N_{eq,SEU,i} \left[P_{undet,seq,i}\left(1 + \frac{\Delta N_{eq,SEU,i}}{N_{eq,SEU,i}}\right) - 1\right], \tag{3.14}$$

where $P_{undet,seq,i}$ is the probability that the error is not detected in the unit $i$ for sequential logic. Effective redundancy will have a negative $\Delta\lambda_w/\lambda_w$ and will decrease the cost function, but it is mathematically possible to have a positive $\Delta\lambda_w/\lambda_w$ for inefficient redundancy, when:

$$P_{undet,seq}\left(1 + \frac{\Delta N_{eq,SEU}}{N_{eq,SEU}}\right) > 1. \tag{3.15}$$

The case where unavailability increases with redundancy is instead more common, because of the unavailability from error handling:

$$\Delta\hat{U} = T_{u,h}\Delta\hat{\lambda}_h + \sum_j T_{u,eh,j}\Delta\hat{\lambda}_{eh,j}, \tag{3.16}$$

where $j$ is the index of the j-th type of unavailability due to error handling, $\Delta\hat{\lambda}_h$ can be found with the same equation as Eq. 3.14 and $\Delta\hat{\lambda}_{eh,j}$ is given by:

$$\Delta\hat{\lambda}_{eh,j} = \sum_i AVF'_i \left[N_{eq,SEU,i}\left(1 + \tfrac{\Delta N_{eq,SEU,i}}{N_{eq,SEU,i}}\right) P_{det,seq,i} + N_{eq,SET,i}\left(1 + \tfrac{\Delta N_{eq,SET,i}}{N_{eq,SET,i}}\right) P_{det,comb,i}\right], \tag{3.17}$$

where $AVF'_i$ is the total AVF for the unit $i$, considering as service interface the point where the redundancy can detect the error in the processor. This change of definition of AVF

in this equation (with $AVF'_i \geq AVF_i$) is needed because redundancy will react also to errors that manage to propagate to this point and that would be masked in the rest of the propagation to the real service interface if redundancy was not included. This implies that the rate of error handling events $\Delta\hat{\lambda}_{eh} = \sum_j \hat{\lambda}_{eh,j}$ is larger than the decrease of the rate of other events $-\Delta\hat{\lambda} = -(\Delta\hat{\lambda}_w + \Delta\hat{\lambda}_h + \Delta\hat{\lambda}_{eh,ut})$. An example is given in Sec. 3.5.1, where $\Delta\hat{\lambda}_{eh}$ is larger than $-\Delta\hat{\lambda}$ by a factor ranging from 1.8x to 13.6×.

In the following sections, several types of redundancy for different units of processors will be introduced, evaluating their efficacy and cost-effectiveness for different designs and different technologies/environments combinations. In order to show different types of redundancy, the cost function will be applied to each part of the processor, decomposed in:

1. Cache Arrays (Sec. 3.5.1)
2. Register Files (Sec. 3.5.2)
3. Logic (Sec. 3.5.2), composed of the remaining combinational and sequential elements.

For each of them the most cost-efficient redundancy for different designs, weights and technologies will be assessed. In Sec. 3.6 the total effect of applying the most cost-efficient to all the components of the processors will be analyzed. More complex optimization methods can be employed, as done in [237].

### 3.5.1. CHOICE OF REDUNDANCY FOR CACHE ARRAYS

Several processors described in literature employ SED in L1 caches and SECDED in L2C (referred to as EDC/ECC), while others have SECDED in both levels (referred to as ECC/ECC) [251]. The main argument in favor of the EDC/ECC approach is that it causes a smaller increase in word length, as the increase in word length increases the access latency to the word. For this reason, the latency penalty per access compared to an unprotected L1 cache is less than 1% for the SED, while for the SECDED it is larger (but still below 10%, as the access latency is dominated by the data array's word-line decoder [251]). However, EDC/ECC cannot correct errors in L1 caches, therefore program execution cannot in general continue after the detection of an error in DC and a reset is required. This problem is typically mitigated reading the correct, up-to-date value from the next level of the memory hierarchy [84]. In order to make this correction possible, this solution requires WT policy for DC, which incurs in significant performance and power overheads [80]. On average EDC/ECC has a runtime penalty comparable with ECC/ECC (+12% vs +2% for SPECint in [251]). However for some specific benchmarks the penalty for using EDC/ECC is much higher. For instance, for *bzip2-graphic* the penalty of EDC/ECC over the unprotected version is +157% and +75% for *vortex3*. However, according to the data in [251], SED incurs in less area overhead (virtually none already for L1 caches of 8 KiB), while the SECDED at L1 causes an area overhead between around 10% and 50%, depending on the area of the cache (in case of 32 KiB it is around 20%). Using CACTI [258] it can be found that the increase in terms of area for applying ECC in a 1 MiB L2C is around 21%. Regarding power, in [251] ECC/ECC has an overhead on the order of 20% for 32 KiB, while using CACTI a 24.46% increase for a cache of 1 MiB is found.

With EDC, also considering the required write to the next level of the memory hierarchy due to the WT policy, it is around 350% [251]. It should be noted that both power and area given previously are at cache subsystem level, thus using them directly in the cost function would overestimate the cost in terms of power and area of cache redundancy (even if caches in most cases consume a large fraction of the power of a processor [281]). To estimate the actual relative increases at processor-level, LE and HE were modelled in McPAT [282]. This modelling shows that DC and IC consume respectively 18.95% and 4.53% of the total power consumption for the LE-1 and 17.12% and 4.11% for the HE-1. In the case of LE-4 DCs consume 14.97% of the total power, ICs 3.58% and L2C 18.95% for LE-4. The same fractions for the HE-4 are respectively 14.82%, 3.56% and 11.13%.



Regarding the changes in failure rate and unavailability, limiting the analysis to triple and quadruple errors in the same word, the probability of miscorrection for a SECDED $P_{SD,misc}$ is:

$$P_{SD,misc} = MBU_3 \cdot P_{misc,3} + MBU_4 \cdot P_{misc,4}. \tag{3.18}$$

Therefore, the change in failure rate due to the ECC/ECC is (from Eq. 3.14, assuming $\Delta N_{eq,SEU}$ negligible):

$$\Delta\hat{\lambda}_w = -[n_c \cdot N_{L1} \cdot (AVF_{DC_{WB}} + AVF_{IC}) + N_{L2C} \cdot AVF_{L2C}] \cdot (1 - P_{SD,misc}) \tag{3.19}$$

where $N_{L1}$ is the size of a single L1 cache and $n_c$ is the number of cores.
In the case of EDC/ECC the change in failure rate is instead:

$$\Delta\hat{\lambda}_w = -[n_c \cdot N_{L1} \cdot (AVF_{DC,WT} + AVF_{IC})(1 - MBU_{even})] - N_{L2C} \cdot AVF_{L2C} \cdot (1 - P_{SD,misc}). \tag{3.20}$$

When calculating the change in availability, employing *AVF* is too optimistic, as once a certain location is read, the error handling mechanism will act also on detected errors that would not reach the service interface (as already pointed out in Eq. 3.17). To eliminate the fraction of masking due to the propagation from the cache to the outputs of the processor, the Cache Vulnerability Factor (CVF), defined in [283] as the probability of an error in the cache to propagate outside the cache (i.e. being read), can be used instead. In [283] average CVF data for a similar cache configuration are given when running 11 benchmarks from SPEC2000. The CVFs found are 38.03% for $L2C_{WB}$, 57.70% for $DC_{WB}$, 16.47% $DC_{WT}$, and 32.05% for IC. The increase in $\lambda_{eh,DUE}$ for ECC/ECC can be then estimated as:

$$\Delta\hat{\lambda}_{eh,DUE} = [n_c \cdot N_{L1} \cdot (CVF_{DC_{WB}} + CVF_{IC}) + N_{L2C} \cdot CVF_{L2C}] \cdot MBU_2. \tag{3.21}$$

In the case of EDC/ECC the change in $\lambda_{eh,DUE}$ is:

$$\Delta\hat{\lambda}_{eh,DUE} = [n_c \cdot N_{L1} \cdot (CVF_{DC_{WT}} + CVF_{IC}) \cdot MBU_{odd}] + N_{L2C} \cdot CVF_{L2C} \cdot MBU_2. \tag{3.22}$$

In Table 3.6 the costs of applying EDC, ECC and EDC+CI to single core versions and EDC/ECC, ECC/ECC and EDC+CI/ECC to quad-core versions are compared in terms of area, power and performance. In Table 3.7 the change of failure rate shows that while

Table 3.6: Relative change [%] in execution time, area and power for redundancy in caches and different designs. Data from [251], [89] and modelling in CACTI and McPAT.

| Redund. | EDC | | ECC | | EDC+CI | | EDC/ECC | | ECC/ECC | | EDC+CI/ECC | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Design | LE-1 | HE-1 | LE-1 | HE-1 | LE-1 | HE-1 | LE-4 | HE-4 | LE-4 | HE-4 | LE-4 | HE-4 |
| $\Delta T_{ex}/T_{ex}$ | 12 | 12 | 2 | 2 | 12 | 12 | 12 | 12 | 2 | 2 | 12 | 12 |
| $\Delta A/A$ | ~0 | ~0 | 14 | 10 | 72 | 49 | 17 | 9 | 20 | 18 | 31 | 29 |
| $\Delta P/P$ | 30 | 10 | 5 | 1 | 54 | 20 | 37 | 12 | 7 | 2 | 45 | 16 |

Table 3.7: Relative changes in $\lambda_w$ and $U$ [%], referred to the respective unprotected version with $DC_{WB}$ (according to Eqs. 3.19, 3.20, 3.21, and 3.22). The SECDED for ECC is an *Odd-Weigth Column* code. In bold the most effective redundancy for each design/technology combination.

| Redundancy | Design | $\frac{\Delta\lambda_{w,LC}}{\lambda_{w,LC}}$ | $\frac{\Delta\lambda_{w,AC}}{\lambda_{w,AC}}$ | $\frac{\Delta\lambda_{w,HC}}{\lambda_{w,HC}}$ | $\frac{\Delta\lambda_{w,SD}}{\lambda_{w,SD}}$ | $\frac{\Delta\lambda_{w,MD}}{\lambda_{w,MD}}$ | $\frac{\Delta U_{LC}}{U_{LC}}$ | $\frac{\Delta U_{AC}}{U_{AC}}$ | $\frac{\Delta U_{HC}}{U_{HC}}$ | $\frac{\Delta U_{SD}}{U_{SD}}$ | $\frac{\Delta U_{MD}}{U_{MD}}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| EDC | LE-1 | -92 | -69 | -53 | -90 | -54 | -92 | -59 | -36 | -91 | -37 |
| | HE-1 | -88 | -64 | -49 | -84 | -51 | -86 | -54 | -33 | -82 | -35 |
| ECC | LE-1 | **-94** | -91 | -92 | -92 | -93 | -87 | -37 | -84 | -86 | -85 |
| | HE-1 | **-90** | -85 | -85 | -85 | -89 | -82 | -34 | -77 | -78 | -80 |
| EDC+CI | LE-1 | **-94** | **-93** | **-93** | **-93** | **-94** | **-95** | **-94** | **-94** | **-94** | **-95** |
| | HE-1 | **-90** | **-88** | **-86** | **-86** | **-90** | **-89** | **-87** | **-85** | **-85** | **-89** |
| EDC/ECC | LE-4 | -95 | -79 | -70 | -94 | -71 | -77 | 56 | -47 | -77 | -47 |
| | HE-4 | -92 | -76 | -66 | -89 | -68 | -75 | 54 | -50 | -73 | -46 |
| ECC/ECC | LE-4 | **-96** | -93 | **-95** | **-95** | **-96** | -75 | 66 | -70 | -74 | -71 |
| | HE-4 | **-93** | -90 | -90 | **-90** | **-93** | -73 | 64 | -67 | -71 | -69 |
| EDC+CI/ECC | LE-4 | **-96** | **-95** | **-95** | **-95** | **-96** | **-79** | **38** | **-75** | **-78** | **-75** |
| | HE-4 | **-93** | **-91** | **-91** | **-90** | **-93** | **-76** | **37** | **-71** | **-74** | **-73** |

EDC+CI and EDC+CI/ECC have the highest reduction for every technology, for technologies with low fraction of MBUs the improvement they can provide over ECC and ECC/ECC is negligible. The unavailability of quad-core designs with AC technology increases compared to a version without redundancy because of DUEs due to a large fraction of $MBU_2$. In Table 3.8 the total cost is shown for each technology/design/redundancy combination. EDC is the most cost-effective for both single-core designs and weight factors for technologies with low fraction of MBUs (i.e. LC, SD). When the fraction of MBUs becomes significant, its distribution determines the most cost-effective redundancy. For instance, AC requires EDC+CI and EDC+CI/EDC because most of its fraction of MBUs causes more than two upsets, while ECC and ECC/ECC is enough in most cases for HC as in this case the majority of MBUs causes only two upsets. It should also be noted that in the case of AC the cost of applying EDAC codes to quad-core designs is always greater than zero, implying that not applying EDAC codes would be more cost-effective. However, EDAC codes are typically applied anyway to achieve requirements in terms of $MTTF_w$.

### 3.5.2. CHOOSING THE REDUNDANCY FOR THE REST OF THE PROCESSOR

The rest of the processor can be divided in residual (smaller than caches) SRAM arrays (e.g. RFs) and logic (i.e. composed of FFs and combinational logic). As pointed out in

Table 3.8: Cost-effectiveness of several redundancies for caches for several technologies and weights. In bold the most cost-effective solutions for each combination of redundancy, design and technology.

| Redundancy | Design | $C_{d,LC}$ | $C_{d,AC}$ | $C_{d,HC}$ | $C_{d,SD}$ | $C_{d,MD}$ | $C_{p,LC}$ | $C_{p,AC}$ | $C_{p,HC}$ | $C_{p,SD}$ | $C_{p,MD}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| EDC | LE-1 | **-1.66** | -1.09 | -0.71 | **-1.63** | -0.72 | **-0.42** | **-0.19** | -0.04 | **-0.41** | -0.05 |
| | HE-1 | **-1.66** | -1.11 | -0.74 | **-1.57** | -0.78 | **-0.48** | **-0.26** | -0.12 | **-0.45** | -0.13 |
| ECC | LE-1 | **-1.66** | -1.13 | **-1.61** | **-1.63** | **-1.64** | -0.37 | -0.12 | **-0.36** | -0.36 | **-0.36** |
| | HE-1 | -1.61 | -1.09 | **-1.51** | -1.52 | **-1.59** | -0.38 | -0.13 | **-0.34** | -0.35 | **-0.37** |
| EDC+CI | LE-1 | -1.23 | **-1.22** | -1.20 | -1.20 | -1.23 | 0.04 | 0.04 | 0.05 | 0.05 | 0.04 |
| | HE-1 | -1.42 | **-1.38** | -1.33 | -1.33 | -1.42 | -0.21 | -0.19 | -0.18 | -0.17 | -0.21 |
| EDC/ECC | LE-4 | -1.42 | 0.06 | -0.87 | -1.40 | -0.88 | -0.23 | 0.47 | -0.02 | -0.23 | -0.02 |
| | HE-4 | **-1.53** | -0.09 | -1.03 | **-1.48** | -1.00 | -0.38 | **0.30** | -0.19 | **-0.36** | -0.17 |
| ECC/ECC | LE-4 | **-1.52** | -0.09 | **-1.46** | **-1.51** | **-1.48** | **-0.28** | **0.43** | **-0.25** | **-0.27** | **-0.26** |
| | HE-4 | -1.51 | -0.11 | **-1.42** | -1.46 | **-1.46** | **-0.30** | 0.40 | **-0.26** | -0.28 | **-0.27** |
| EDC+CI/ECC | LE-4 | -1.34 | **-0.16** | -1.29 | -1.32 | -1.30 | -0.13 | 0.45 | -0.11 | -0.13 | -0.12 |
| | HE-4 | -1.44 | **-0.29** | -1.37 | -1.39 | -1.41 | -0.27 | **0.30** | -0.24 | -0.25 | -0.25 |

Table 3.9: Relative increase [%] in execution time, area and power for redundancies protecting RFs, logic and both simultaneously. Data from [81, 94–97, 277] and modelling in McPAT.

| Redund. | RF-ECC | | RF-TMR | | FF-TMR | | FFD-TMR | | C-TMR | | C-DMR | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Design | LE-1 | HE-1 | LE-1 | HE-1 | LE-1 | HE-1 | LE-1 | HE-1 | LE-1 | HE-1 | LE-1 | HE-1 |
| $\Delta T_{ex}/T_{ex}$ | 4 | 4 | 1 | 1 | 8 | 8 | 45 | 45 | 0 | 0 | 0 | 0 |
| $\Delta A/A$ | 0 | 1 | 8 | 26 | 9 | 32 | 15 | 53 | 65 | 107 | 28 | 51 |
| $\Delta P/P$ | 5 | 9 | 9 | 17 | 66 | 65 | 71 | 71 | 60 | 61 | 43 | 44 |

1.4.2, two main approaches can be found in literature: protecting separately RFs and the logic [81] or protecting them simultaneously [98]. The rest of this section investigates the impact of these two approaches on different designs.

### CHOOSING THE REDUNDANCY FOR THE RFS

Similarly to caches, RFs are typically protected with information redundancy. However, as they are smaller than caches, replicating the RF may also be a viable solution. For this reason, the effects in the case of SECDED (RF-ECC) and TMR of the RFs (RF-TMR) are compared. In [277] RF-ECC is reported to increase the power of the RF by 100% and the area of 4.9%. Table 3.9 reports how these estimations increase area and power at processor level for the two designs and Table 3.10 shows which relative variations in terms of failure rate and unavailability they produce. As shown in Table 3.9, RF-TMR is in general more expensive in terms of area and power, although less expensive in terms of performance. In [234], a Double Modular Redundancy (DMR) with parity is proposed as a less expensive version of RF-TMR, which is capable of achieving the same relative change in failure rate and unavailability with lower overhead in terms of area and power. Nevertheless, RF-TMR can be more cost-effective than RF-ECC when the focus is on performance and for some designs (e.g. HE-1), as shown in Table 3.11.

Table 3.10: Relative changes [%] in wrong outputs and of unavailability for the LE-1 and HE-1 when adding redundancy to IRF and RFR. In bold the most effective solutions for each combination of redundancy, design and technology.

| Redund. | Design | $\frac{\Delta\lambda_{w,LC}}{\lambda_{w,LC}}$ | $\frac{\Delta\lambda_{w,AC}}{\lambda_{w,AC}}$ | $\frac{\Delta\lambda_{w,HC}}{\lambda_{w,HC}}$ | $\frac{\Delta\lambda_{w,SD}}{\lambda_{w,SD}}$ | $\frac{\Delta\lambda_{w,MD}}{\lambda_{w,MD}}$ | $\frac{\Delta U_{LC}}{U_{LC}}$ | $\frac{\Delta U_{AC}}{U_{AC}}$ | $\frac{\Delta U_{HC}}{U_{HC}}$ | $\frac{\Delta U_{SD}}{U_{SD}}$ | $\frac{\Delta U_{MD}}{U_{MD}}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| RF-ECC | LE-1 | **-3.6** | **-3.5** | -1.7 | **-3.5** | -3.5 | -1.2 | -0.3 | -1.2 | -1.2 | -1.2 |
| | HE-1 | **-2.9** | -3.7 | -4.6 | -4.6 | -2.9 | -1.8 | -1.9 | -2.9 | -2.9 | -1.8 |
| RF-TMR | LE-1 | **-3.6** | **-3.5** | **-3.5** | **-3.5** | **-3.6** | **-1.4** | **-1.4** | **-1.3** | **-1.3** | **-1.4** |
| | HE-1 | **-2.9** | **-3.8** | **-4.7** | **-4.7** | **-2.9** | **-1.9** | **-2.5** | **-3.0** | **-3.0** | **-1.9** |

Table 3.11: Cost of redundancy in IRF and FRF in the case of SECDED (RF-ECC) and triplication of the RFs (RF-TMR). In bold the most cost- effective solutions for each combination of redundancy, design and technology.

| Redund. | Design | $C_{d,LC}$ | $C_{d,AC}$ | $C_{d,HC}$ | $C_{d,SD}$ | $C_{d,MD}$ | $C_{p,LC}$ | $C_{p,AC}$ | $C_{p,HC}$ | $C_{p,SD}$ | $C_{p,MD}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| RF-ECC | LE-1 | **-0.01** | **0.00** | **0.01** | **-0.01** | **-0.01** | **0.05** | **0.05** | **0.05** | **0.05** | **0.07** |
| | HE-1 | **0.01** | **0.00** | **-0.02** | **-0.02** | **0.03** | 0.15 | 0.15 | 0.14 | 0.14 | 0.15 |
| RF-TMR | LE-1 | 0.04 | 0.04 | 0.04 | 0.04 | 0.04 | 0.08 | 0.08 | 0.08 | 0.08 | 0.08 |
| | HE-1 | 0.04 | 0.03 | 0.02 | 0.02 | 0.04 | **0.08** | **0.07** | **0.07** | **0.07** | **0.08** |

### CHOOSING THE REDUNDANCY FOR LOGIC

As the cross section for a triplicated FF is between three to one orders of magnitude less compared to the cross section of an unprotected FF in [284], for FF-TMR and FFD-TMR it is assumed that 10% of the events will corrupt more than one of the FFs in MD and HC, 1% of the events will corrupt more than one FF in AC and 0.1% in LC and SD. Furthermore, FFD-TMR is considered to mask all SETs, as [94] report immunity to spikes up to 105 ps. Table 3.12 shows the relative changes in terms of $\lambda_w$ and $U$ when applying FF-TMR and FFD-TMR to the logic of LE-1 and HE-1. Furthermore, Table 3.13 shows that, despite the optimistic assumptions on the capability of FFD-TMR to mask all the SETs, its cost is so high that FF-TMR is preferable for all designs, technologies and weights considered (the table for the cost-effectiveness is not reported for sake of brevity). This is due to the large area overhead of FFD-TMR for $C_d$ and to the performance overhead of FFD-TMR for $C_p$. Even considering the weights and type of technology for which FFD-TMR is less expensive ($C_d$ and SD) and reducing the overhead compared to FF-TMR by 50% (e.g. $\Delta T_{ex}/T_{ex} = 0.27$), FFD-TMR is still less cost-effective than FF-TMR. However, the cost of both FF-TMR and FFD-TMR is positive for any design/technology/weight combinations, showing that they are both expensive types of redundancy in general.

To reduce the cost of redundant sequential elements, different designs of sequential elements have been proposed to replace FFD-TMR and FF-TMR. For instance, a DICE-FF cell has a reduction of 61.54% in terms of area, between 40.30% and 48.72% in terms of power (depending on the switching activity) and 15.13% in terms of delay compared to a sequential element of FF-TMR [96]. However, while FF-TMR uses three simple FFs and a voter (and therefore in principle could be implemented in RTL) as a redundant cell, FFD-TMR and other designs require technology-specific adjustments at layout and electrical level within the sequential element. For instance, the DICE-FF requires the design of a custom cell not available in commercial technologies [96].

Table 3.12: Relative changes [%] in $\lambda_w$ and U for redundancies protecting logic. In bold the most effective redundancy for each design/technology combination.

| Redund. | Design | $\frac{\Delta\lambda_{w,LC}}{\lambda_{w,LC}}$ | $\frac{\Delta\lambda_{w,AC}}{\lambda_{w,AC}}$ | $\frac{\Delta\lambda_{w,HC}}{\lambda_{w,HC}}$ | $\frac{\Delta\lambda_{w,SD}}{\lambda_{w,SD}}$ | $\frac{\Delta\lambda_{w,MD}}{\lambda_{w,MD}}$ | $\frac{\Delta U_{LC}}{U_{LC}}$ | $\frac{\Delta U_{AC}}{U_{AC}}$ | $\frac{\Delta U_{HC}}{U_{HC}}$ | $\frac{\Delta U_{SD}}{U_{SD}}$ | $\frac{\Delta U_{MD}}{U_{MD}}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| FF-TMR | LE-1 | **-2.2** | -1.7 | -0.9 | -1.5 | -1.3 | **-3.7** | -3.0 | -2.5 | -2.5 | -2.6 |
| | HE-1 | **-7.1** | -6.7 | -3.2 | -5.1 | -4.3 | **-8.9** | -7.1 | -3.5 | -8.2 | -6.1 |
| FFD-TMR | LE-1 | **-2.2** | **-3.0** | **-2.8** | **-3.8** | **-0.9** | **-3.7** | **-4.1** | **-3.2** | **-4.8** | **-1.5** |
| | HE-1 | **-7.1** | **-8.1** | **-8.1** | **-11.1** | **-2.8** | **-8.9** | **-9.8** | **-7.9** | **-11.7** | **-4.9** |

Table 3.13: Cost of redundancy in mixed logic in the case of FF-TMR and FFD-TMR. In bold the most cost-effective redundancy for each design/technology combination.

| Redund. | Design | $C_{d,LC}$ | $C_{d,AC}$ | $C_{d,HC}$ | $C_{d,SD}$ | $C_{d,MD}$ | $C_{p,LC}$ | $C_{p,AC}$ | $C_{p,HC}$ | $C_{p,SD}$ | $C_{p,MD}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| FF-TMR | LE-1 | **0.34** | **0.35** | **0.36** | **0.35** | **0.36** | **0.43** | **0.44** | **0.44** | **0.44** | **0.44** |
| | HE-1 | **0.35** | **0.37** | **0.44** | **0.37** | **0.40** | **0.50** | **0.51** | **0.54** | **0.51** | **0.52** |
| FFD-TMR | LE-1 | 0.48 | 0.47 | 0.48 | 0.46 | 0.52 | 0.86 | 0.85 | 0.86 | 0.85 | 0.87 |
| | LE-1 | 0.57 | 0.55 | 0.57 | 0.50 | 0.65 | 1.01 | 1.00 | 1.06 | 1.03 | 1.04 |

### PROTECTING SIMULTANEOUSLY SMALL SRAM ARRAYS AND LOGIC

Table 3.14 shows the relative changes in terms of $\lambda_w$ and $U$ when applying C-DMR and C-TMR. Based on these results, Table 3.15 shows the comparison between C-TMR, C-DMR, and the most cost-effective solution found combining the results from Table 3.11 and FF-TMR. It shows that protecting separately RFs and logic and replicating the core have in general similar cost-effectiveness, so they are both viable solutions. As a trend, FF-TMR and RF-ECC are more cost-effective for (older) technologies with relative low fraction of MBUs and high masking of SETs (i.e. LC, AC) and when the focus is on dependability. For (newer) technologies higher fraction of MBUs and SETs sampled C-DMR is more cost-effective. In case the focus is on performance ($C_p$), C-DMR is found as the most cost-effective solution regardless of other parameters. C-TMR is generally the least cost-effective solution because of a larger area and power overhead. It becomes more cost-effective than C-DMR only in case the weight of the availability is higher ($\epsilon$=5 instead of 1 for $C_d$).

Table 3.14: Relative changes in $\lambda_w$ and U [%] for redundancies protecting both small SRAM array and logic for different technologies and designs. In bold the most effective redundancy for each design/technology combination. It should be noted that the model defined in this chapter does not find any difference between C-TMR and C-DMR in terms of relative change of failure rate. In order to differentiate the effects of the two approaches, Common Cause Failures (e.g. due to issues on power or shared clock signals among the replicas [285]) should be considered.

| Redund. | Design | $\frac{\Delta\lambda_{w,LC}}{\lambda_{w,LC}}$ | $\frac{\Delta\lambda_{w,AC}}{\lambda_{w,AC}}$ | $\frac{\Delta\lambda_{w,HC}}{\lambda_{w,HC}}$ | $\frac{\Delta\lambda_{w,SD}}{\lambda_{w,SD}}$ | $\frac{\Delta\lambda_{w,MD}}{\lambda_{w,MD}}$ | $\frac{\Delta U_{LC}}{U_{LC}}$ | $\frac{\Delta U_{AC}}{U_{AC}}$ | $\frac{\Delta U_{HC}}{U_{HC}}$ | $\frac{\Delta U_{SD}}{U_{SD}}$ | $\frac{\Delta U_{MD}}{U_{MD}}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| C-TMR | LE-1 | **-5.8** | **-6.6** | **-7.4** | **-7.4** | **-5.8** | **-5.1** | **-5.7** | **-6.3** | **-6.3** | **-5.1** |
| | HE-1 | **-10.0** | **-12.2** | **-14.4** | **-14.4** | **-10.0** | **-10.8** | **-12.7** | **-14.6** | **-15.2** | **-10.8** |
| C-DMR | LE-1 | **-5.8** | **-6.6** | **-7.4** | **-7.4** | **-5.8** | 4.5 | 4.6 | 4.8 | 4.8 | 4.5 |
| | HE-1 | **-10.0** | **-12.2** | **-14.4** | **-14.4** | **-10.0** | 5.8 | 3.7 | 2.6 | 6.9 | 5.2 |

Table 3.15: Comparison of cost-effectiveness of C-TMR, C-DMR and the most cost-effective solution from Table 3.11 and FF-TMR. In bold the most cost-effective redundancy for each design/technology combination.

| Redundancy | Design | $C_{d,LC}$ | $C_{d,AC}$ | $C_{d,HC}$ | $C_{d,SD}$ | $C_{d,MD}$ | $C_{p,LC}$ | $C_{p,AC}$ | $C_{p,HC}$ | $C_{p,SD}$ | $C_{p,MD}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| C-TMR | LE-1 | 0.52 | 0.50 | 0.49 | 0.49 | 0.52 | 0.59 | 0.58 | 0.57 | 0.57 | 0.59 |
| | HE-1 | 0.63 | 0.59 | 0.55 | 0.54 | 0.63 | 0.76 | 0.75 | 0.73 | 0.73 | 0.76 |
| C-DMR | LE-1 | 0.34 | **0.34** | **0.33** | **0.33** | **0.34** | **0.36** | **0.36** | **0.36** | **0.36** | **0.36** |
| | HE-1 | 0.43 | 0.39 | **0.36** | **0.40** | **0.43** | **0.48** | **0.46** | **0.45** | **0.47** | **0.48** |
| Tab 3.11+FF-TMR | LE-1 | **0.32** | 0.35 | 0.37 | 0.34 | **0.34** | 0.48 | 0.49 | 0.50 | 0.49 | 0.50 |
| | HE-1 | **0.36** | **0.37** | 0.42 | 0.36 | 0.44 | 0.58 | 0.59 | 0.61 | 0.58 | 0.60 |



## 3.6. EXPECTED IN-ORBIT BEHAVIOR AND VALIDATION

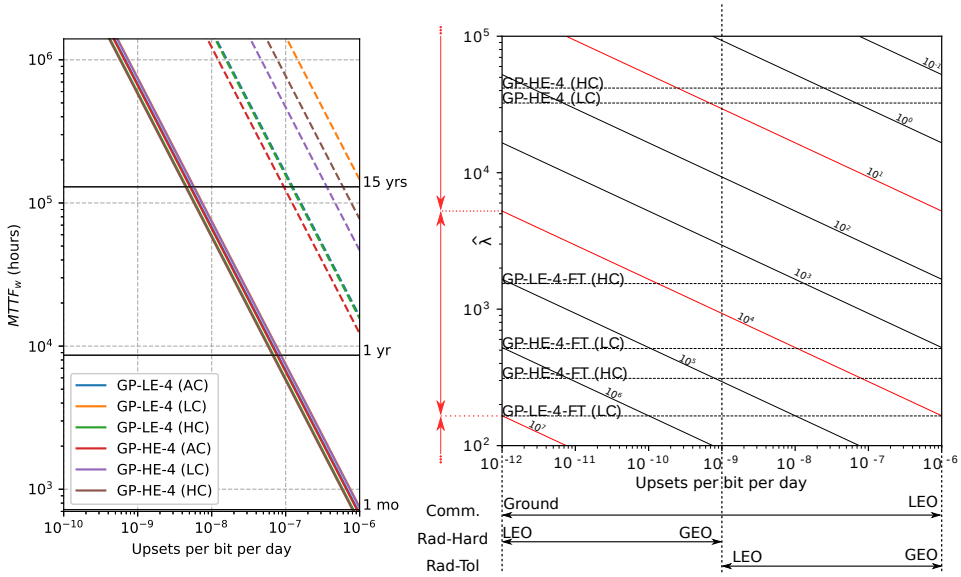


Figure 3.10: On the left, $MTTF_w$ for LE-4 and HE-4 in different technologies. Solid lines indicate unprotected versions and dashed lines versions with the most cost-effective redundancy found according to $C_d$. On the right, $MTTF_w$ in hours for different designs depending on $\hat{\lambda}$ and $\lambda_{ev}$. (FT indicates a "Fault-Tolerant" implementation, i.e. employing the most cost-effective redundancies found with $C_d$). Red lines indicate different classes of designs in terms of $MTTF_w$.

Fig. 3.10 (left) shows the absolute $MTTF_w$ of LE-4 and HE-4 before and after the most cost-effective solutions according to $C_d$ are employed. It is worth to note that, while from a quantity perspective the vulnerability is roughly the same for all types of technologies and quad-core designs (LE-4 and HE-4), the quality of the vulnerability is so different that applying redundancy produces very different $MTTF_w$ (around one order of magnitude of difference). Fig. 3.5 shows that this is the case because caches dominate the $MTTF_w$ and have small variations in terms of vulnerability changing the type of technology. Also, the comparison between LE-4-FT/HE-4-FT and LE-1-FT/HE-1-FT in Fig. 3.11 shows that going multicore has a large cost in terms of availability, e.g. for $\lambda_{ev} = 10^{-7}$ up-

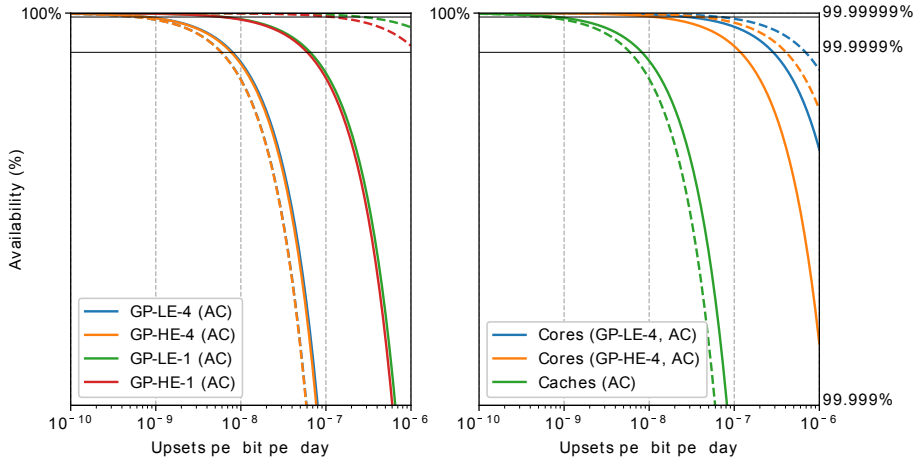


Figure 3.11: On the left, availability of LE-4 and HE-4 (AC). On the right, decomposition in core (RFs+logic) and caches. Solid lines indicate unprotected versions and dashed lines versions with the most cost-effective redundancy found for each combination of design/technology. The unavailability has been estimated with Eq. 3.8.

sets/bit/day LE-4-FT and HE-4-FT cannot meet an availability target of 99.999%, while both LE-1-FT and HE-1-FT can meet a 99.99999% target. This findings show that techniques to reduce the vulnerability of L2Cs, e.g. employing a $L2C_{WT}$ to lower the AVF of the [253] and to decrease the $T_{eh,due}$, may be a cost-effective solution to increase the $MTTF_w$ and the availability of quad-core processors.

When the focus is on a target $MTTF_w$ instead of cost-effectiveness, a chart similar to Fig. 3.10 (right) can be employed to evaluate possible trade-offs. Once the $MTTF_w$ target is set, a combination of microarchitecture and redundancy will fit only if a technology (for the target environment) exists for which the horizontal line corresponding to the microarchitecture/redundancy is below the oblique line representing the $MTTF_w$ target. Assuming a target of $MTTF_w = 10,000$ hrs (1.14 years), the combinations below the respective line are:

- HE-4-FT (HC) (C-DMR + EDC/ECC) on RH technology in LEO (HC, $\lambda_{ev} \leq 10^{-10}$ upsets/bit/day).
- HE-4-FT (LC) (FF-TMR + RF-ECC + ECC/ECC) on rad-hard technology in LEO and GEO or rad-tol (radiation-tolerant) in LEO (the latter only up to $\lambda_{ev} = 10^{-8}$ upsets/bit/day).
- HE-4-FT (HC) (C-TMR + ECC/ECC) on rad-hard technology in LEO and GEO or rad-tol in LEO (the latter only up to $\lambda_{ev} = 10^{-7}$ upsets/bit/day).
- LE-4-FT (LC) (FF-TMR + RF-ECC + ECC/ECC) for commercial technologies in LEO, and both rad-hard and rad-tol technologies both in GEO and LEO.

Interestingly, Fig. 3.10 (right) shows that processors without redundancy achieve low $MTTF_w$ with commercial technology at ground level. This is reflected by the trend of including redundancies in processors for terrestrial applications, mainly in caches [84] and sometimes also in RFs [83]. However, the figure also shows that designs intended for space operate also at $\lambda_{ev} \geq 10^{-10}$ upsets/bit/day and therefore they require more redundancy. Fig. 3.10 (right) also shows that having a limited range for $\lambda_{ev}$ implies that it is not possible in general to make a certain design reliable enough to achieve an arbitrary $MTTF_w$ target using a rad-hard technology. Therefore, processors in Fig. 3.10 (right) can be binned in three classes in terms of $MTTF_w$ using non-overlapping $MTTF_w$ isolines (e.g. $10^7$, $10^4$ and 10 hours, shown in red). For instance, none of the design/redundancy combinations can meet a target $MTTF_w$ higher than $10^7$ hours and there is a higher level of $MTTF_w$ for which quad-core processors are not fit for any design/redundancy combination and the designer must resort to smaller implementations.



### 3.6.1. VALIDATION

The most rigorous method to validate a processor for usage in space is radiation testing [81, 286]. The main advantage of radiation testing is that it can reproduce exactly the physical mechanisms that will be experienced by the device in space. For this reason, radiation testing can be used both to validate the design and as a validation of the error models employed to select the redundancy (e.g. fraction of MBUs and of SETs sampled).

Sometimes FI is proposed as a validation method. However, FI is not capable of validating the fault models (e.g. percentage of MBUs) as the model of fault injected is arbitrary. On the other hand, radiation testing typically has difficult controllability and observability [101]. Therefore, it is hard to pinpoint where the error was generated.

Because observability is limited, usually the outcomes of radiation tests are binned according to a simpler classification compared to the one shown in Fig. 3.3. Typically a run is either successful or the run is interrupted by a particle strike which leads to a temporary non-functionality (or interruption of normal operation) of the affected device. In the latter case it is said that a Single Event Functional Interrupt (SEFI) happened.

Furthermore, in order to achieve meaningful statistics in a limited time, sometimes the flux of particles employed during radiation testing is several orders of magnitude higher than in space. This can produce artifacts, as, for instance, the probability of accumulation of two errors in the scrubbing period is much larger than in space. A field example is provided in [287], where the beam flux had to be throttled down in order to allow error handling in caches to complete successfully and to allow the logging of errors.

Several works in literature (e.g. [143, 287]) compare failure rates from FIs and simulations to failure rates during beam testing. The most severe underestimation found in literature is of a factor 20× [65] compared to data from radiation tests. However, this value has been found by simply multiplying the AVF by the population of FFs, so this can be seen as an upper boundary of the possible underestimation. For instance, in [143] the underestimation of the AVF compared to radiation tests is of 11× and the expected failure rate on the field lies in between these two values. This suggests the adoption of a safety factor of at least 10 when setting a target in terms of $MTTF_w$ and to prefer radiation testing for validation, as it provides a worst case estimation. However, it is shown in [287] a similar AVF for FI, protons tests and neutrons tests (respectively 5.02%, 4.35%,

2.65%) when the flux is tuned down enough.

Typically after radiation test new processors are validated in space with an IOD mission. Data of the behavior of processors in space are not common in literature. Data from [54] shows in-orbit statistics for six identical LEO satellites. The average number of reset is 4.67 reboots per year for each satellite, with an $MTTF_{DUE}$ of 2.57 months. This reflects roughly the models described in this chapter for a LE-1 (typically employed as OBC as shown in Chapter 2) for technologies with $\lambda_{ev}$ ranging from $10^{-8}$ to $10^{-7}$ upsets/bit/day (rad-tol technology in LEO).



## 3.7. SUMMARY

A summary of the framework (containing the description of each step, the references to sections, figures and equations in the paper and possible adaptations or extensions) is reported in Table 3.16.

Table 3.16: Summary of the framework presented and adaptations required for use with different designs.

| # - Step | Ref. | Possible adaptations/extensions |
|---|---|---|
| 1 - Definition of fault and error models | Sec. 3.3.1 | Use of $SF_{SET\%}$, $SF_{FF\%}$, $SBU_{\%}$, $MBU_n$, and $\lambda_{ev}$ for specific technology and frequency. |
| 2 - Definition of failure models | Sec. 3.3.3<br>Fig. 3.3 | Use of QoS to distinguish between acceptable and unacceptable behavior. |
| 3 - Estimation of $N_{eq}$ or $SER$ | Sec. 3.4.2<br>Eq. 3.1 | MCPath can be employed for closed source processors (only total area). |
| 4 - Estimation of $AVF$ | Sec. 3.4.3 | Extended microarchitectural simulators (e.g. [274]) or fault injection on FPGA prototype. |
| 5 - Estimation of $\hat{\lambda}$ and $\hat{U}$ | Sec. 3.4.1<br>Eqs. 3.4-3.8 | Use of alternative derating factors and decompositions to convert SER to $\hat{\lambda}$ (e.g Sec. 3.4.4). |
| 6 - Application of redundancy | Sec. 3.5<br>Eqs. 3.14-3.17 | Requires estimation of $P_{det}$. If $AVF'$ and $\Delta N_{eq}/N$ are unknown, they can be approximated respectively as $AVF$ and 0. |
| 7 - Analysis of cost-effectiveness | Sec. 3.5<br>Eq. 3.11 | Sensitivity analysis on weights and technical parameters ( $\Delta T_{ex}/T_{ex}$, $\Delta A/A$, $\Delta P/P$). |
| 8 - Meet requirements | Sec. 3.6<br>Figs. 3.10-3.11 | Use of more complex optimization algorithms (e.g. [237]) to minimize cost of achieving $MTTF \leq MTTF_{target}$ and $U \leq U_{target}$. |
| 9 - Validation | Sec. 3.6.1 | SEE radiation tests (protons, heavy ions) and IOD missions. |

This chapter provides a framework to evaluate the fitness of a microarchitecture for a certain space environment or any other environment where the failure rate is dominated by soft errors. This framework allows to include considerations on soft errors when selecting and configuring an open-source IP core like most of those based on the RISC-V ISA.

Models to evaluate the vulnerability of different processor units and evaluate the cost-effectiveness of redundancy in terms of penalties in area, performance, power and availability for several case studies were introduced. However, the framework can be easily adapted to different designs and data for a specific technology can be employed to model a specific implementation. Furthermore, the chapter also provides tools to find the microarchitecture/redundancy/technology combinations which meet specific $MTTF_w$ and availability requirements.

From the models developed, technology and microarchitecture are the factors impacting the dependability of a processor the most. Furthermore, this work also highlights that estimations of AVF are not the only concern when characterizing the depend-

ability of processors, as other parameters influence the final dependability of the design (e.g. total area, the ratio between sequential and combinational area, temporal masking, etc.) in a comparable way. Caches are shown to be the most vulnerable structures (especially in multi-core processors) and therefore information redundancy in caches is typically very cost-efficient. However, it can be expensive in terms of availability for particular distributions of MBUs for which the number of uncorrectable errors is high. Furthermore, scrubbing has low efficacy in caches (as opposed to when dealing with large external memories), as accumulation in caches has negligible effects compared to MBUs.



Work is still required to characterize the SER in space of ASIC technologies below 28 nm (for instance in terms of fraction of MBUs for unprotected FFs and FF-TMR) and some specific relationships between AVF and microarchitectural choices (for instance the effect of different microarchitectures on the AVF of caches). Furthermore, at the moment of writing no validated extended microarchitectural simulators to estimate soft error vulnerability supporting the RISC-V ISA are available to the public.

# 4

# Analysis of Workloads for On-Board Decision Making

*This is a nightmare, which will pass away with the morning.*
*For the resources of nature and men's devices are just as fertile and productive as they were.*

John Maynard Keynes, Essays In Persuasion (1931)

*The use of Deep Neural Networks (DNNs) in terrestrial applications went from niche to widespread in few years, thanks to relatively inexpensive hardware for inference and large datasets available. The applicability of this paradigm to space systems, where both large datasets and inexpensive hardware for on-board inference are not readily available, is more difficult and thus still rare. Furthermore, when large amounts of labelled data are not available, other machine learning approaches, like unsupervised learning algorithms, might perform better. Nevertheless, given the high potential of DNNs, this chapter analyzes the impact of DNNs and other machine learning approaches on On-Board Decision Making (OBDM) capabilities of space systems, identifying the specific criticalities of deploying Artificial Intelligence (AI) on-board satellites. Although several types of DNNs are introduced, the focus will be on CloudNet, a Fully Convolutional Network (FCN) for cloud detection.*

Parts of this chapter have been published in [288].

## 4.1. INTRODUCTION

The success of 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 training and inference in reasonable timescales, for instance GPUs [11]. 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 to its commercial counterpart.

One of the main issues in terms of hardware design 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 [289]. For instance, the GPU tested in [290] is reported to fail (due to SEFI or SEL) during an irradiation with a high energy proton beam on average every 43 s. The main reason behind this very low MTTF is that GPUs are much larger (e.g. they can have 2.2 billion transistors[1], which corresponds to roughly 550 MGE if four transistors per GE[2] are assumed) than single-core single-issue processors (890 kGE for the one in [59]) typically employed in space.

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 non-demanding control and housekeeping operations, while on-board data processing is typically a not attractive solution, since 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. encryption or compression).

This chapter analyzes the benefits that DNNs can provide at the system level (Sec. 4.2) and the feasibility of the deep learning approach for space applications (Sec. 4.3). Then, an analysis of the software workloads required for DNNs is carried out in Sec. 4.4. Finally, Sec. 4.5 concludes with a summary of the main findings and several recommendations to systematically enable OBDM with RISC-V vector processors in the medium term.

The software workloads introduced in this chapter will be employed in Chapter 6 to evaluate the effectiveness of the RISC-V Vector Extension (RVVE) in speeding up DNNs.

## 4.2. SYSTEM-LEVEL IMPACT

While GEO satellites can continuously communicate with the ground station, LEO satellites can only communicate with the ground station periodically (unless a relay satellite is employed [291]), in some cases with large periods between contacts [292]. In this way, the satellite may enter an unsafe state and the ground operator in the worst case can only intervene hours later.

[1] https://www.techpowerup.com/gpu-specs/radeon-e9173-pcie.c3031

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

However, there is a trend of launching several LEO satellites to compose constellations and mega constellations [293], with the possibility of mitigating the risk of failure of a single satellite at the constellation level and replace them if they fail (as they are much cheaper than large GEO satellites). Therefore, in this case there is a trade-off to be made between dependability of a single satellite, its cost and number of spare satellites in a constellation. Furthermore, if a higher cost can be accepted compared to the case where a single ground station is employed, more than one base station may be available, thus reducing the period between contacts.

Furthermore, space systems are inherently constrained in terms of available power (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 [151].

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 it will be shown how OBDM can help mitigating these shortcoming of LEO satellites. Then, in Sec. 4.3 the feasibility of applying DNNs to these problems is investigated.



### 4.2.1. DOWNLINK EFFICIENCY

In [151] 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. On the other hand, in [151], an image from a simple VGA camera with 640×480 pixels has a file size of 900 KiB and a data cube from a hyperspectral sensor with 32 bands and 1024×1024 pixels requires 32 MiB. In LEO a potential access duration to the ground station can be assumed to be 5 minutes every pass, and in this time span only 64 VGA images or roughly half a cube of 32 bands can be transmitted to the ground. Considering the 6U CubeSat described in [152], the power budget for the downlink limits the data rate to 14 Mbps, while its spectrometer (with five spectral bands) generates 255 Mbps. Without considering data compression, during a single ground station pass[3] only up to 31.5 s worth of imaging data can be downlinked, which is only around 0.6% of the data that could be theoretically collected during a typical LEO orbit (around 90 minutes in [294]). 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 ground.

#### BENEFITS OF DATA REMOVAL AND COMPRESSION

Given the constraints discussed above, many on-board processing applications are not of interest in space systems. For instance, noise filtering can be executed on ground with cheaper hardware, while on the other hand data compression is already deployed in many missions (e.g. in [152] a 2:1 compression is employed) because it mitigates the bottleneck of the downlink. Efficiency of the downlink can be increased even further, removing useless data instead of sending it to ground (i.e. data removal [5]). For instance, in the Landsat datasets [295], 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

[3] A pass is a period of time a spacecraft is visible to a ground station.

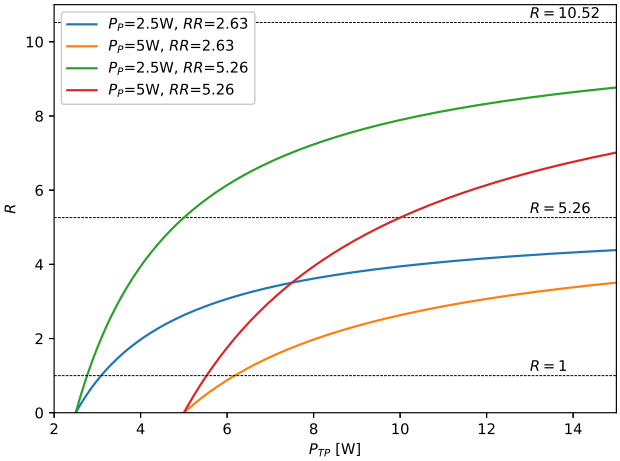


Figure 4.1: Ratio $R$ between useful data transmitted with and without data removal as a function of $P_{TP}$ allocated for the transmission subsystem and data removal for different $P_P$ and $RR$. In all cases 2:1 compression is assumed.

with less than 10% of cloud cover results in average in a 2.63x data reduction. Combining data removal with a 2:1 compression, the amount of useful data sent increases by 5.26× compared to a system without on-board data processing.

### 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 (as shown in Fig. 2.1). While for the mass memory technologies with long retention time and low power dissipation can be employed (e.g. flash 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) and Double Data Rate 3 (DDR3), depending on the radiation resilience/performance trade-off required [296]. In the datasheet [297] of the 1Gb DDR2 DRAM tested in [296], a peak power consumption of around 0.5 W is reported, which will be taken here as an estimation of power consumption for DRAMs of this capacity, and 1 W for the most powerful version of the 22-nm FD-SOI vector processor ASIC in [19] running a peak-performance application. A requirement of 1 GiB of main memory will be assumed (in [298] the memory footprint for DNNs ranges from 645 MiB to 1.49 GiB), while the cost in terms of power $P_P$ of applying data reduction and compression will be assumed to be 5 W.

Assuming a common amount of power allocated for the transmission and data processing subsystems ($P_{TP}$), the following equation can be employed [151] to estimate the amount of useful data transmitted per station contact $D_C$ when data is not processed on board:

$$D_C = \frac{P_{TP}}{RR} \cdot k, \tag{4.1}$$

where $k$ is a constant (dependent on the transmission subsystem, receiver, propagation and required BER) and $RR$ is the optimal Removal Rate, i.e. the ratio between useful data and data produced by the payload. When only useful data is selected and a data compression of CR:1 is applied, the amount of useful data transmitted is instead:

$$D_C = CR \cdot (P_{TP} - P_P) \cdot k. \tag{4.2}$$

The ratio $R$ between the amount of useful data trasmitted in the two cases is then:

$$R = RR \cdot CR \cdot \left( \frac{P_{TP} - P_P}{P_{TP}} \right), \tag{4.3}$$

which for $P_{TP} >> P_P$ tends to its maximum, i.e. $RR \cdot CR$. This means that data removal is more effective for larger satellites, which have more power available for transmission and processing.

To understand what the effect of a more power-efficient solution would be, in Fig. 4.1 it is shown the ratio $R$ as a function of the power budget $P_{TP}$ for two different values of power spent for data processing $P_P$ (5 W and 2.5 W). While the fraction of the maximum ratio achieved for a certain $P_{TP}$ is independent from $RR \cdot CR$, 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.

#### FURTHER DEVELOPMENTS

Having a DNN that classifies pixels in covered by cloud or not covered by cloud is a basic form of semantic segmentation of an image, i.e. each pixel is classified depending on a particular label. While this is fine in some applications like cloud detection [299] or landscape classification (classified in buildings, roads, water and crops [300]), some other applications require to differentiate among different instances of a certain object. In this case, object detection may be employed, where each object of interest is identified and its location is marked with squared boxes. For instance, in [301], boats are detected with this method. The final objective of this approach will be to have images where the objects are identified and sent to ground on demand. For example, a user from ground station may ask all the images of boats in a certain sea area or buildings in a large rural area of interest. Sending the images of this large areas to the ground may require several orbits, while sending only the images of the objects requested maximizes downlink efficiency.

### 4.2.2. ON-BOARD VIRTUAL OPERATOR

In [294] it is considered the case of a LEO satellite with an orbit period of 90 minutes[4] and that access to the ground station can be booked every 3 orbits, for 5 minutes per

[4] LEO satellites have a circular or elliptical orbit with an altitude ranging from 250 to 2000 km (measured from the Earth surface). The orbit period varies with the altitude and ranges from 90 (lower LEO orbits) to 120 minutes (higher LEO orbits) [149].

contact. However, even when a base station is always available, most LEO satellites for Earth Observation are in a Sun-Synchronous Orbit (SSO). While a ground station close to the pole will see the satellite every orbit, other ground stations at mid latitudes will not see the satellite every orbit. For instance, for TU Delft's ground station in Delft (52° N) the satellite will be seen three times in the morning, followed by a long period of lack of any contacts and again three times in the evening. Even if there is a pass, communication with the satellite may still not be possible because of issues in the link, for instance due to the elevaltion of the satellite during the pass.

In this section, to provide an evaluation for two extreme cases, two opposite cases are considered:

- Polar station: a contact with the ground station either 5 minutes every orbit (6% of the orbital period)
- Mid-latitude station: three consecutive 5 minutes contacts every orbit followed by a gap without contact of 9 hours (contact possible for 2.1% of the day)

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. DNNs can be employed to predict the telemetry of the next orbit given the previous one (or more) [302]. This can be used to help diagnose anomalous behaviors before the next contact with the ground station [302].

#### BENEFITS OF AN ON-BOARD VIRTUAL OPERATOR

Assuming a constant failure rate $\lambda$ (typical of soft errors, as shown in Chapter 3), the reliability of the spacecraft after the end of the $i^{th}$ contact and before $(i+1)^{th}$ contact can be expressed as $R(t) = R(t_i) \cdot e^{-\lambda(t-t_i)}$, where $t_i$ is the time instant of the end of the $i^{th}$ 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[5] $S(t)$ 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 the failures are detected with a certain Detection Factor ($DF$), then:

$$S(t) = R(t_i) \cdot e^{-(1-DF)\cdot\lambda(t-t_i)} \tag{4.4}$$

when not in contact. Fig. 4.2 shows an example for $DF = 0.9$, a $\lambda$ of $10e^{-4}$ failures/min (approximately one failure per week) for a polar station and a mid-latitude station. The average safety increases from 99.60% to 99.96% in the first case and from 97.95% to 99.79% 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%).

[5] In this case, safety will be defined as the probability that the satellite is either working in nominal conditions or it is in a safe state because of a detected failure.

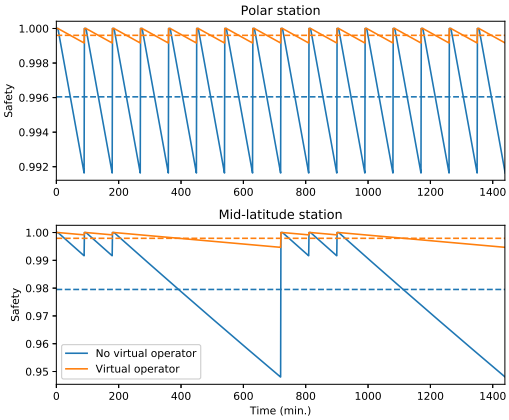


Figure 4.2: Increase of safety for a virtual operator with $DF = 0.9$ for a polar and a mid-latitude station. Dashed lines represent average values over time.



#### COST OF THE REQUIRED HARDWARE

The most attractive solution to deploy telemetry analysis and forecasting is to use an enhanced version of a typical OBC. As a matter of fact, the requirements for this kind of applications are less stringent compared to DNNs for image analysis. In [302], telemetry forecasting was implemented with a 64Mb DRAM on a single core reaching 661 predictions per second, meaning that all the parameters of the satellite telemetry in [302] (13216 in total) can be predicted in around 20 s. Furthermore, enhancing a general purpose processor with minimal vector facilities (i.e. 2 lanes) increases power consumption of the processor from 52 mW [59] to 138 mW [19] (+65%).

## 4.3. ALGORITHMS

### 4.3.1. SUPERVISED LEARNING

The most common problems in machine learning can be solved learning a function, given a set of examples of inputs and golden outputs (training). In this case supervised learning algorithms are employed. Among this type of algorithms, 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 [303]. 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 [304], where CNNs are trained for multi-class classification with different datasets sizes. It is shown that for a 3-class classification problem using 5,000 images per class achieves 97% accuracy[6] on average, while bringing the training set down to 1,000 images per class lowers accuracy to 74% on average.

[6] In this case, to evaluate the performance of the DNN, accuracy is defined as the ratio of the number of correct classifications on the total number of classifications. In some cases, where some classes are much more rare in the data set, more refined metrics considering true positives and true negatives are employed, like recall and precision [305].

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% (9 classes) [304].

One of the most popular datasets for terrestrial applications is ImageNet[7]. It is a large image dataset typically used to asses the effectiveness of a certain neural network architecture for image classification, containing RGB images of 256×256 pixels for a total of 1000 classes [303]. There are $1.3 \cdot 10^6$ training images (ranging from 732 to 1300 per class) and $10^5$ test images [303]. 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[8], which provides hyperspectral images composed of 11 bands ranging from ultra blue to thermal infrared. A large number of land cover classification solutions were developed on subsets of the Landsat datasets [306]. Setting up a reference, standardized dataset is instead more difficult for more specific applications, especially those involving housekeeping data, like telemetry forecasting or anomaly detection. Design parameters like orbits, observed signals and nominal values change from mission to mission. Furthermore, space industry primes (i.e. large system-level integrators) have very restrictive data policies concerning open access to telemetry data. However, some datasets containing telemetry of housekeeping data are available to the public, like those of the GOCE mission[9]. 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 [302].

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 is available. As reported in [293], existing and planned constellations comprise hundreds to thousands satellites (e.g. 4200 for the planned constellation from Samsung) thus making the use of DNNs potentially very effective also for mission-specific parameters.

### 4.3.2. UNSUPERVISED LEARNING

Supervised learning is not directly applicable when the dataset is not large enough or it is not labelled. For instance, in anomaly detection the goal is to classify telemetry segments according to whether they contain an anomalous behavior or not [307]. In this case, it is possible that there are not have enough faulty behaviors to train a DNN (e.g. a couple of failures after years and years of telemetry). A possible approach is to use unsupervised learning, where the algorithm itself analyzes the structure of the data to find anomalies [308].

[7] http://www.image-net.org
[8] https://www.usgs.gov/land-resources/nli/landsat/landsat-8
[9] http://eo-virtual-archive1.esa.int/GOCE_TLM_HK.html

## 4.4. WORKLOADS

In this section, the processor workloads to be executed in order to apply algorithms of interest for OBDM are investigated. A particular focus will be on CloudNet, as a representative image segmentation algorithm. The other type of applications identified in Sec. 4.2, i.e. telemetry analysis, was analyzed and implemented in detail in [302].

### 4.4.1. CLOUDNET

As a case study of DNN for image analysis, the public code[10] of CloudNet will be employed [299]. It is a Fully Convolutional Network (FCN) [309] for cloud detection, i.e. its output is a mask of the same size as the input image indicating the pixels covered with clouds. The use of a FCN instead of a CNN is due to the fact that the output of a FCN is an image, making it particularly fit for semantic segmentation. On the other hand, CNNs are more fit for object detection or image classification (e.g. defining the percentage of the pixels covered in clouds rather than the position of each pixel). Furthermore, while FCNs are composed by a contracting branch, responsible for extracting features and producing deep low-level features of the input image, and an expanding branch, to utilize those features to generate the output mask [299, 309], CNNs are only composed of the contracting branch, typically followed by one or more Fully Connected layers[11]. Therefore, the computational workload of a CNN can be assumed to be roughly half the computational workload of a comparable FCN.

As any other DNN, CloudNet is composed by a sequence of layers[12] operating on the output data of one or more layers. Analyzing the model in Keras[13], it can be found that CloudNet contains 38 convolutional layers (of which 5 are transposed), 15 addition layers, 31 batch normalization layers, 46 standalone activation layers (plus two activation layers embedded in the first and last convolutional layers), and 53 concatenate layers. To give an idea of the contribution of each of these layers, the execution of the model on a quad-core Intel i7-6600U was profiled. The breakdown of the execution type for each type of layer is shown in Fig. 4.3 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 main memory utilization of 836.65 MiB.

When considering the input layetr, in the case of CloudNet four spectral bands of the large images of Landsat 8 (e.g. 7621×7791 pixels) are divided in non-overlapping patches of 384×384 pixels, which are then downsampled to 192×192 pixels. Therefore the input layer size is 192×192×4.

#### CONVOLUTIONAL LAYERS

As straightforward software implementations of convolutions achieve low performance, performances are typically improved unrolling the convolutions into matrix-matrix multiplications [310]. Fig. 4.4 shows that applying a convolutional layer with $N$ kernels of size $C \times J \times K$ to an input of dimensions $C \times W \times H$ generates an output of $N$ matrices of

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

[11] Although in some works like [299] FCNs are called CNNs, we use this term for them, which highlights their lack of FC layers.

[12] A layer is a set of 'nodes' performing a certain function on input data.

[13] https://keras.io

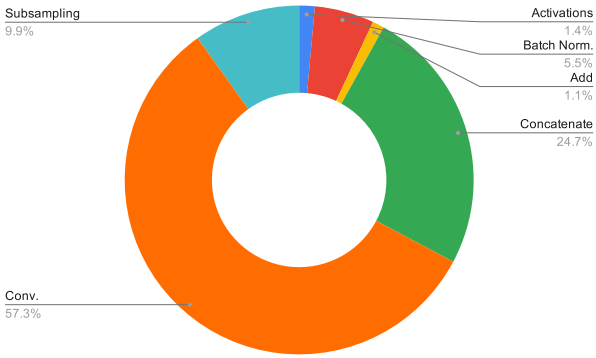




Figure 4.3: Breakdown of the execution time for an inference of CloudNet.

size $U \times V$ [186], with $U$ and $V$ depending on the stride $S$ and padding $P$ of the convolutional layer with the equations [311]

$$U = \lfloor (W - J + 2P)/S \rfloor + 1 \tag{4.5}$$

and

$$V = \lfloor (H - K + 2P)/S \rfloor + 1. \tag{4.6}$$

In this case, the number of FLOPs for each layer is estimated as $\#FLOP = 2UVNCJK$, given that there are $UVN$ output coefficients and for each of them $CJK$ multiplications and accumulations are required. The read traffic from memory is $MT_R = 4(NCJK + UVCJK)$ and the write traffic to memory is $MT_W = 4NUV$, while the total memory traffic is the sum of the two.

Further performance enhancements can be obtained by mapping the matrix-matrix multiplication with optimized libraries. In [310] it is shown that using Basic Linear Algebra Subroutines (BLAS) instead of coding the unrolled version from scratch produces a speedup ranging from 2.43x to 3x depending on the architecture and on the input size. Using BLAS subroutines, matrix-matrix multiplications are mapped to the *sgemm* subroutine[14], which (in its non-transposed form) implements the following algorithm:

$$C \leftarrow \alpha A \cdot B + \beta C \tag{4.7}$$

where $A$, $B$, $C$ are matrices of respectively size $n_1 \times n_2$, $n_2 \times n_3$ and $n_1 \times n_3$, and $\alpha$ and $\beta$ are scalars, with $n_1 = N$, $n_2 = CJK$ and $n_3 = UV$. Fig. 4.5 proposes a C function for *sgemm* assuming $\alpha = \beta = 1$. For brevity, all the variables of the listings in this chapter

[14] BLAS routines are typically employed for floating operations and a letter is added to the name of the routine according to the precision (e.g. *sgemm* for single precision, *dgemm* for double precision).

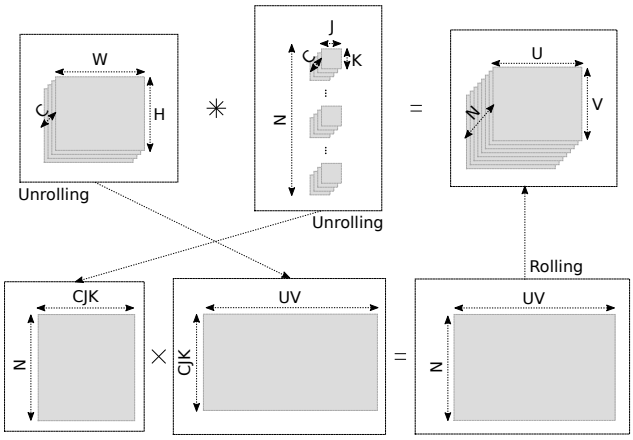


Figure 4.4: Unrolling of a convolution [186].

are assumed to be already defined as global variables (when possible) and according to the size described in the respective section. In Table 4.1 the size of the unroll of the convolution for the 38 convolutional layers of the network is shown. Some observations can be made:

1. OIs are large (in the order of tens of FLOP/B, with a maximum of 85.33 FLOP/B) except for convolutions with $J = K = 1$ for which $OI$ can go down to 0.06 FLOP/B (although still with an average of 4.26 FLOP/B and a maximum of 9.14 FLOP/B).

2. Even if OI is large and therefore the workloads can be assumed to be compute-bound, the absolute amount of memory traffic is very high (3 to 86 MiB per layer). These values require a dedicated design of the memory subsystem compared to processors for non compute-intensive workloads (which in the best case can leverage a 64 KiB L1 DC per core and a shared 2 MiB L2C), which will be carried out in Sec. 5.4.

3. The memory traffic is for a large majority composed by reads ($MT_{R\%}$ is 92.83% on average, with a minimum of 69.24%).

```
void sgemm() {
  for (size_t i = 0; i < n1; ++i)
    for (size_t j = 0; j < n3; ++j)
      for (size_t k = 0; k < n2; ++k)
        c[i*n3+j] += a[i*n2+k]*b[j+k*n3];}
```

Figure 4.5: C function employed for *igemm* on the scalar processor.

Assuming a square matrix at the output ($n_1 = n_3$) together with $\alpha = \beta = 1$, *sgemm* has

Table 4.1: Workload characterization for the 38 convolutional layer of CloudNet [299]. OI is in FLOP/B, MT in MiB.

| Layer | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $n_1$=N | 16 | 32 | 16 | 32 | 64 | 32 | 64 | 128 | 64 | 128 | 256 | 128 | 256 | 512 | 512 | 256 | 512 | 512 | 1024 |
| $n_2$=CJK | 36 | 144 | 144 | 32 | 288 | 288 | 64 | 576 | 576 | 128 | 1152 | 1152 | 256 | 2304 | 4608 | 2304 | 512 | 512 | 4608 |
| $n_3$=UV | 36864 | 36864 | 36864 | 36864 | 9216 | 9216 | 9216 | 2304 | 2304 | 2304 | 576 | 576 | 576 | 144 | 144 | 144 | 144 | 144 | 36 |
| J=K | 3 | 3 | 3 | 1 | 3 | 3 | 1 | 3 | 3 | 1 | 3 | 3 | 1 | 3 | 3 | 3 | 1 | 1 | 3 |
| MT | 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 | 3.66 | 11.81 | 11.81 | 18.77 |
| $MT_{W\%}$ | 30.76 | 18.17 | 10.00 | 9.99 | 18.08 | 9.97 | 9.94 | 17.39 | 9.76 | 9.52 | 13.33 | 8.33 | 7.14 | 4.65 | 2.38 | 3.85 | 2.38 | 2.38 | 0.75 |
| #MFLOP | 42.5 | 339.7 | 169.9 | 75.5 | 339.7 | 169.9 | 75.5 | 339.7 | 169.9 | 75.5 | 339.7 | 169.9 | 75.5 | 339.7 | 679.5 | 169.9 | 75.5 | 75.5 | 339.7 |
| OI | 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 | 44.31 | 6.10 | 6.10 | 17.26 |

| Layer | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 1T | 2T | 3T | 4T | 5T |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $n_1$=N | 1024 | 512 | 512 | 512 | 512 | 256 | 256 | 128 | 128 | 764 | 64 | 32 | 32 | 1 | 512 | 256 | 128 | 64 | 32 |
| $n_2$=CJK | 9216 | 512 | 9216 | 4608 | 4608 | 4608 | 2304 | 2304 | 1152 | 1152 | 576 | 576 | 288 | 32 | 4096 | 2048 | 1024 | 512 | 256 |
| $n_3$=UV | 36 | 36 | 144 | 144 | 144 | 576 | 576 | 2304 | 2304 | 9216 | 9216 | 36864 | 36864 | 36864 | 144 | 576 | 2304 | 9216 | 36864 |
| J=K | 3 | 1 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 1 | 2 | 2 | 2 | 2 | 2 |
| MT | 37.41 | 9.70 | 23.34 | 11.81 | 11.81 | 15.19 | 7.88 | 22.50 | 11.81 | 43.03 | 22.64 | 85.57 | 45.04 | 40.64 | 23.34 | 15.19 | 22.50 | 43.03 | 85.57 |
| $MT_{W\%}$ | 0.38 | 0.72 | 1.20 | 2.38 | 2.38 | 3.70 | 7.14 | 5.00 | 9.52 | 5.23 | 9.94 | 5.26 | 9.99 | 0.35 | 1.20 | 3.70 | 5.00 | 5.23 | 5.26 |
| #MFLOP | 679.5 | 18.9 | 1,359.0 | 679.5 | 679.5 | 1,359.0 | 679.5 | 1,359.0 | 679.5 | 1,359.0 | 679.5 | 1,359.0 | 679.5 | 2.4 | 1,359.0 | 1,359.0 | 1,359.0 | 1,359.0 | 1,359.0 |
| OI | 17.32 | 1.86 | 55.52 | 54.86 | 54.86 | 85.33 | 82.29 | 57.60 | 54.86 | 30.12 | 28.62 | 15.15 | 14.39 | 0.06 | 55.52 | 85.33 | 57.60 | 30.12 | 15.15 |



$$OI = \frac{n_1^2(1+2n_2)}{8(n_1^2+n_1\,n_2)}. \tag{4.8}$$

Furthermore, assuming that $2n_2 >> 1$ (as it is the case for every convolutional layer in CloudNet):

$$OI \approx \frac{n_1\,n_2}{4(n_1+n_2)}, \tag{4.9}$$

which given a certain memory traffic (i.e. $n_1 + n_2 = \text{const}$) is maximized for $n_1 = n_2$, reaching $OI \approx n_1/8$. 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 (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 [312]:

$$E_{sgemm} = \frac{FLOP/CC}{MTP_{CC}}. \tag{4.10}$$

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. 5.4.

### CONCATENATE LAYERS

Given that CloudNet is very deep (38 convolutional layers), it requires specific solutions in its architecture to mitigate the vanishing gradient problem [313]. The designers of CloudNet handled this problem using skip connections, and addition and concatenation layers [299]. As can be seen in Fig. 4.3, while the impact of addition layers on the

Table 4.2: Workload characterization for the 53 concatenation layers of CloudNet, where two 3D matrices of respectively size $n1\times n1\times n2$ and $n1\times n1\times n3$ are concatenated to create a $n1\times n1\times(n2+n3)$ 3D matrix [299]. *MT* is measured in MiB.

| Layer | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *n1* | 192 | 96 | 48 | 24 | 12 | 6 | 24 | 48 | 48 | 48 | 96 | 96 | 96 | 96 | 96 | 96 | 96 | 192 | 192 | 192 | 192 | 192 | 192 | 192 | 192 | 192 | 192 |
| *n2* | 16 | 32 | 64 | 128 | 256 | 512 | 256 | 128 | 256 | 384 | 64 | 128 | 192 | 256 | 320 | 384 | 448 | 32 | 64 | 96 | 128 | 160 | 192 | 224 | 256 | 288 | 320 |
| *n3* | 16 | 32 | 64 | 128 | 256 | 512 | 256 | 128 | 128 | 128 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 |
| *MT* | 9.0 | 4.5 | 2.3 | 1.1 | 0.6 | 0.3 | 2.3 | 4.5 | 6.8 | 9.0 | 9.0 | 13.5 | 18.0 | 22.5 | 27.0 | 31.5 | 36.0 | 27.0 | 27.0 | 36.0 | 45.0 | 54.0 | 63.0 | 72.0 | 81.0 | 90.0 | 99.0 |

| Layer | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *n1* | 192 | 192 | 192 | 192 | 192 | 12 | 48 | 96 | 96 | 96 | 192 | 192 | 192 | 192 | 192 | 192 | 192 | 24 | 96 | 192 | 192 | 192 | 48 | 192 | 96 | 192 |
| *n2* | 352 | 384 | 416 | 448 | 480 | 512 | 128 | 64 | 128 | 192 | 32 | 64 | 96 | 128 | 160 | 192 | 224 | 256 | 64 | 32 | 64 | 96 | 128 | 32 | 64 | 32 |
| *n3* | 32 | 32 | 32 | 32 | 32 | 512 | 128 | 64 | 64 | 64 | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 256 | 64 | 32 | 32 | 32 | 128 | 32 | 64 | 32 |
| *MT* | 108.0 | 117.0 | 126.0 | 135.0 | 144.0 | 1.1 | 4.5 | 9.0 | 13.5 | 18.0 | 18.0 | 27.0 | 36.0 | 45.0 | 54.0 | 63.0 | 72.0 | 2.3 | 9.0 | 18.0 | 27.0 | 36.0 | 4.5 | 18.0 | 9.0 | 18.0 |

execution time is negligible (1.1%), concatenation layers take a considerable part of the execution time (24.7%).

In a concatenation layer of CloudNet, two 3D matrices of respectively size $n_1\times n_1\times n_2$ and $n_1\times n_1\times n_3$ are concatenated to create a $n_1\times n_1\times(n_2+n_3)$ 3D matrix, i.e. the concatenation in Keras is executed on the third axis. Therefore, concatenate operations contain no FLOPs and consist mainly of memory transfers. For this reason, Table 4.2 does not contain *#MFLOPS* and *OI*. Furthermore, as the concatenation means basically copying the two matrices to different locations, $MT_W$ is always half the *MT*. Fig. 4.6 shows *conc*, a C function implementing concatenation, where the *memcpy* function from the standard library *string.h* is employed to copy the arrays into a new one.

```
#define A_SIZE n1*n1*n2
#define B_SIZE n1*n1*n3

void conc(){
  memcpy(c, a, A_SIZE*4);
  memcpy(&c[A_SIZE], b, B_SIZE*4);}
```

Figure 4.6: C function employed for *conc* on the scalar processor.

### SUBSAMPLING LAYERS

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 [314]). In CloudNet all the subsampling layers are implemented with max pooling, i.e. the maximum value of an $s\times s$ window is selected to compose the downsampled output matrix. This is common in state-of-the-art DNNs [186], even if also average pooling, mixed strategies and statistic pooling are employed in some cases. Fig. 4.7 reports the C function *maxpool2d* operating on an input matrix $A$ of size $n_1\times n_1\times n_2$ and generating an output matrix $B$ of size $(n_1/s)\times(n_1/s)\times n_2$. Table 4.3 shows the sizes and the subsampling factor $s$ of the subsampling layers in CloudNet.

### BATCH NORMALIZATION

In CloudNet batch normalization is applied to $n1\times n1\times n2$ matrices, where each batch is composed by a $n1\times n1$ 2D-matrix. Batch normalization layers are employed to speed up training and increase accuracy of DNNs [315]. This type of layer also acts as a regularizer,

```
void maxpool2d(){
for (size_t p = 0; p < n2; ++p) //rows
  for (size_t i = 0; i < n1; ++i)
    for (size_t j = 0; j < n1/s; ++j){
      b_t[j+i*n1/s+p*n1*(n1/s)] = a[s*j+i*n1+p*n1*n1];
      for (size_t k = 1; k < s; ++k)
        if (a[k+s*j+i*n1+p*n1*n1] > b_t[j+i*n1/s+p*n1*(n1/s)])
          b_t[j+i*n1/s+p*n1*(n1/s)] = a[k+s*j+i*n1+p*n1*n1];}

for (size_t p = 0; p < n2; ++p) //columns
  for (size_t i = 0; i < n1/s; ++i)
    for (size_t j = 0; j < n1/s; ++j){
      b[j+i*(n1/s)+p*(n1/s)*(n1/s)] = b_t[j+i*n1+p*(n1/s)*n1];
      for (size_t k = 1; k < s; ++k)
        if (b[j+i*(n1/s)+p*(n1/s)*(n1/s)] < b_t[j+k*n1/s+i*n1+p*(n1/s)*n1])
          b[j+i*(n1/s)+p*(n1/s)*(n1/s)] = b_t[j+k*n1/s+i*n1+p*(n1/s)*n1];}}
```

Figure 4.7: C function implementing *maxpool2d*.

Table 4.3: Workload characterization for the 15 subsampling layers (all of them are max pooling) [299].

| Layer | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $n1$ | 192 | 96 | 48 | 24 | 12 | 24 | 48 | 96 | 192 | 48 | 96 | 192 | 96 | 192 | 192 |
| $n2$ | 32 | 64 | 128 | 256 | 512 | 512 | 512 | 512 | 512 | 256 | 256 | 256 | 128 | 128 | 64 |
| $s$ | 2 | 2 | 2 | 2 | 2 | 2 | 4 | 8 | 16 | 2 | 4 | 8 | 2 | 4 | 2 |

keeping the magnitude of the coefficient low and thus avoiding overfitting [315, 316]. When executing batch normalization during inference, each element $a_{ijk}$ of the input 3D matrix $A$ from the previous layer is normalized according to [317]:

$$c_{ijk} = \gamma_k \cdot \left( \frac{a_{ijk} - E[A_k]}{\sqrt{Var[A_k] + \epsilon}} \right) + \beta_k \tag{4.11}$$

where $E[A_k]$ and $Var[A_k]$ are respectively the expected value and the variance calculated on each of the $n2$ 2D matrices of the input (obtained from cumulative statistics collected during training), $\epsilon$ is a small constant to ensure convergence, and $\gamma_k$ and $\beta_k$ are two scalars learned during training. To minimize the number of operations, Eq. 4.11 can be rewritten as:

$$c_{ijk} = \frac{\gamma}{\sqrt{Var[A_k] + \epsilon}} \cdot a_{ijk} - \frac{\gamma_k \cdot E[A_k]}{\sqrt{Var[A_k] + \epsilon}} + \beta_k. \tag{4.12}$$

In this way, $\gamma_k / \sqrt{Var[A_k] + \epsilon}$ and $-(\gamma_k \cdot E[A_k]) / \sqrt{Var[A_k] + \epsilon} + \beta_k$ can be precomputed after training and stored in memory as single parameters. For this reason, each layer has a number of parameters equal to $2 \cdot n3$. Fig. 4.8 shows *batchnorm*, a C function implementing a batch normalization layer, where the two parameters from Eq. 4.12 are respectively named *gamma_denom* and *mean_beta_denom*. The size, #FLOP and MT of each batch normalization layers is reported in Table 4.4. The number of operations required for $n_1 \times n_1$ matrix is $2 \cdot n$ and the amount of elements to be read from and written to memory is $n_1 + 4$. Therefore, the $OI$ is

$$OI = \frac{{n_1}^2}{2({2n_1}^2+1)}, \tag{4.13}$$

which, given that ${n_1}^2 >> 1$, in all cases gives around the same low value (0.25 B/FLOP). This value shows that this layer is typically memory bound, taking up a non-negligible part of the total execution time of CloudNet (5.5%).

```
void batchnorm(){
  for (size_t i = 0; i < n2; ++i)
    for (size_t j = 0; j < nl*nl; ++j)
      c[j+i*nl*nl] = (a[j+i*nl*nl]*gamma_denom[i])+mean_beta_denom[i];}
```

Figure 4.8: C functions employed for *batchnorm*.



Table 4.4: Workload characterization for the 31 batch normalization layers

| Layer | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *n1* | 192 | 192 | 192 | 96 | 96 | 96 | 48 | 48 | 48 | 24 | 24 | 24 | 12 | 12 | 12 | 12 |
| *n2* | 32 | 16 | 32 | 64 | 32 | 64 | 128 | 64 | 128 | 256 | 128 | 256 | 512 | 512 | 256 | 512 |
| #FLOP | 4.72 | 2.36 | 4.72 | 2.36 | 1.18 | 2.36 | 1.18 | 0.59 | 1.18 | 0.59 | 0.29 | 0.59 | 0.29 | 0.29 | 0.15 | 0.29 |
| *MT* | 9.00 | 4.50 | 9.00 | 4.50 | 2.25 | 4.50 | 2.25 | 1.13 | 2.25 | 1.13 | 0.56 | 1.13 | 0.57 | 0.57 | 0.29 | 0.57 |

| Layer | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *n1* | 12 | 6 | 6 | 6 | 12 | 12 | 12 | 24 | 24 | 48 | 48 | 96 | 96 | 192 | 192 |
| *n2* | 512 | 1024 | 512 | 1024 | 512 | 512 | 512 | 256 | 256 | 128 | 128 | 64 | 64 | 32 | 32 |
| #FLOP | 0.29 | 0.15 | 0.07 | 0.15 | 0.29 | 0.29 | 0.29 | 0.59 | 0.59 | 1.18 | 1.18 | 2.36 | 2.36 | 4.72 | 4.72 |
| *MT* | 0.57 | 0.30 | 0.15 | 0.30 | 0.57 | 0.57 | 0.57 | 1.13 | 1.13 | 2.25 | 2.25 | 4.50 | 4.50 | 9.00 | 9.00 |

### ACTIVATION LAYERS

Activation layers apply a non-linear function to each element of a 3D matrix of size $n_1 \times n_1 \times n_2$, producing an output of the same size. Analyzing Cloudnet in Keras, a total of 84 activation layers are found. However, only 48 are non-linear (the others being pass-through functions which do not have any computational impact), of which 47 are Rectified Linear Unit (ReLU) 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 [314]. Sigmoids (and hyperbolic tangents) are computationally more expensive, as in principle they require the calculation of a non-linear math function. A typical approach to implement them is to use a lookup-table [314] or a piece-wise linear approximation [302]. Table 4.5 reports the non-linear activation layers present in CloudNet and their size. In Fig. 4.9 a C function implementing a ReLu activation layer on a $n_1 \times n_1 \times n2$ 3D matrix.

### ADDITION LAYERS

In the addition layers of CloudNet, $n$ 3D matrices of dimensions $n_1 \times n_1 \times n_2$ are added. In this case, the number of operations is $\#FLOP = (n-1) \cdot {n_1}^2 \cdot n_2$, the memory traffic is $MT = 4 \cdot (n+1) \cdot ({n_1}^2 \cdot n_2)$, and $MT_{W\%} = 1/(n+1)$. Fig. 4.10 shows the *madd2* and *madd3* C functions to perform respectively addition of 2 and 3 matrices. The C functions *madd4* and *madd5* are not reported for brevity, but they are composed using madd2 as shown in madd3.

Table 4.5: Workload characterization for the 48 activation layers of CloudNet. All of them are *relu*, while the last one is *sigmoid*. $MT_{W\%}$ is around 50% for all layers.

| Layer | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $n1$ | 192 | 192 | 192 | 192 | 192 | 96 | 96 | 96 | 96 | 48 | 48 | 48 | 48 | 24 | 24 | 24 | 24 | 12 | 12 | 12 | 12 | 12 | 12 | 6 |
| $n2$ | 16 | 32 | 16 | 32 | 32 | 64 | 32 | 64 | 64 | 128 | 64 | 128 | 128 | 256 | 128 | 256 | 256 | 512 | 512 | 256 | 512 | 512 | 512 | 1024 |
| MT | 4.50 | 9.00 | 4.50 | 9.00 | 9.00 | 4.50 | 2.25 | 4.50 | 4.50 | 2.25 | 1.13 | 2.25 | 2.25 | 1.13 | 0.56 | 1.13 | 1.13 | 0.56 | 0.56 | 0.28 | 0.56 | 0.56 | 0.56 | 0.28 |

| Layer | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $n1$ | 6 | 6 | 6 | 12 | 12 | 12 | 12 | 12 | 24 | 24 | 24 | 24 | 48 | 48 | 48 | 48 | 96 | 96 | 96 | 96 | 192 | 192 | 192 | 192 |
| $n2$ | 512 | 1024 | 1024 | 512 | 512 | 512 | 512 | 512 | 256 | 256 | 256 | 256 | 128 | 128 | 128 | 128 | 64 | 64 | 64 | 64 | 32 | 32 | 32 | 1 |
| MT | 0.14 | 0.28 | 0.28 | 0.56 | 0.56 | 0.56 | 0.56 | 0.56 | 1.13 | 1.13 | 1.13 | 1.13 | 2.25 | 2.25 | 2.25 | 2.25 | 4.50 | 4.50 | 4.50 | 4.50 | 9.00 | 9.00 | 9.00 | 0.28 |

```
void relu(){
 for (size_t k = 0; k < n2; ++k)
   for (size_t j = 0; j < n1; ++j)
     for (size_t i = 0; i < n1; ++i)
       if (a[k*n1*n1+j*n1+i] < 0)
         b[k*n1*n1+j*n1+i] = 0;
       else
         b[k*n1*n1+j*n1+i] = a[k*n1*n1+j*n1+i];}
```

Figure 4.9: C function implementing *relu*.

```
void madd2(float *a, float *b, float *c){
  for (size_t i = 0; i < n1*n2; ++i)
    for (size_t j = 0; j < n1; ++j)
        c[i*n1+j] = a[i*n1+j] + b[i*n1+j];}

void madd3(float *a, float *b, float *c, float *d){
  madd2(a, b, d);
  madd2(c, d, d);}
```

Figure 4.10: C functions employed for *madd2*, and *madd3*. In this case local variables are used to allow the composition of functions with more than two operands.

Table 4.6: Workload characterization for the 15 addition layers of CloudNet, each composed an addition of $m$ 3D matrices, each of size $n1 \times n1 \times n2$ [299].

| Layer | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| m | 2 | 2 | 2 | 2 | 3 | 2 | 5 | 3 | 4 | 3 | 3 | 3 | 2 | 3 | 3 |
| n1 | 192 | 96 | 48 | 24 | 12 | 6 | 12 | 12 | 24 | 24 | 48 | 48 | 96 | 96 | 192 |
| n2 | 32 | 64 | 128 | 256 | 512 | 1024 | 512 | 512 | 256 | 256 | 128 | 128 | 64 | 64 | 32 |
| MT | 1.38 | 0.69 | 0.35 | 0.17 | 0.12 | 0.04 | 0.17 | 0.12 | 0.29 | 0.23 | 0.46 | 0.46 | 0.69 | 0.92 | 1.84 |
| $MT_{W\%}$ | 33.33 | 33.33 | 33.33 | 33.33 | 25.00 | 33.33 | 16.67 | 25.00 | 20.00 | 25.00 | 25.00 | 25.00 | 33.33 | 25.00 | 25.00 |
| #FLOP | 1.18 | 0.59 | 0.29 | 0.15 | 0.15 | 0.04 | 0.29 | 0.15 | 0.44 | 0.29 | 0.59 | 0.59 | 0.59 | 1.18 | 2.36 |
| OI | 1.23 | 1.23 | 1.23 | 1.23 | 0.82 | 1.23 | 0.61 | 0.82 | 0.68 | 0.82 | 0.82 | 0.82 | 1.23 | 0.82 | 0.82 |

### 4.4.2. 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, using CNNs. In these cases, the convolutional layers increase in $N$ and decrease in $UV$ going to successive layers (as in CloudNet from the convolutional layer 1 to 19/20 in Table 4.1), but then instead of being followed by a final cascade of convolutional layers where there is an

opposite trend (until the output has the same 2D size of the input), the DNN is terminated by adding Fully Connected (FC) layers (to make a decision based on the information contained in groups of pixels). FC layers can be seen as convolutional layers where there is no sharing of coefficients, i.e. where (using the same notation as in Sec. 4.4.1) $J = K = W = H$ [186]. This implies that the output is a vector of size $N$, the number of operations is $\#FLOPs = 2NCHW$, the memory traffic is $MT = 4[CHW(N+1)+N]$ and

$$OI = \frac{1}{2[1+1/N+1/(CHW)]}. \tag{4.14}$$

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, the memory traffic and OI for the convolutional layers in Table 4.1 were compared to FC layers with same $C$, $W$, $H$ and $N$. The $MT$ of the FC layers ranges between 1.31x and 18.36x compared to the respective convolutional layer, while the OI is 3.3x to 154.2x smaller. The high MT associated with FC layers is confirmed by [186], although a trend can be noticed: for early CNNs with few convolutional layers (e.g. AlexNet with 5 convolutional layers and 3 FC layers) the percentage of parameters in the FC layers is very high (for AlexNet 96.07%), while state-of-the-art deeper CNNs (typically achieving higher accuracy) like ResNet [313] 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 [186]. 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[15]. When processing $B$ input features in parallel, the number of operations is $\#FLOP = 2NBCHW$, the memory traffic is $MT = 4[CHW(N+B)+BN]$ and the operational intensity is

$$OI = \frac{1}{2[1/B+1/N+1/(CHW)]}. \tag{4.15}$$

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

$$OI_{max} = \frac{1}{2[1/N+1/(CHW)]} \tag{4.16}$$

is reached (in this example around 254 $FLOP/B$). A relatively high value of $B$ may be required to achieve an $OI$ in the order of the tens (e.g. 15.05 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

[15] Batching is instead not effective with convolutions, as the amount of parameters in a convolution is very small (e.g. 3x3x16).

in real-time applications and is further analyzed in Sec. 4.4.3. Despite the described criticalities of FC layers, they typically have limited impact on the execution time of CNNs. For instance, in [318] 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.

### 4.4.3. RECURRENT NEURAL NETWORKS

Recurrent Neural Networks (RNNs) are typically employed in time series analysis like speech recognition and Natural Language Processing (NLP) [319–321] 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 [302]. 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 high accuracy, while Gated Recurrent Unit (GRU) implementations provide lower accuracy with higher performance [302]. Furthermore, one or more FC layers are placed before the output [302].

#### LSTM LAYERS

LSTM layers are typically memory-bounded [322]. Similarly to [302], the linear part of the LSTM layer can be described as:

$$s_t = W \cdot x_t + U \cdot h_{t-1} + b \tag{4.17}$$

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 $2(n^2 + n + n \cdot m)$, the $MT$ seen by main memory is $4(n^2 + nm + 3n + m)$ and the $OI$ is:

$$OI = \frac{\#FLOP}{2(\#FLOP + 2n + m)} \tag{4.18}$$

with a maximum value of 0.5 FLOP/B for large matrices, i.e. $\#FLOP >> 2n + m$. 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:

$$OI = \frac{\#FLOP \cdot B}{2[\#FLOP + (3B - 1)n + m \cdot B]} \tag{4.19}$$

This equation shows that the efficacy of batching in terms of increase of $OI$ saturates as $B$ grows, until the upper bound of

$$OI_{max} = \frac{\#FLOP}{6n + 2m} \tag{4.20}$$

is achieved. This upper bound is a relatively large value, for instance being 27.29 for $m = 27$ and $n = 60$ (typical values of $m$ and $n$ in [302]). 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 [322] increasing batching from 16 to 64 increases performance to 2.41× the original value, while the time required to complete execution in more than 99% of the cases increases from 7.2 ms to 21.3 ms (2.95x).

### 4.4.4. UNSUPERVISED LEARNING

As a case study of unsupervised learning algorithm for anomaly detection, DBSCAN [323] will be investigated. Each point is classified according to how many points are in its neighborhood. This means that this algorithm is a composed by a loop, calculating the distance between points. Therefore, its core operation is

$$(x_i - x_j)^2 + (y_i - y_j)^2 < \epsilon^2. \tag{4.21}$$



The OI of this type of operation is low ($OI = 1/5$), as 5 elements ($x_i$,$x_j$,$y_i$,$y_j$ and $\epsilon^2$) are read from memory and 4 operations are executed (two subtractions and two multiplications).

## 4.5. SUMMARY

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 which 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 ground 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 data). Therefore, there is a need for large, public and standardized datasets to be used as reference data set to benchmark DNN architectures to be deployed in space applications. However, part of future LEO satellites are planned to be part of large constellations, making large datasets more easily available in the future.

The analysis of the workloads associated with CloudNet, a DNN for cloud detection, shows that most parts are very compute-intensive and can be mapped to matrix-matrix multiplications with high OI (*sgemm*). Although MT is typically high also for this type of kernels, it is mainly composed of read transactions. These characteristics of the workload of DNNs will be exploited in the next chapter to define an appropriate hardware platform.

# 5

# Analysis of Potential and Challenges of Vector Processors

*Se un'idea è più moderna di un'altra,*
*è segno che non sono immortali né l'una né l'altra.*
*When an idea is more modern than another one,*
*neither of them is immortal.*

Carlo Emilio Gadda, La cognizione del dolore (1941)

*In this chapter, the RISC-V Vector Extension (RVVE) is introduced. Then, an analysis of the potential and challenges of the microarchitectures of vector processors is carried out. The preliminary design of a RISC-V vector processor to be employed as a generic platform to enable energy-efficient OBDM both for payload and platform applications is investigated. The design of the memory subsystem is explored in depth to allow full exploitation of the computational resources in typically resource-constrained space systems. Then, the kernels of CloudNet introduced in Chapter 4, are implemented using the RVVE, proving that large gains in terms of performance are possible (in particular a sizeable reduction of the number of instructions) when scalar kernel implementations are replaced with vector implementations.*

Parts of this chapter have been published in [288].

## 5.1. INTRODUCTION

It is still matter of discussion whether it will be feasible to deploy AI systematically on-board satellites in the next 10-15 years. To meet the requirements to execute DNNs in reasonable time scales, already identified in this dissertation by analyzing CloudNet in Chapter 4, the space industry is following three main approaches:

1. Work is being done to efficiently map DNNs on resource-constrained state-of-the-art space processors [324], accepting a consistent loss of performance compared to DNNs in high-performance processors for terrestrial applications. This approach can exploit synergies with the trend in IoT of implementing DNNs in low-power and resource constrained processors [325].

2. High-performance proprietary COTS processors employed in terrestrial applications are being proposed [326]. Although they can achieve higher-order magnitude performance compared to state-of-the-art space processors [8], they come with a large 'cost of ownership' to avoid losses in terms of dependability [117], and possible restrictions on their usage and knowledge of internal behavior (see Chapter 1).

3. FPGAs allow the design of a customized hardware accelerator, typically connected to either a hard or soft processor through an interconnect [8]. The accelerator can be either handcrafted in HDLs or autogenerated from software, after profiling to identify the most computational intensive functions. In [327] it is shown that Vivado HLS with enabled optimizations (i.e. pipelining and concurrent execution of operations) achieves a 6.23x speedup for a small CNN and 9x for a larger CNN on a Zynq compared to the software implementation on its hard processor (which can be considered roughly equivalent to space processors). The custom accelerator approach in space application is typically limited to FPGAs (and not financially viable for ASICs), given the niche-sized market available. Furthermore, in [328] 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%).

In this dissertation, a novel approach is followed. Instead of employing COTS components or using automated tool to generate hardware for custom FPGA accelerators, the idea is to improve the performance of space processors with DLP, to achieve a level of performance per CC of the same order of magnitude of terrestrial processors for DNNs. This processor is meant to be used in the future in rad-hard and/or rad-tol ASICs. This approach will systematically enable OBDM in space applications, as opposed to the three approaches described previously, which are viable only in specific contexts. For instance, a COTS component successfully used to enable OBDM in a previous mission may be discontinued or replaced with new components with worse radiation tolerance. Even smaller changes, like changing process, fab or even lot-to-lot variance can hinder the use of a similar solution in new missions.

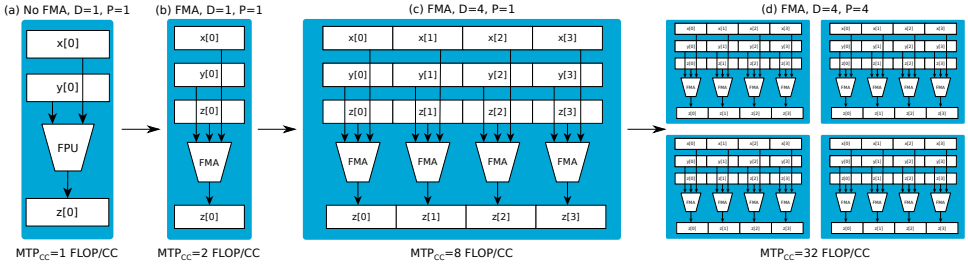


Figure 5.1: Steps to increase the $MTP_{CC}$ of space processors in a power- and area-efficient way, with $D$ number of elements of a vector on which a single instruction can operate and $P$ number of processing cores.

### 5.1.1. PROPOSED APPROACH

State-of-the-art processors for space applications typically execute instructions on two scalar operands [30]. Considering a single core, this type of platform has a $MTP_{CC}$ 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[1], achieving a $MTP_{CC}$ of 2 FLOP/CC (as shown in Fig. 5.1). This requires modifications to the FPU and 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 [329].

To increase the $MTP_{CC}$ even further, DLP is the most energy-efficient solution available [330]. 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 [59] shows that the instruction cache dissipates 19.63% of the total energy, the 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. Fig. 5.1 (c) 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 [330], 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. In [331], the speedup found in the convolutional layers of a CNN using the data-parallel NEON extension over the baseline ARM ranges from 2.45x to 2.78x, 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 5.98x to 15.50x 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. 5.1 (d) the core is replicated 4 times ($P = 4$), achieving an $MTP_{CC}$ of 32 FLOP/CC (together with FMA and $D = 4$). Going

[1] Both will be indicated with FMA, unless a distinction is to be done.

above four cores typically reduces the utilization of the functional units. For instance, in [189] it is shown that with 8 cores it is possible to obtain for CNNs performances ranging from 3.99x to 5.76x the performance of a single core. Similarly, with 8 cores it is possible to reach 5.55x the performance of a single-core implementation of an LSTM RNN [302]. The next section details the state-of-the-art of DLP ISAs.

### 5.1.2. DATA-PARALLEL INSTRUCTION SET ARCHITECTURES

When compute-intensive applications were to be addressed in the commercial market, computer architects resorted to packed SIMD ISAs both for Personal Computers (with the Intel's MMX extensions (1996) for integers [332] and the SSE extensions (1999) for floating point elements [333]) and for embedded applications (with the NEON extension [138]). The success of ARM in high-end embedded applications made the SIMD NEON extension, first introduced in the ARMv7-A Cortex-A8 (2005) [138], 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 [137].

Packed-SIMD extensions are typically chosen by hardware designers because they can be applied to scalar processors without extensive modifications to the microarchitecture [18]. 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) [20]. 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% [334] and it is composed by 48 computing cores, each achieving around 57 DP-GFLOPS [334].

Vector extensions can be seen as more flexible versions of packed-SIMD extensions thanks to their time-multiplexed and VLA approach. For VLA ISAs, the software is not required to know the hardware vector length of a specific implementation and the code can be written to run the same executable with the largest parallelism available on every platform without any modification [18–20]. In SIMD extensions, the data width of the operations is encoded directly in the instruction opcodes instead. Therefore, the code must target a specific width. Furthermore, when the architects of SIMD ISAs want to increase performance by widening the vectors, they must add a new set of instructions to process these vectors [18]. Therefore, application code compiled for previous versions of the ISA cannot automatically leverage the widened vectors of new implementations and the code compiled for wider SIMD extensions fails to execute on older machines (as the new instructions are not known to older implementations) [18].

For these reasons, the proposal for packed-SIMD floating-point was dropped in favor of the V extension for large floating-point vector operations [121]. However, there is 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 [121].

### 5.1.3. OUTLINE

In Sec. 5.2, the RVVE is presented. This RISC-V extension allows the introduction of DLP in space processors, without the shortcoming of SIMD ISAs. Then, the microarchitec-

ture of vector processor is analyzed and the information collected is used to define a suitable hardware platform for space applications. To account both for computational and memory constraints, separate discussions are carried out for the microarchitecture of the processing core and its memory subsystem, respectively in Sec. 5.3 and Sec. 5.4. Finally, the C functions required to implement CloudNet on scalar processors (defined in Sec. 4.4.1) are rewritten with the RVVE instructions in Sec. 5.5 to exploit DLP in vector processors, analyzing the performance increase due to the vector instructions and verifying the functional equivalence between scalar and vector functions.

## 5.2. THE RISC-V VECTOR EXTENSION

The RISC-V Vector Extension (RVVE) is similar to the ARMv8-A SVE and was heavily inspired by the Hwacha[2] development [335]. 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) [121] and allow the same binary code to work efficiently across a variety of hardware implementations, which vary in physical vector storage capacity and datapath parallelism. Additionally, ARMv8-A SVE includes 16 scalable predicate registers (not defined in the baseline RVVE [336]) to optimize loops, using the predicate controlled loops vectorization style [20].

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 [136], and Ara, a single-core RISC-V vector processor based on Ariane achieving up to 33 GFLOP/s and 41 GFLOP/J, with clock frequency higher than 1 GHz on 22 nm FD-SOI technology. Furthermore, work is being done to support the RVVE in popular DNN frameworks like TensorFlow Lite [337].

The rest of this section introduces the most important additions of the RVVE to the base RISC-V ISA. However, this section is not intended to give an exhaustive description of the RVVE, for which Draft 0.9 specifications [336] are available[3].

### 5.2.1. VECTOR REGISTERS

The vector extension adds 32 architectural vector registers (*v0-v31*) to the base scalar RISC-V ISA. Each implementation defines a number of bits in a vector register, *VLEN*, and the maximum size of a vector element that any operation can produce or consume in bits, *ELEN*. Both of them must be power of 2. These two fixed values for each RVVE implementation define also the maximum number of elements in a vector register, i.e. $VLEN = VLMAX \cdot ELEN$. In most implementations *VLMAX* is also the number of lanes $N_{Lanes}$, each of them operating on a element of size *ELEN* bits of a vector.

[2] 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 [18]. This can potentially lead to higher performance, but also higher complexity.

[3] They have not been ratified yet and non backward-compatible changes will potentially happen [336].

### 5.2.2. CONFIGURATION AND STATUS REGISTERS

The RVVE adds several Control and Status Registers (CSRs) [336]. Among them:

- *vtype*: it configures the type of elements of the vector. For instance, the field *vsew* configures the Selected Element Width (SEW), i.e. the bit size of each element of the vector (e.g. 8-bit, 16-bit, 32-bit etc.). When a certain SEW is set, also the number of elements in a vector register is set: if a vector register is composed of 128 bits (VLEN=128), when configured with 8 bit vectors it stores 16 elements, while when configured for 16-bit vectors it stores 8 elements. To deal with vectors longer than VLEN, multiple vector registers can be grouped together to form a vector register group and a single vector instruction can read a single operand from multiple vector registers. For instance, when the field *lmul* is set to 2, vector registers are organized to store 16 vectors of $2 \cdot VLEN/SEW$ elements and the instruction *vadd v2, v4, v6* will execute *(v2, v3) := (v4,v5) + (v6,v7)*, where *(vx, vy)* is the concatenation of *vx* and *vy*.

- *vl*: it contains an unsigned integer specifying the number of elements to be updated by a vector instruction. Elements in any destination vector register group with indices greater or equal than *vl* are not modified during execution of a vector instruction.



### 5.2.3. OPERATIONS

The registers *vtype* and *vl* must be configured before using the vector unit, as shown below (the final i stands for "immediate"):

```
vsetvli rd, rs1, vtypei, lmul # rd = new vl, rs1 = AVL, vtypei = new vtype
                              # if rs1 = x0, then use current vector length
```

where AVL is the Application Vector Length, i.e. the number of elements in the vector defined at software-level. The AVL encoded in the instruction (given that it is within certain boundaries reported in [336]) will become the new *vl*). If AVL is set to zero, the *vl* is set to VMAX. Furthermore, the new *vl* will be written in the scalar register *rd*.

After the configuration, vector arithmetic instructions on values held in vector register elements can be executed. Therefore loads (and stores) to move bit patterns between vector register elements and memory are required. The RVVE supports unit-stride (accessing elements stored contiguously in memory starting from the base effective address), strided (accessing the first memory element at the base effective address, and then accessing subsequent elements at address increments given by the byte offset contained in the GPR register specified by *rs2*), and indexed (using a vector of offsets and adding the contents of each element of the vector offset operand specified by vs2 to the base effective address to give the effective address of each element) addressing modes. Vector load/store base registers and strides are taken from a scalar GPR register. As opposed to other instructions, the width of each element of the vector is specified by the load and store instructions, for instance:

```
# vd destination, rs1 base address
vle16.v vd, (rs1), vm  # 16-bit unit-stride load
vle32.v vd, (rs1), vm  # 32-bit unit-stride load

# vs3 store data, rs1 base address
vse16.v  vs3, (rs1), vm # 16-bit unit-stride store
```

```
vse32.v  vs3, (rs1), vm # 32-bit unit-stride store

# vd destination, rs1 base address, rs2 byte stride
vlse16.v vd, (rs1), rs2, vm # 16-bit strided load
vlse32.v vd, (rs1), rs2, vm # 32-bit strided load

# vs3 store data, rs1 base address, rs2 byte stride
vsse16.v vs3, (rs1), rs2, vm # 16-bit strided store
vsse32.v vs3, (rs1), rs2, vm # 32-bit strided store
```

In the instructions above, (rs1) indicate the address contained in the scalar register rs1. As most RVVE instructions, vector load and store instructions can be masked with a *vm* field:

```
vop.vv    v1, v2, v3, v0.t  # enabled for element i when v0[i].LSB=1, m=0
vop.vv    v1, v2, v3        # unmasked vector operation, m=1
```

As an example of possible operations after the data is loaded, some Vector Integer Arithmetic Instructions are provided below:

- Vector Single-Width Integer Add and (reverse) Subtract. Among them, there are vector addition instructions between vectors, vector and scalar, and vector and immediate value:

  ```
  vadd.vv vd, vs2, vs1, vm   #Vector-vector: vd[i] = vs1[i] + vs2[i]
  vadd.vx vd, vs2, rs1, vm   #vector-scalar: vd[i] = x[rs1] + vs2[i]
  vadd.vi vd, vs2, imm, vm   #vector-immediate: vd[i] = imm + vs2[i]
  ```

  In the instructions above, *vd[i]* represents the $i^{th}$ element of the vector vd (same for other vector registers), while *x[rs1]* represents the scalar values contained in the scalar registers rs1.

- Vector Single-Width Integer Multiply-Add Instructions (i.e. FMA operations). Among them, Integer multiply-add instructions overwriting the addend or the multiplicand:

  ```
  # Integer multiply-add, overwrite addend
  vmacc.vv vd, vs1, vs2, vm    # vd[i] = +(vs1[i] * vs2[i]) + vd[i]
  vmacc.vx vd, rs1, vs2, vm    # vd[i] = +(x[rs1] * vs2[i]) + vd[i]

  # Integer multiply-add, overwrite multiplicand
  vmadd.vv vd, vs1, vs2, vm    # vd[i] = (vs1[i] * vd[i]) + vs2[i]
  vmadd.vx vd, rs1, vs2, vm    # vd[i] = (x[rs1] * vd[i]) + vs2[i]
  ```

- Other examples are: Vector Widening Integer Operations (where the destination element width is doubled), Vector Bitwise Logical Instructions, Vector Integer Min/Max Instructions (unsigned and signed), Vector Integer Add-with-Carry/Subtract-with-Borrow Instructions (to support multi-word arithmetic), etc.

Instructions similar to those described for integers are given for Fixed Point and Floating Point operations. For instance, for Floating Point additions:

```
vfadd.vv vd, vs2, vs1, vm   # Vector-vector
vfadd.vf vd, vs2, rs1, vm   # vector-scalar
```

The standard defines also more specific instructions, for instance:

- Vector Reduction Operations: they take a vector register group of elements and a scalar held in element 0 of a vector register, and perform a reduction using some binary operator, to produce a scalar result in element 0 of a vector register. For instance:

```
vredmax.vs  vd, vs2, vs1, vm   # vd[0] = max(vs1[0], vs2[*])
```

  In the instructions above the asterisk indicates that the operation is executed on all the elements of *vs2*.

- Vector Mask Instructions. For instance:

```
vfirst.m rd, vs2, vm
```

  In this case, the instruction finds the lowest-numbered active element of the source mask vector that has its Least Significant Bit (LSB) set and writes that element's index to a GPR. If no element has an LSB set, -1 is written to the GPR.

- Vector Permutation Instructions. For instance slide instructions. For instance:

```
vslideup.vx vd, vs2, rs1, vm        # vd[i+rs1] = vs2[i]
```



### 5.2.4. EXCEPTIONS

When a trap occurs during a vector instruction (caused by either a synchronous exception or an asynchronous interrupt), a CSR is written with a pointer to the errant vector instruction, while the RVVE-specific *vstart* CSR is written with the element index that caused the trap to be taken. In this way, it is possible to resume partially executed vector instructions to reduce interrupt latency and to simplify forward-progress guarantees. This is similar to the scheme in the IBM 3090 vector facility [336]. To ensure forward progress without the *vstart* CSR, implementations would have to guarantee that an entire vector instruction can always complete atomically without generating a trap.

Some platforms may choose to provide a privileged mode bit to select between precise and imprecise vector traps. Imprecise mode would run at high-performance but possibly make it difficult to discern error causes, while precise mode would run more slowly, but support debugging of errors [336].

## 5.3. MICROARCHITECTURE OF VECTOR PROCESSORS

There are two main approaches to design a vector processor:

1. Many supercomputers like the Fujitsu AF64X coprocessor have a joint scalar and vector pipeline with separated register files and execution units [334]. 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. as done in the Fujitsu AF64X with up to 4-ways).

2. 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 [19] and it is shown in Fig. 5.2. Another advantage of this approach is that it provides a more modular solution and a vector version of a scalar RISC-V processor can be achieved with minimal modifications to the scalar pipeline (i.e. the introduction of a front end).

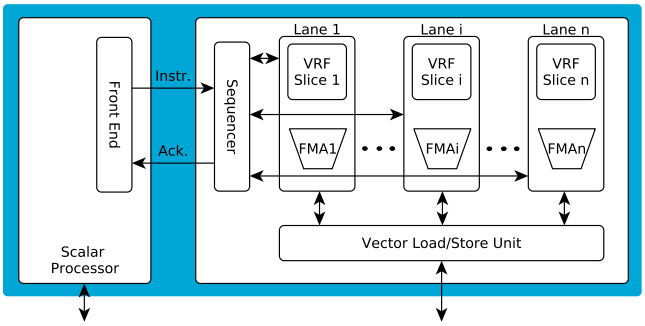


Figure 5.2: Block diagram of a decoupled vector pipeline.

The critical elements of a vector processor are shown in Fig. 5.2. The following subsections will focus on the Vector Register File (Sec. 5.3.1), and on the issues limiting scalability of performance (Sec. 5.3.2). Furthermore, Sec. 5.3.3 provides insights on the soft error vulnerability of vector processors.

### 5.3.1. VECTOR REGISTER FILE

Vector Register Files (VRFs) are typically more complex than scalar RFs, as they have in general more contention given FMA operations and masked execution [19]. 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) [19], executes the operation only if the mask has a certain value and writes the result of the operation to a register. A straightforward solution to avoid contention in the VRF is therefore to employ a multiported 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 [330], the area of the VRF for the T0 vector processor according to different the 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 two 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 with the area:

- One 5R3W bank of 256 elements (1×5R3W)
- Two 3R2W banks of 128 elements each (2×3R2W)
- Four 2R1W banks of 64 elements each (4×2R1W).

Data from [330] 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) [19]. This is particular effective to avoid conflicts when the functional units fetch the first elements of two vectors. [19]. However, this organization leaves some residual contention, which is addressed with round-robin arbitration [19]. 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 [335].



### 5.3.2. SCALABILITY

Although existing RISC-V vector processors have good scalability in terms of peak performance and efficiency (as can be seen in Table 5.1), 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.

Table 5.1: Scalability of Ara in terms of number of lanes (peak values in bold) for 22FDX process (FD-SOI). Data derived from [19].

| Number of lanes | 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** |

#### FREQUENCY

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

In [19] Ara has been implemented in the Global Foundries 22FDX process (FD-SOI). As can be seen in Table 5.1, the 2-lane and 4-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 implies 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 [19].

It should be noted that the maximum frequency of the scalar processor on the same technology is 1.7 GHz [59]. Therefore, the 2-lane version already comes with a penalty of at least 30% percent compared to the scalar processor.

#### AREA AND ENERGY EFFICIENCY

The increasing energy efficiency in Tab. 5.1 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 mW (2 lanes) and 794 mW (16 lanes) at peak performance [19]. 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 10735 kGE. In particular, the area of Ariane and Ara with 4 lanes is 3434 kGE. i.e. 4.28x a single-core Ariane comprising L1 caches. Therefore a four-lane vector processor has similar requirements in terms of die area compared to state-of-the-art quad-core processor for space [30]. This implies that, for space applications, the use of FPUs in vector processors typically limits the number of lanes to four. For this reason, in Sec. 5.5 the functions are written for integers and in the next chapter the design of the vector processor will be carried out only for integer instructions.

Employing fixed point values (substantially integers with scaling factors of $2^s$) instead of floating-point values is a well-known technique when implementing DNNs in embedded systems. As a matter of fact, hardware implementation of the FPGAs cannot afford floating point operations and typically employ fixed point values. In [338] it is shown that precision of integers can be brought down to 8-4 bits, giving up only around 0.27% to 1.01% of the accuracy.

#### SMALL MATRICES

Along with the memory bound identified by the roofline model, the authors of Ara [19] 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 [19], for a $n \times n$ *igemm* with 32-bit elements, an upper bound due to the issue rate can be found. As the lanes operate in lockstep when enabled, this upper

bound in terms of number of FLOP/CC can be written as the utilization of the functional units in the processor multiplied by the $MTP_{CC}$. The former can be calculated as the number of CCs each functional unit is busy with an FMA instruction, divided by the the minimum number of clock cycles required to issue two instructions executing operations:

$$\#FLOP/CC \leq MTP_{CC} \cdot \frac{2n/MTP_{CC}}{\Delta CC_{issue}}. \tag{5.1}$$

Considering the OI of the *igemm* kernel with 32-bit elements ($OI = 8/n$), $n$ can be replaced:

$$\#FLOP/CC \leq \frac{16 \cdot OI}{\Delta CC_{issue}}. \tag{5.2}$$

Therefore, $OI^*$ (in this case representing the boundary between issue-bound and compute-bound kernels) is:

$$OI^* = \frac{MTP_{CC} \cdot \Delta CC_{issue}}{16}. \tag{5.3}$$

Eq. 5.3 shows that doubling the issue rate (i.e. using a dual-issue microarchitecture) will halve the $\Delta CC_{issue}$, thus halving $OI^*$. For instance, as a FMA instruction can be issued every five CCs in Ara, the worst $OI^*$ is 5 FLOP/B (considering an 8-lane version with $MTP_{CC}$ = 16 FLOP/CC), while a dual-issue version lowers this value to 2.5 FLOP/B. As will be seen in Sec. 5.4, these values are comparable with upper bounds due to memory bandwidth and therefore can have an impact on performance when they produce a higher $OI^*$ than memory bandwidth.

#### INTERCONNECT

In order to increase the $OI^*$ due to the memory bandwidth, Ara uses a single $32 \cdot N_L$ bit wide bus interface for all the lanes together[4], reaching 512 bits for 16 lanes. To keep the same ratio between peak operations per CC and peak memory transfer (in this case 0.5 DP-FLOP/B), a 32-lane implementation would need a 1024 bit wide bus interface. However, this problem can be mitigated using a L1 cache for vector data, i.e. Vector Cache (VC), which allows large bandwidth for data residing in it without requiring a wide crossbar (Fig. 5.3). The design of an area efficient memory subsystem for RISC-V vector processors is described in Sec. 5.4.

### 5.3.3. DEPENDABILITY

Vector processors typically achieve high utilization of the FPU (e.g. 97% in [19]), while 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 [78] relating utilization and soft error vulnerability. Furthermore, the increase of frequency compared to state-of-the-art processors

[4] Hwacha, instead, uses an interface per lane [191].

for space (e.g. from 250 MHz to 1 GHz) points to an increased percentage of errors from combinational logic (as shown in [47]), which compose the majority of the area in FPUs and ALUs . For instance, the BOOM processor[5] was synthesized on a 65nm ASIC technology and the area of the FPU and ALUs (comprising hardware multiplication and division) were found to be composed respectively for 79.52% and 86.11% of combinational logic. Finally, scaling efficiently at least up to 16 lanes, vector processor can achieve high performance when large ASIC implementations are possible. For this reason, small technology nodes should be preferred. However, in [51] it is reported that going below 28 nm increases the SER in the terrestrial environment. In FD-SOI technologies this is mainly due to an increase of SER due to protons, while 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 4-lane version is in the vector pipeline and allows for a maximum frequency of around 1 GHz, while the scalar pipeline has a critical path allowing up to 1.7 GHz [59]. Therefore, applying state-of-the-art techniques to improve fault-tolerance only to the scalar pipeline (e.g. TMR at flip-flop level in the scalar pipeline and EDAC codes in the scalar register file) will not cause any penalties in terms of maximum frequency. In this way, there will be no penalties in terms of $MTP_{CC}$ compared to the non-FT version. As a matter of fact, TMR and EDAC are reported to cause only 9% decrease in frequency in the LEON2 [81]. A similar decrease would keep the maximum frequency of Ariane from 1.7 GHz [59] to around 1.5 GHz, which is still above the maximum frequency possible in the vector pipeline.



## 5.4. MEMORY HIERARCHY

Fig. 5.3 shows a possible memory hierarchy for a vector processor. As a typical memory hierarchy for scalar processors, it comprises a DC, an IC, a unified L2C[6], and a main memory. However, an VC is added to increase performance especially for workloads with low $OI$. The figure also indicates the width $W_i$ of the interface between levels (where $i$ is $D$ for the interface of a DRAM module channel and the memory controller, *L2,MCon* for the one between memory controller and L2C, $x$ for the interface between interconnect and VC, *C,V* for the interface between the vector pipeline and the VC), which determines the bandwidth $B_i$ of the interface together with its clock frequency $f_{clk_i}$ (while for the DRAM and interconnect the same nomenclature as $W_i$ is employed, for the other interfaces for brevity *clk* is omitted and instead of the interface the focus is on the module: $f_{CPU}$ and $f_{MCon}$) , according to $B_i = f_{clk_i} \cdot W_i$. For instance, the DC of the Intel 2600K in [339] has a 384-bit interface and, for this reason, a maximum bandwidth of 384 b/CC. In the case of main memories (composed by DRAMs):

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

[6] 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.

$$B_D = R_D \cdot C_D \cdot f_{clk_D} \cdot W_D, \tag{5.4}$$

where $R_D$ is the data rate of the DRAM (intended as the number of times the data is valid during a cc: 1 for SDR and 2 for DDR), $C_D$ is the number of channels for the main memory[7], $f_{clk_D}$ the clock frequency of the DRAM chips and $W_D$ the width of the interface for each channel. For the DDR3 DRAM employed in the Intel 2600K in [339] $C_D = 2$, $f_{clk_D} = 0.8$ GHz, $W_D = 64$ and therefore $B_D$ is 25.6 GB/s.

A cache-aware roofline model [339], shown in Fig. 5.4, highlights the main benefits of adopting a memory hierarchy similar to Fig. 5.3. When data resides in main memory, $OI^*$ is around 2.50-6.02 FLOP/B (depending on the DRAM technology), while if data resides in a L2C (with $W_X = 64$ b) $OI^*$ becomes 0.25 FLOP/B and a dedicated VC with $W_{C,V} = 356$ b reduces $OI^*$ to 0.04 FLOP/B. Furthermore, from Fig. 5.4 it can be deduced that keeping a processor in a compute-bounded state for a given $OI$ sets 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-bound except for very high $OI$ [322].

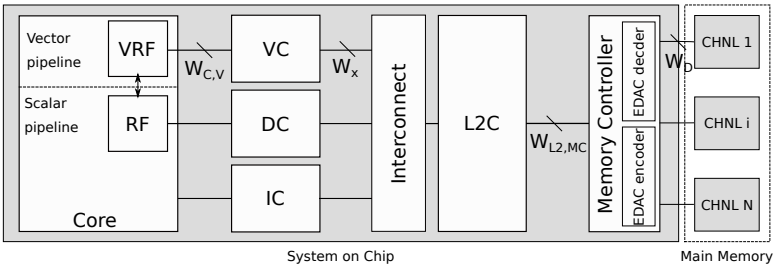


Figure 5.3: Possible memory hierarchy for a vector processor. Other cores and peripherals (not shown in figure) can be connected to the interconnect."CHNL" stands for "channel".

### 5.4.1. MAIN MEMORY

The need for radiation tolerant (or radiation hardened) 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 [162]. The SDR DRAM (ISSI IS42S86400B-7TL) tested in [296] 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 lower compared to the DDR3 DRAM of the Intel 2600K in [339]. Faster DRAMs are also being considered, as the DDR2 tested in [296] (IS43DR81280B-25DBLI) which has 8 bits for Input/Output (I/O) data and achieves up to 400 MHz. This means a $B_D$ of 6.4 Gbps, which is still more

[7] The number of channels represents the number of simultaneous transfers between the processor and main memory. This should not confused with the data rate in each of these channels.

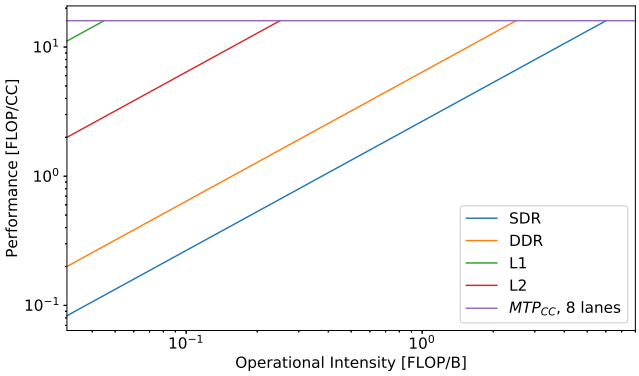


Figure 5.4: Theoretical performance improvement expressed in number of operations per processor CCs for low OI workloads for matrices residing in L2C and VC compared to single-chip SDR (ISSI IS42S86400B-7TL) and DDR2 (IS43DR81280B-25DBLI) memory modules, obtained plotting $\#FLOP/CC = (B_i/f_{CPU}) \cdot OI$ for each of them (with $f_{clk_C} = 1GHz$). The horizontal line is the upper boundary due to computational capabilities of a vector processor with 8 lanes, obtained $\#FLOP/CC = MTP_{CC}$ .



than one order of magnitude lower compared to the DDR3 DRAM of the Intel 2600K in [339].

### EDAC CODES

In the space environment, DRAMs suffer from SEUs and MBUs like SRAMs do [340]. However, as opposed to SRAMs, in DRAMs most of the upsets happen in a set of weakened cells [341]. Furthermore, DRAMs are also more likely to suffer from stuck bits (cells stuck to a value, mostly related to variable bit retention [342]) and SEFIs than SRAMs. SEFIs in a DRAM can affect from some tens of bits to an entire chip per read cycle and can be recovered only with a chip reset or sometimes with a full power cycle [341]. 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 [30]. A $RS(n,k)$ code takes a word of $k$ symbols and generates a codeword 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 [343]. 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) [344]. It is therefore a straightforward choice to use as symbol size the I/O width of a single chip. We will assume the chip to have an I/O 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 [30], 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 [296] with RS reduces from 2.66 Gbps to 2.00 Gbps ($r = 0.25$) and 1.33 Gbps ($r = 0.5$), while 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 [345] has a latency of $L = n+10t+20$ CCs. Typically, critical paths of Memory Controllers (MCons) are deeper than those of processors and run at lower frequency. For instance, the length of the critical path reported in [346] ranges between 547 and 48 gates depending on the design complexity. Assuming 0.02 ns per gate as for Ara [19], this limits the frequency in a range between 1.04 GHz to 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 MCon and L2 compared to the one between L2 and VC. Assuming that the frequency of the interconnect is the same of the processor:

$$W_X = \frac{W_{L2,MCon}}{f_{CPU}/f_{MCon}}, \tag{5.5}$$

where $f_{CPU}$ and $f_{MCon}$ are respectively the frequency of the vector processor and the frequency of the MCon. Therefore, the latency expressed in terms of number of CCs of Ara, keeping into account that $n = (1+r)k$ and $t = (r/2)k$, is

$$L_{CPU} = \frac{f_{CPU}}{f_{MCon}}[(1+r)\cdot k + 6\cdot r\cdot k + 20]. \tag{5.6}$$

Given that $k = W_{L2,MCon}/8$ (as a symbol was assumed to be composed of 8 bits) and $W_X = 32\cdot N_{Lanes}$ (following the rule of thumb reported in Sec. 5.3.2), the final expression is:

$$L_{CPU} = 4N_{Lanes}(1+6r)\left(\frac{f_{CPU}}{f_{MCon}}\right)^2 + 20\frac{f_{CPU}}{f_{MCon}} \tag{5.7}$$

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 $f_{CPU}/f_{MCon}$ ratio is very effective to help the scaling of performance with the number of lanes. Substituting $N_{Lanes} = 4$, $r = 0.25$ and $f_{CPU}/f_{MCon} = 2$, 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 [297], i.e. 15-20 CCs for $f_{CPU} = 1$ 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.

#### Vulnerability of DNN parameters

In order to evaluate the effect of not applying EDAC on the DRAM when running a DNN, we estimate the effect of SEUs on the parameters residing in the DRAM for CloudNet.

According to [341], a 512 Mb SDR DRAM memory (MMSD08512408S-Y) experiences $2.75 \times 10^{-11}$ upset/bit/day in LEO (1336 km altitude, inclination of 66°). 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. 4.4). 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 of the DNN is the Overall Accuracy (*OA*) defined in [299] as:

$$OA = \frac{TN + TP}{\#Pixels}, \tag{5.8}$$

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*, a 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 has 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 [316]. 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 (e.g. $OA = 61.3\%$) are present only if one of the upset is in the MSB of the exponent. Assuming 0.19 upsets/day and that only upsets in the MSB of the exponent will cause a failure due to insufficient 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 [245] it is shown that the FC layers in the last layers of CNNs are more vulnerable compared to 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 [347] 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, [245] shows that using Half Precision (HP) floating point can increase robustness for some architectures compared to SP floating point.

Fixed point representation can mitigate the failure mechanism described for floating point thanks to their limited range [245]. 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 to floating point representations [245].

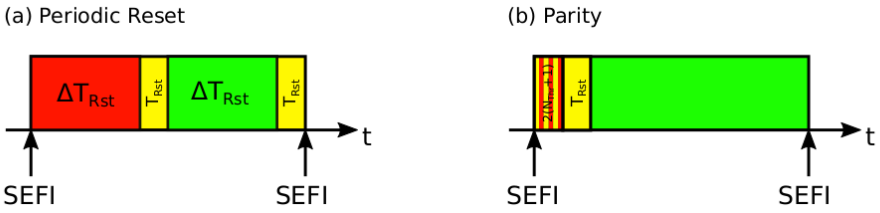


Figure 5.5: Fraction of time (red) where all the inferences are wrong because of SEFIs and the system unavailable (yellow), in case of a periodical reset (left) and in case of parity (right). Intervals where all inferences are correct are represented in green, intervals where an inference has half the probability of being detected (unavailable) or undetected (wrong inferences) are represented with yellow and red stripes. Fraction of times are not to scale.



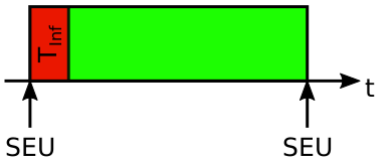


Figure 5.6: Fraction of time where inferences are wrong (red) because of SEUs (case without EDAC). A similar figure holds when parity is employed, where the red is replaced by yellow (unavailable). Fraction of times are not to scale.

### 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 [341], a 512 Mb SDR DRAM memory (MMSD08512408S-Y) experiences 1.33e-3 SEFI/device/day in LEO. To achieve the peak memory required (identified in Sec. 4.4.1), 14 chips are required and therefore not including any EDAC will produce a SEFI 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, as shown in Fig. 5.5 (a). Assuming a reset every 2 hours and, as a worst case estimation, that the SEFI happens just after a reset, the percentage of failed inferences due to SEFIs $WI_{SEFI}$ in the worst case is:

$$WI_{SEFI} = \frac{Failures(SEFI)}{Total\ inferences} = \frac{\Delta T_{Rst}}{MTTF_{SEFI}} = 0.16\%. \tag{5.9}$$

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. 5.4.1 on the vulnerable bits of floating point coefficients and $T_{Rst}$ is replaced with the time required for a single inference $T_{Inf}$, which we assume to be 5 s. The value found

is negligible (two order of magnitude less than the contribution of SEFIs). However, it should be considered that, while $WI_{SEFI}$ is independent from the performance in terms of $inferences/s = 1/T_{Inf}$, $WI_{SEU}$ increases for slower systems. The final value of average reliability, i.e. the fraction of correct inferences, $R_{Avg} = 1 - WI_{SEU} - WI_{SEFI}$ (99.84%) can be not deemed enough for critical applications. The availability when there is a periodical reset instead depends also on the unavailability due to the maintenance time after a reset. In this case, as shown in Fig. 5.5 (a):

$$Availability_{SEFI} = 1 - \frac{T_{Rst} \cdot \left\lfloor \frac{MTTF_{SEFI}}{\Delta T_{Rst} + T_{Rst}} \right\rfloor + \Delta T_{Rst}}{MTTF_{SEFI}}. \tag{5.10}$$

If a maintenance time of 30 s is assumed for each reset, we find that the availability of the service is 99.43%, while a maintenance time of 300 s produces an availability of 95.85%. Both values do not meet typical requirements of dependable systems (e.g. 99.9% [145]).

A trade-off 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. The effects of this approach are shown in Fig. 5.5 (b). 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:

$$WI_{SEFI} = \frac{N_{thr} + 1}{MTTF_{SEFI}/T_{Inf}} = 0.0004\%. \tag{5.11}$$

Regarding SEUs, neglecting accumulation and MBUs, all the upsets are detected and $WI_{SEU} = 0$. Therefore $R_{av} = 99.9996\%$, which is a substantial improvement compared to employing no EDAC. There is a substantial improvement in availability too. In this case, as shown in Fig. 5.6 (b):

$$Availability_{SEFI} = 1 - U_{SEFI} = 1 - \frac{2(N_{thr} + 1) \cdot T_{Inf} + T_{Rst}}{MTTF_{SEFI}}, \tag{5.12}$$

which yields 99.998% and 99.993% respectively for 30 s and 300 s of unavailability per reset ($T_{Rst}$). This shows that employing parity limits the effect on the system of longer maintenance periods. However, in this case there is also a component of unavailability due to SEUs (assuming, as a worst case that all the inference is to be repeated when an error is detected):

$$Availability_{SEU} = 1 - U_{SEU} = 1 - \frac{T_{Inf}}{MTTF_{SEU}}, \tag{5.13}$$

Table 5.2: Approaches suggested for applications with different criticality levels (reliability/availability) and achievable performance. The failure rate due to low QoS is indicated with $\lambda_{QoS}$. Values for the RS approach are reported to be (almost) ideal, as the error models employed in this analysis do not find any failure with RS.

| Approach | No EDAC | Parity | RS |
|---|---|---|---|
| Reset strategy | Periodic | Threshold | After SEFI |
| $\lambda_{QoS}$ | $0.03\lambda_{SEU} + \lambda_{SEFI}$ | $\lambda_{SEFI}$ | $\simeq 0$ |
| $R_{Avg}$ | 99.84% | 99.9996% | $\simeq 100\%$ |
| Availability ($300s - 30s$) | 95.85-99.43% | 99.992-99.997% | 99.994-99.9997% |
| Performance | High | Medium | Low |

which yields 99.9988%, comparable with the values found for SEFIs. Therefore the total availability is $Availability = 1 - U_{SEU} - U_{SEFI}$, which yields respectively 99.997% and 99.992%. These value are considerably higher than those found applying a periodic reset, meeting typical availability requirements of safety criminal space systems (e.g. 99.9%). Assuming an RS code capable of correcting a single symbol and a reset of the chip carried out every time a SEFI is detected, the introduced error models do not identify any loss of reliability[8]. However, the reset causes a loss of availability, according to Eq. 5.14:

$$Availability = 1 - U_{SEFI} = 1 - \frac{T_{Rst}}{MTTF_{SEFI}}, \tag{5.14}$$

which yields 99.9997% and 99.994% respectively for a $T_{Rst}$ of 30 s and 300 s. Table 5.2 summarizes the different EDAC and reset approaches discussed to protect DRAMs for DNNs.

### 5.4.2. L1 VECTOR CACHE

Many vector processors use L1 caching for instructions and for scalar data, leaving vector data uncached (e.g. Ara [19]), as historically locality in vector workloads was assumed to be less pronounced compared to scalar workloads [348]. The work in [348] 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 [349] it is shown that the use of caches helps masking memory latency, as increasing by 3.21x 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 a L1 cache for vector data.

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

#### SIZE

From Table 4.1, 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

[8] An alternative approach could be resetting when the availability is not reduced (e.g. during routine operations, plausible given that SEFIs occur every 53.7 days on average). However this approach will damage reliability, as there is a non-zero opportunity of a second SEFI on another chip, that will cause wrong inferences.

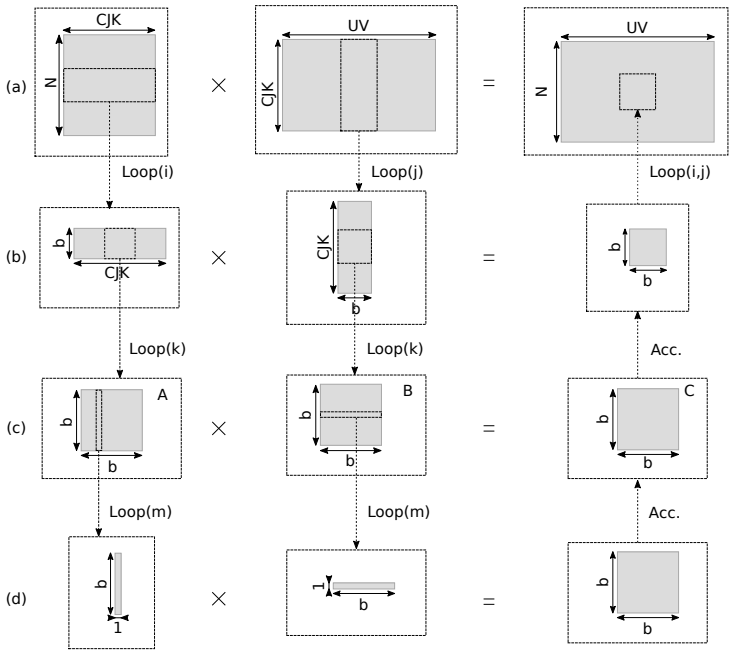


Figure 5.7: Example of tiling of a matrix-matrix multiplication. 'Acc.' stands for accumulation.

[30]). This problem can be addressed with tiling, as shown in Fig. 5.7. In this approach, two levels of looping (shown in Fig. 5.7 with index $i$ and $j$) select a subset of the matrix-matrix multiplication which produces one of the $\lceil \frac{UV}{b} \rceil \lceil \frac{N}{b} \rceil$ tiles of dimensions $b \times b$ of the result. By increasing the size of the cache, it is possible to work on larger matrix blocks residing in the VC. The subset of operations obtained in Fig 5.7 (b) can be decomposed in $\lceil \frac{CJK}{b} \rceil$ segments, and the results of these segments can be accumulated to generate the final result of the tile. The level (c) in Fig. 5.7 is where the mapping to *igemm* (decribed in Sec. 4.4.1) can be applied.

One of the possible implementations of *igemm* (Fig. 5.7 (d)) is a loop selecting the $m^{th}$ column of $A$ and the $m^{th}$ row of $B$ and generating a matrix where the $p^{th}$ column is the $m^{th}$ column of $A$ multiplied by $b_{mp}$. Vectorization is applied with maximum vector length of *VLMAX*, with FMA (accumulate) operations between the column vector $a_m$ and a scalar $b_{mp}$. A matrix representation of this implementation for a 2×2 example is shown below.

$$C = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix} \tag{5.15}$$

$$C = \begin{pmatrix} | & | \\ a_1 b_{11} & a_1 b_{12} \\ | & | \end{pmatrix} + \begin{pmatrix} | & | \\ a_2 b_{21} & a_2 b_{22} \\ | & | \end{pmatrix}. \tag{5.16}$$

As the focus in this section is on investigating the speed increase due to the use of a VC for small matrices, memory-bounded conditions will be assumed (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 move the matrices from main memory to the VC and the time required to write the result to main memory.

Loading a vector of length *VLMAX* from main memory takes:

$$T_{L,V} = T_{LM} + \frac{SEW \cdot VLMAX}{B_M}, \tag{5.17}$$

where $SEW$ is the size of a single element of the vector, $B_M$ is the bandwidth of the main memory and $T_{LM}$ is the latency of the first element of the vector from main memory[9]. The time required to copy a row vector of length $b$ from main memory to VC is:

$$T_{L,b} = T_{L,V} \cdot \left\lceil \frac{b}{VLMAX} \right\rceil + \frac{SEW \cdot b}{B_M}, \tag{5.18}$$



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

$$T_{L,b\times b'} = \left( T_L \cdot \left\lceil \frac{b'}{VLMAX} \right\rceil + \frac{SEW \cdot b'}{B_M} \right) \cdot b. \tag{5.19}$$

There are three possible implementations, depending on which tile (of the coefficient, input feature and output feature matrix) is kept into the VC during the innermost looping, i.e. Loop(m) in Fig. 5.7. Considering the associated continuous functions (without modulo, ceiling and floor functions), it is possible to show that the fastest implementation is the one keeping in VC 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. Assuming that the output feature matrix is kept in VC, the time required during the loop on CJK to load all the tiles in a $CJK \times b$ stripe of the $CJK \times UV$ input feature matrix, as shown in Fig.5.7 (b), is:

$$T_{L,CJK\times b} = T_{L,b\times b} \cdot \frac{CJK}{b}, \tag{5.20}$$

while for a $b \times CJK$ stripe of the $N \times CJK$ matrix is:

$$T_{L,b\times CJK} = T_{L,b\times b} \cdot \left\lfloor \frac{CJK}{b} \right\rfloor + T_{L,b\times b'}, \tag{5.21}$$

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

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

$$T_{L,N\times CJK} = T_{L,b\times CJK} \cdot \left\lceil \frac{UV}{b} \right\rceil \cdot \frac{N}{b}, \tag{5.22}$$

where the ceiling is required because all the matrix of the coefficients 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:

$$T_{L,CJK\times UV} = \left( T_{L,CJK\times b} \cdot \left\lfloor \frac{UV}{b} \right\rfloor + T_{L,b\times b'} \cdot \frac{CJK}{b} \right) \left\lceil \frac{N}{b} \right\rceil. \tag{5.23}$$

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

$$T_{S,N\times UV} = \left( T_{S,N\times b} \cdot \left\lfloor \frac{UV}{b} \right\rfloor + T_{L,b\times b'} \cdot \frac{N}{b} \right). \tag{5.24}$$

To trade-off the speedup against the increase in size due to a larger VC, the area efficiency in terms of FLOP/CC/GE for matrix multiplications with matrices residing in VC will be considered. To give a realistic estimation of the cache size that maximizes the area efficiency, the increase of area due to the inclusion of a VC in Ara will be considered. The area of Ariane and Ara ranges from 2228 for two lanes to 10735 kGE for 16 lanes. As a worst case for memory-bounded conditions, we assume 16 lanes ($VLMAX = 16$) and in this case the area without VC is 10735 GE. The area of the L1 cache is estimated as $A_{VC,GE} = (6/4) \cdot N_b$, assuming 6T SRAM cells and a GE corresponding to four transistors.

In this section, 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) are condidered. Table 5.3 shows the results of this model. The main observations are that the optimal size of VC is much larger (256 KiB-1 MiB) than a typical DC (e.g. 16 KiB [30]) and that the most impacting factor on the area 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. The optimum size of the vector cache is instead largely independent from the latency and bandwidth of the main memory, showing that a single design can be employed efficiently for different applications with different requirements in terms of dependability.

#### ORGANIZATION

The model in the previous section assumes that it is possible to keep the tiles in VC, avoiding that loading a vector of one of the tiles causes the eviction of data belonging

Table 5.3: Estimates of area $A_{tot}$ [MGE] and area efficiency AE [FLOP/CC/MGE] for a 16 lane vector processor with different sizes of VC, main memory (latency and bandwidth) and maximum size of the tile $b \times b$ when applying tiling to the layers of CloudNet.

| Size | 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 | **2.44E+0** | 2.34E+0 |
| $AE_{(50,4)}$ | 4.09E-1 | 5.44E-1 | 7.23E-1 | 8.59E-1 | **8.82E-1** | 8.28E-1 |
| $AE_{(300,1)}$ | 1.57E-1 | 2.14E-1 | 2.84E-1 | 3.48E-1 | **3.59E-1** | 3.44E-1 |
| $AE_{(300,10)}$ | 2.06E-1 | 2.83E-1 | 3.76E-1 | 4.68E-1 | **4.86E-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 | **2.40E+0** | 2.33E+0 | 2.08E+0 |
| $AE_{(50,4)}$ | 2.62E-1 | 5.97E-1 | 7.65E-1 | **8.72E-1** | 8.43E-1 | 7.33E-1 |
| $AE_{(300,4)}$ | 6.48E-2 | 2.35E-1 | 3.01E-1 | **3.54E-1** | 3.43E-1 | 3.05E-1 |
| $AE_{(300,40)}$ | 7.09E-2 | 3.11E-1 | 3.99E-1 | **4.78E-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 | **7.44E-1** | 7.10E-1 | 5.74E-1 | 4.15E-1 |
| $AE_{(50,4)}$ | 3.69E-1 | 2.77E-1 | **2.89E-1** | 2.68E-1 | 2.13E-1 | 1.50E-1 |
| $AE_{(300,4)}$ | 9.35E-2 | 1.06E-1 | **1.11E-1** | 1.05E-01 | 8.51E-2 | 6.11E-2 |
| $AE_{(300,40)}$ | 1.21E-1 | 1.38E-1 | **1.45E-1** | 1.40E-1 | 1.13E-1 | 8.26E-2 |



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 [336]) typically support vector load and store operations with non-unit stride $V_S$, i.e. two contiguous elements of the vector are placed in non-contiguous location separated by $V_S - 1$ elements. According to the model in [349], the fraction of non-unit strides in a workload determines whether organizations similar to scalar processors are enough to achieve acceptable performance or organizations specific for vector processors are required. One example of the latter are prime-mapped caches [349], 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 [330] 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 [336]). The improvement with prime-mapped caches for a typical workload with unit stride of 70% is 2x over the cacheless version, while the improvement for direct mapped caches is below 1.5x [349].

Typical applications which require non-unit strides are Fast Fourier Transform (FFT) and its inverse Inverse Fourier Transform (IFT) [349]. FFT is employed in several compute-intensive workloads. For instance, in [350] it is proposed to speedup CNN execution, as

convolutions can be substituted by a sequence of FFT, element-wise multiplication and IFT.

To investigate whether vector loads and stores with non-unit 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[10]. 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 [313] model did not produce non-unit 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 non-unit vector strides, in general different cache organizations are needed compared to those typically employed for scalar processors.

#### 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 caches is modified and an old value is read from then other. This can be addressed with a write-through policy for VC and DC, although this comes with substantial penalties especially in terms of power [351], memory traffic [251] and performance [115].



## 5.5. CLOUDNET KERNELS FOR RISC-V VECTOR PROCESSORS

After having investigated the potential and criticalities of the microarchitecture of vector processors, in this section the impact of the RVVE on the kernels of CloudNet is shown. The aim of this section is to investigate which RVVE instructions have the potential to speedup DNNs kernels. In Chapter 6 these instructions will be implemented in a hardware prototype and their efficiency in speeding up CloudNet evaluated.

Furthermore, discussion in Sec. 5.3 showed that the need of an FPU per lane, to support FP operations, implies large area increase that cannot be tolerated in highly constrained embedded systems. For this reason, in this section the kernels will be implemented on integer elements instead of floating point elements, and comparisons will be carried out with integer versions of the kernels reported in 4.4.1.

### 5.5.1. METHODOLOGY

The use of instrinsics allows considering only a subset of the RVVE, as the generated instructions depend on the intrinsic functions employed [352]. For instance, the intrinsic function:

```
size_t vsetvl_e32m1 (size_t avl)
```

generates a *vsetvli* instruction with a $SEW = 32$ (*e32*) and no register grouping (*m1*), having in the field *rs1* the register address of the register storing AVL. The function returns the value of *vl* in the register indicated in the field *rd* of the related assembly instruction, which is implicitly selected by the user by assigning the result of the intrinsic function to a variable. Examples of use of this intrinsic function are reported in Sec. 6.3.

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

### IMPACT ON PERFORMANCE

Keeping in mind the performance model provided in Eq. 1.1, the main criterion to evaluate the effectiveness of the addition of vector instructions to a scalar processor is the factor of which the number of instruction *NI* is reduced when moving from a scalar to a vector function, given by $NI_s/NI_v$. For brevity, this factor will be called ideal speedup, as it is the speedup due to the use of vector instructions in a hardware prototype when there are no penalties on *CPI* and $f_{clk}$ (see Eq. 1.1). The choice of this parameter is due to the fact that at this point a hardware prototype is not available yet. In Chapter 6, when the required instructions will be implemented in the prototype, also the impact on *CPI* and $f_{clk}$ will be evaluated.

In order to measure the decrease in terms of NI, both versions of each kernel were executed on a version of the Spike ISA simulator supporting the RVVE [11].

### VERIFICATION

In order to verify that the vectorized C implementations of the kernels are functionally equivalent to the scalar implementations proposed in Chapter 4, both versions of each kernel were executed on a version of the Spike ISA simulator supporting the RVVE [12]. The same randomized input, using the function *rand()* from the standard library *stdlib.h*, was fed to both functions and the outputs were checked with a simple loop (Fig. 5.8) which prints the differences found (if any). Furthermore, if a difference is found, the processor enters an infinite loop and the program hangs. The latter functionality lets the user know that the result is wrong even in runs where the *printf* functionality is not implemented. The verification has been successful for all the kernels reported in this chapter.

```
int pass = 1;
for (int i = 0; i < OUTPUT_LEN; i++) {
  if (golden[i] != c[i]) {
     printf("index %d failed, %d!=%d\n", i, golden_array[i], c_array[i]);
    pass = 0;
  }
}

while(!pass);
```

Figure 5.8: Loop to verify the functional equivalence between scalar and vector implementations.

Furthermore, the measures of *NI* are employed to validate the analytical models provided to analyze the effect of *VLMAX* and of matrix sizes on the ideal speedup. The nomenclature of the variables of the analytical models follows the one shown in the case of a nested loop in Fig. 5.9.

### PROCEDURE

The first step is compiling:

[11] https://github.com/riscv-software-src/riscv-isa-sim, checkout: debdad850ba3c97171b027fc590db1e83cd9383a

[12] https://github.com/riscv-software-src/riscv-isa-sim, checkout: debdad850ba3c97171b027fc590db1e83cd9383a

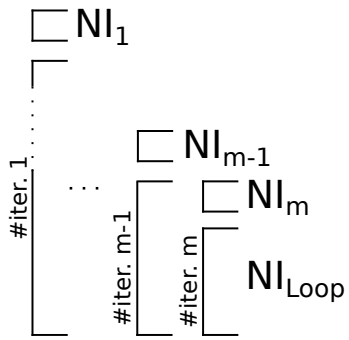


Figure 5.9: Nomenclature for the number of instructions in a kernel composed of nested loops. The number of iterations *#iter.* $i$ is a function of the matrix sizes and $VLMAX$. The $NI_i$ chunks are those executed before and after (in figure indicated only before) the $loopi$ (executed $\#iter._i$ times).

```
riscv64-unknown-elf-gcc -O2 -o conc  conc.c
```

Then the executable is run with Spike:

```
spike --isa=RV64IMAV --varch=vlen:4096,elen:32 -l  pk conc  2>conc.log
```

where the switch *-l* is employed to generate a log of instructions, which with *2>conc.log* prints the executed instruction in a *conc.log* logfile. The switch *–varch=vlen:4096,elen:32* is employed to select a specific *VLEN* and *ELEN*. If the execution completes successfully and no difference is found then the two implementations are functionally equivalent. To count the number of instruction for both implementations, a dump of the instructions in the object file is executed with:

```
riscv64-unknown-elf-objdump --disassemble-all  conc > conc.dump
```

### 5.5.2. IMPLEMENTATION OF THE KERNELS

This section contains C implementations of the CloudNet kernels using the RVVE intrinsics. The functions proposed here will be compared to the one presented for scalar processors in Sec. 4.4.1.

#### CONVOLUTIONAL LAYERS

As opposed to the matrix multiplication implementation described in Sec. 5.4.2 (operating on columns), here we use an implementation that operates on rows of matrix $B$ and scalar coefficients from matrix $A$, according to the decomposition shown in Eq. 5.25:

$$C = \begin{pmatrix} a_{11}(— \ b_1 \ —) \\ a_{21}(— \ b_1 \ —) \end{pmatrix} + \begin{pmatrix} a_{12}(— \ b_2 \ —) \\ a_{22}(— \ b_2 \ —) \end{pmatrix}. \tag{5.25}$$

The main advantage of this implementation is that it does not require vector stride loads and stores. The function implementing *igemm* is shown in Fig. 5.10, where the loop on

```
void igemm_vec(const int *a, size_t lda, const int *b, // k * n matrix
               size_t ldb, int *c, size_t ldc) {
  size_t vl;
  for (int i = 0; i < n1; ++i) {
    const int *b_k_ptr = b;
    int *c_k_ptr = c;
    for (int c_k_count = n3; (vl = vsetvl_e32m1(c_k_count )); c_k_count -= vl) {
      const int *a_j_ptr = a;
      const int *b_j_ptr = b_k_ptr;
      vint32m1_t acc = vle32_v_i32m1(c_k_ptr);
      for (size_t k = 0; k < n2; ++k) {
        vint32m1_t b_k_data = vle32_v_i32m1(b_j_ptr);
        acc = vmacc_vx_i32m1(acc, *a_j_ptr, b_k_data);
        b_j_ptr += n3;
        a_j_ptr++;}
      vse32_v_i32m1(c_k_ptr, acc);
      c_k_ptr += vl;
      b_k_ptr += vl;}
    a += n2;
    c += n3;}}
```

Figure 5.10: C function employed for *igemm* on the vector processor.

$n_3$ is parallelized, accumulating the result on *VLMAX* elements of a row. The parallelization of these two implies that the amount of loop iterations compared to the scalar implementation is divided by a factor *VLMAX*. Eq. 5.26 describes the ideal speedup when moving from the scalar to the vector implementation:

$$\frac{NI_s}{NI_v} = \frac{NI_{1s} + n_1 \cdot \left[NI_{2s} + n_3 \cdot \left(NI_{3s} + n_2 \cdot NI_{Loop,s}\right)\right]}{NI_{1v} + n_1 \cdot \left[NI_{2v} + \left\lceil \frac{n_3}{VLMAX} \right\rceil \left(NI_{3v} + n_2 \cdot NI_{Loop,v}\right)\right]}. \tag{5.26}$$

Fig 5.11 shows the equation above in case of $n_1 = n_2 = n_3 = n$, when varying $n$. The parameters employed were found from Spike simulations and are $NI_{1v} = 22$, $NI_{2v} = 9$, $NI_{3v} = 14$, $NI_{Loop,v} = 10$, $NI_{1s} = 15$, $NI_{2s} = 9$, $NI_{3s} = 8$ and $NI_{Loop,s} = 7$. The equation has been verified to produce the same results of a Spike simulation for the case with $n = 128$ and $VLMAX = 32$ ($NI_s/NI_v$=22.32) and the case with $VLMAX = 128$ ($NI_s/NI_v$=88.80).

Regarding the behavior when $n$ is fixed and *VLMAX* is varied, a roughly linear behavior is obtained until $VLMAX = n$ is reached. Further increasing *VLMAX* does not provide any advantage. The almost-linear behavior can be explained noting that the function is of the type $a/(b + c/VLMAX)$, which for $b << c/VLMAX$ can be approximated as $a/(c/VLMAX)$ as shown below:

$$\frac{NI_s}{NI_v} \approx \frac{NI_{1,s} + n_1 \cdot \left[NI_{2,s} + n_3 \cdot \left(NI_{3,s} + n_2 \cdot NI_{Loop,s}\right)\right]}{n_1 \cdot \left\lceil \frac{n_3}{VLMAX} \right\rceil \left(NI_{3,v} + n_2 \cdot NI_{Loop,v}\right)}. \tag{5.27}$$

The behavior predicted with this approximated equation is shown with a solid blue line in Fig. 5.12, along with red dots from the complete equation.

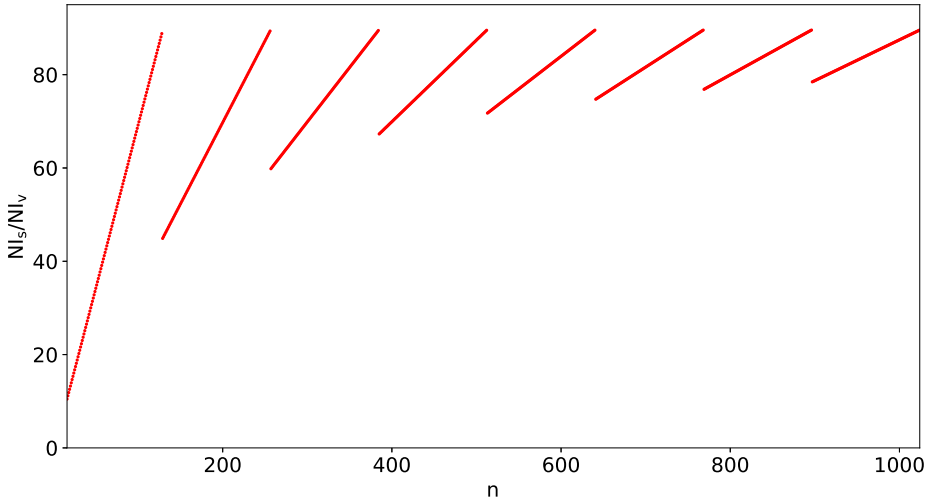


Figure 5.11: Ideal speedup for *igemm* when employing vector instructions instead of scalar instructions (Eq. 5.26).

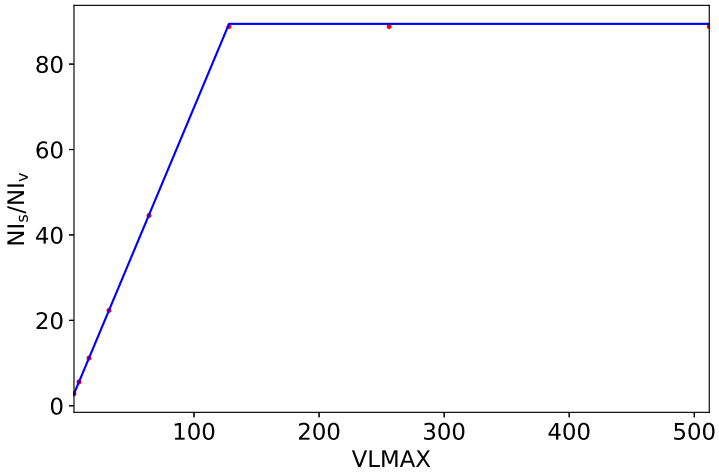


Figure 5.12: Ideal speedup when increasing the VLMAX from 4 to 512, with $n1 = n2 = n3 = n - 128$. In blue, the linearized continuous function associated with Eq. 5.27, in red actual values for allowed *VLMAX* integer values

### CONCATENATE LAYERS

Although no operations are present in concatenate layers, the loads and the stores required to copy A and B into the concatenated matrix C can be parallelized. In this case,

```
void conc_vec(int *a, int *b, int *c) {
  size_t vl;
  int *c_n_ptr = c;
  int *b_n_ptr = b;
  int *a_n_ptr = a;
  vint32m1_t acc;

  for (int n112 = n1*n1*n2; (vl = vsetvl_e32m1(n112)); n112 -= vl){
    acc = vle32_v_i32m1(a_n_ptr);
    vse32_v_i32m1(c_n_ptr, acc);
    c_n_ptr += vl;
    a_n_ptr += vl;}

  for (int n113 = n1*n1*n3; (vl = vsetvl_e32m1(n113 )); n113 -= vl) {
    acc = vle32_v_i32m1(b_n_ptr);
    vse32_v_i32m1(c_n_ptr, acc);
    c_n_ptr += vl;
    b_n_ptr += vl;}}
```

Figure 5.13: C function to implement *conc* on the vector processor.

the expected gain is avoiding the latency of main memory each time a single element is loaded and stored, and the cache misses of the scalar pipeline. Fig. 5.13 show this implementation on vector processor. Increasing *VLMAX* will decrease the number of instructions until VLMAX approaches ${n_1}^2 \cdot n_2$ for the first loop and ${n_1}^2 \cdot n_3$ for the second one. The ideal speedup can be found as:

$$\frac{NI_s}{NI_v} = \frac{NI_{1s} + \left\lceil \frac{{n_1}^2(n_2+n_3)}{CPL} \right\rceil \cdot NI_{Loop,s}}{NI_{1v} + \left( \left\lceil \frac{n_1^2 \cdot n_2}{VLMAX} \right\rceil + \left\lceil \frac{n_1^2 \cdot n_3}{VLMAX} \right\rceil \right) NI_{Loop,v}}, \tag{5.28}$$

where *CPL* is the number of elements copied in a single loop. After the division by *CPL*, no ceiling function is applied because both ${n_1}^2 \cdot n_2$ and ${n_1}^2 \cdot n_3$ are divisible by the *CPL* selected by the compiler, i.e. 8.

Considering Layer 1 in Table 4.2 ($n_1 = 192$, $n_2 = n_3 = 16$) when $VLMAX = 128$, NI is reduced by a factor of 19.55. When considering Layer 6, the one with the lowest parallelization possible for both loops (${n_1}^2 \cdot n_2 = {n_1}^2 \cdot n_3 = 18432$), the ideal speedup is by a factor of 19.51. Regarding the kernels with the higher parallelization possible, for the layer with the highest one for the first loop (layer 32 with ${n_1}^2 \cdot n_2 = 17694720$ and) NI is reduced by 19,56x (with $NI_{1v} = 8$, $NI_{Loop,v} = 9$, $NI_{1s} = 14$ and $NI_{Loop,v} = 11$). Given that

$$NI_{1v} << \frac{{n_1}^2 \cdot n_2 + {n_1}^2 \cdot n_3}{VLMAX} \cdot NI_{Loop,v} \tag{5.29}$$

and

$$NI_{1s} << \frac{n1^2 \cdot n2 + n1^2 \cdot n3}{CPL} \cdot NI_{Loop,s}, \tag{5.30}$$

the ideal speedup for large matrices is almost independent from their size, as shown in the resulting equation:

$$\frac{NI_s}{NI_v} \approx \frac{NI_{Loop,s}}{NI_{Loop,v} \cdot CPL} \cdot VLMAX. \tag{5.31}$$

Fig 5.14 shows that for already relatively small $n$ (around 10), the ideal speedup approaches a value independent of $n$. On the other hand, Fig. 5.15 shows that there is a linear increase of $NI_s/NI_v$ when $VLMAX$ is increased up to $VLMAX = {n_1}^2 \cdot n_2 = {n_1}^2 \cdot n3$ is reached. This is also confirmed by empirical data from Spike simulations: layer 6 with $VLMAX = 64$ has an ideal speedup of 9.77x, with $VLMAX = 32$ of 4.89x. In this case, $NI_0 = 4$, $NI_{Loop,s} = 11$ and $NI_{Loop,v} = 9$.

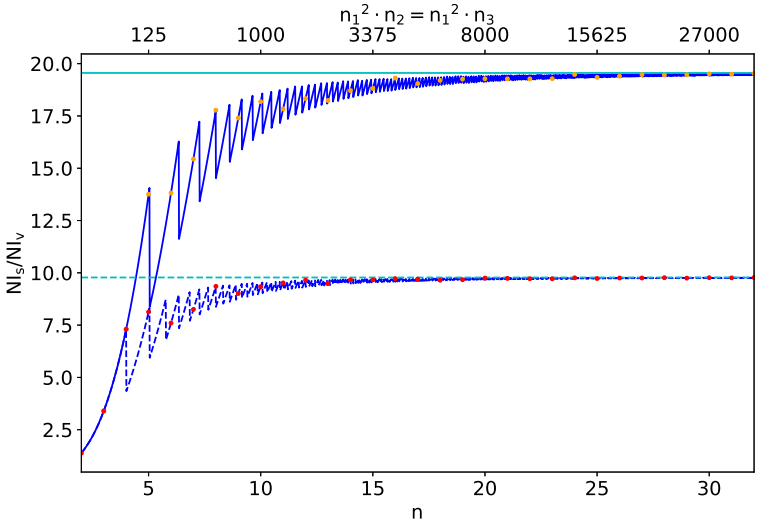


Figure 5.14: Ideal speedup for concatenation layers when employing vector instructions instead of scalar instructions for *conc*. In blue, the continuous function associated with Eq. 5.28 for $VLMAX = 128$ (solid line) and for $VLMAX = 64$ (dashed line). In orange and red dots, the values for integer $n$ for the former and for the latter. In cyan, the asymptotic value for $VLMAX = 128$ (solid line) and for $VLMAX = 64$ (dashed line).

### BATCH NORMALIZATION

Fig. 5.16 shows how *batchnorm_vec* has been implemented with RVVE. As the RVVE does not define a Fused Multiply-Add (FMA) instruction with a vector operand and two scalar operands, and expanding one of the two scalars into a large vector (minimum length) is deemed inefficient, the implementation has been carried out on each batch with a vector loop executing the *vmul.vx* and *vadd.vx* instructions on the input matrix. In this case, the normalization can be parallelized on a complete ${n_1}^2$ batch. Although in CloudNet the median value of this potential parallelization factor is high (576) (see Tab. 4.4), three layers out of 31 (Layers 18, 19, and 20) have a value of 36.

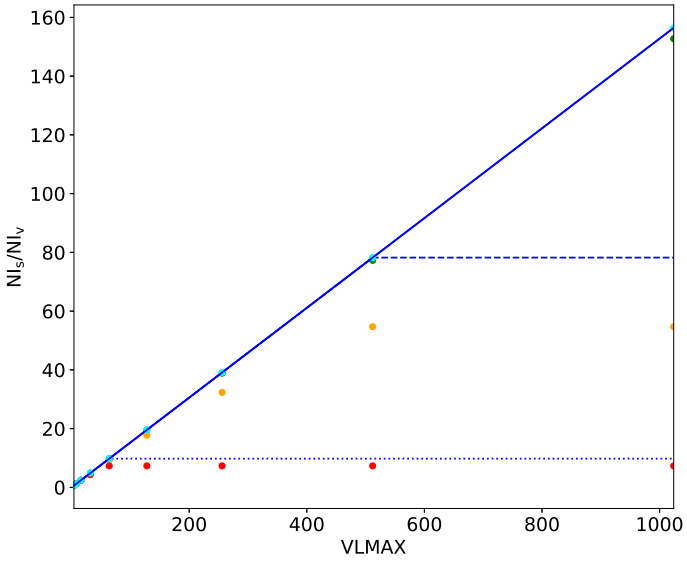


Figure 5.15: Ideal speedup when employing vector instructions instead of scalar instructions for *conc* when varying *VLMAX*. In blue, the continuous function associated with Eq. 5.32. The solid line represent the linear approximation for Layer 32, the dotted line ${n_1}^2 \cdot n_2 = {n_1}^2 n_3 = 512$ and the dashed line for ${n_1}^2 n_2 = {n_1}^2 n_3 = 64$. In cyan and orange the actual ideal speedup values for allowed integer values of *VLMAX* in case of Layer 32 and 6 respectively, and in green and red for ${n_1}^2 n_2 = {n_1}^2 n_3 = 512$ and ${n_1}^2 n_2 = {n_1}^2 n_3 = 64$ respectively.

```
void batchnorm_vec() {
  size_t vl;
  vint32m1_t va, vb;
  for (size_t i = 0; i < n2; ++i){
    int32_t batchsize = nl*nl;
    for (;vl = vsetvl_e32m1 (batchsize); batchsize-= vl) {
      va = vle32_v_i32m1 (a);
      vb = vmul_vx_i32m1 (va, gamma_denom[i]);
      vb = vadd_vx_i32m1 (vb, mean_beta_denom[i]);
      vse32_v_i32m1 (b,vb);
      a += vl; b += vl;}}}
```

Figure 5.16: C function for *batchnorm_vec* on the vector processor.

The peak parallelization is possible for the layers with $n_1 = 192$, i.e. Layers 1, 2, 30 and 31. Considering Layer 1 ($n_1 = 192$ and $n_2 = 32$), the ideal speedup for $VLMAX = 128$ is 88.47. Considering one of the layers with the minimum parallelization possible (Layer 19, with $n_1 = 6$ and $n_2 = 512$), the ideal speedup is 17.41.

$$\frac{NI_s}{NI_v} = \frac{NI_{1s} + n_2 \cdot (NI_{2s} + {n_1}^2 NI_{Loop,s})}{NI_{1v} + n_2 \cdot \left(NI_{2v} + \left\lceil \frac{{n_1}^2}{VLMAX} \right\rceil \cdot NI_{Loop,v}\right)}. \tag{5.32}$$

As shown in Fig. 5.17, for relative large matrices, the ideal speedup tends to

$$\frac{NI_s}{NI_v} \approx \frac{NI_{Loop,s}}{NI_{Loop,v}} \cdot VLMAX. \tag{5.33}$$

Furthermore, from Fig. 5.17, it can be seen that $n2$ has low impact on the ideal speedup, as the cyan (largest $n_2$) dots are very close to the blue line (smallest $n_2$). Although the differences between red dots and cyan dots are visible for small $n_1$ (performance for larger $n_2$ is slightly higher), they become almost unnoticeable for around $n_1 > 40$.



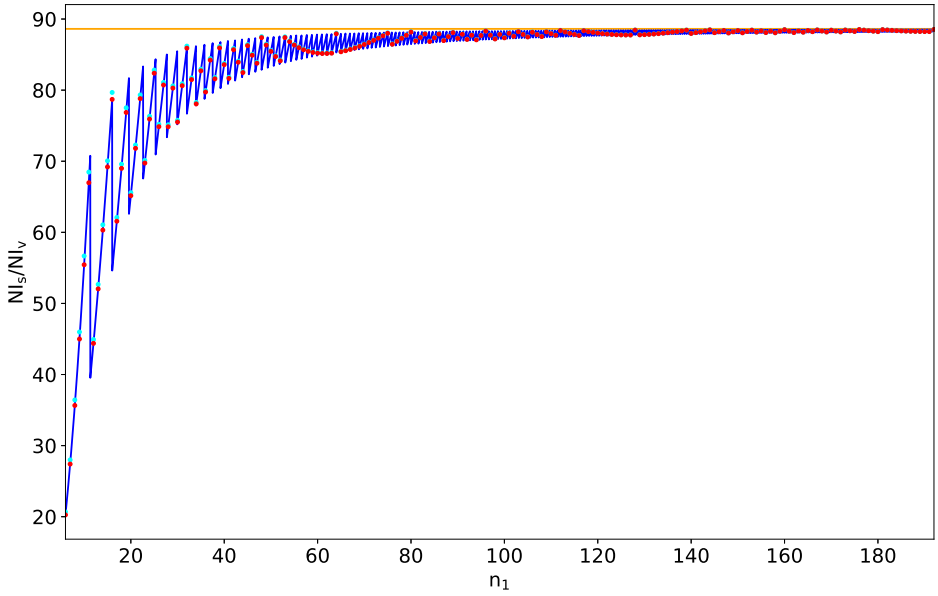


Figure 5.17: Ideal speedup when employing vector instructions instead of scalar instructions for *batchnorm*. In blue, the continuous function associated with Eq. 5.32 for $n2 = 16$. In red, values for integer $n_1$ in case of $n_2 = 16$. In cyan, values for integer $n_1$ in case of $n2 = 1024$. In orange, the asymptotic value for large $n1$, which in this case is around 0.69 times *VLMAX* (i.e. 88.62). The range of $n_1$ considered (6-192) was defined from the minimum and maximum values in Table 4.4.

### SUBSAMPLING LAYERS

Fig. 5.18 shows how to implement *maxpool2d_totalvec* with RVVE[13]. The function uses an intermediate $B_t$ matrix of size $(n_1/s)\times n_1 \times n_2$. In this case, the first loop, the one comparing rows, can be easily parallelized with the *vmax_vv* instruction, i.e. an instruction comparing element-by-element two vectors and writing the maximum for each of them in an output vector. To parallelize the loop second loop, i.e. the one comparing columns, non-unit stride load and store instructions are needed, because (as explained in Sec. 5.4) elements in the C language are stored in row-major order, i.e. elements of a row are contiguous in memory while elements of a column of the input are $4 \cdot n_1$ bytes apart, with $n_1$ being number of columns. While the parallelization of the first loop achieves relatively low values (the maximum value of $n_1$ in Table 4.3 is 96, with a median value of 24 and a minimum value of 6), the second loop deals with vectors of $n_2 \cdot n_1/s$ elements (the maximum value in Table 4.3 for this is 3072, with a median value of 1536 and a minimum of 384). However, support of non-unit strides puts higher constraints on the memory subsystem. For this reason, a second implementation is provided (*maxpool2d_halfvec*), where the second loop (the one requiring non-unit stride load and stores) is replaced with a scalar implementation. Fig. 5.19 shows the scalar loop employed. In the case of $s = 2$:



$$NI_{s1} = NI_{1s1} + n_2(NI_{2s1} + n_1[NI_{3s1} + \frac{n_1}{s}(NI_{Loop,s1} + P_r \cdot NI_{if1})]) \tag{5.34}$$

$$NI_{s2} = NI_{1s2} + n_2\{NI_{2s2} + \frac{n_1}{s}[NI_{3s2} + \frac{n_1}{s}(NI_{Loop,s2} + P_r \cdot NI_{if2})]\} \tag{5.35}$$

When considering the full vectorized version, with non-unit stride load and stores:

$$NI_v = NI_{1v} + \frac{n_1 n_2}{s} \cdot \left(NI_{2v1} + \left\lceil \frac{n_1}{VLMAX} \right\rceil NI_{Loop,v1}\right) + \frac{n_1}{s}\left[NI_{2v2} + \left\lceil \frac{n_1 n_2}{sVLMAX} \right\rceil NI_{Loop,v2}\right), \tag{5.36}$$

while instead, when only the loop comparing rows is vectorized:

$$NI_v = NI_{1v} + \tfrac{n_1 n_2}{s} \cdot \left(NI_{2v1} + \left\lceil \tfrac{n_1}{VLMAX} \right\rceil \cdot NI_{Loop,v1}\right) + n_2\left\{NI_{2v2} + \tfrac{n_1}{s}\left[NI_{3v} + \tfrac{n_1}{s}\left(NI_{Loop,v1} + P_r \cdot NI_{if2}\right)\right]\right\}. \tag{5.37}$$

In both cases:

$$\frac{NI_s}{NI_v} = \frac{NI_{s1} + NI_{s2}}{NI_v}. \tag{5.38}$$

From Fig. 5.20 it is clear that $n_2$ has a small impact on the ideal speedup, although very small values ($n_2 = 6$) produce noticeably slightly lower results already for small differences in terms of $n_2$ (e.g. $n_2 = 20$). The loss in terms of ideal speedup due to the use

[13] To avoid the use of the *memcpy* function in the scalar version, the additional flag "*-fno-tree-loop-distribute-patterns*" was added during compiling.

```
void maxpool2d_vec(int *a, int* b_t, int *b) {
size_t vl;
vint32m1_t va, vb, vc, val, vb_t;

stride_rows = 4*nl;
stride_rows_s = 4*nl/s;
b_saved = b; b_t_saved = b_t; a_saved = a;
for (int j=0;j<(nl*n2)/s;j++){
  for (int i=0;i<s-1; i++){
    columns = nl;
    b_t= b_t_saved+j*nl;
    a= a_saved+j*nl*m+(i+1)*nl;
    for (;vl = vsetvl_e32m1 (columns); columns -= vl) {
    if(i==0){
      a-=nl;
          vb_t = vle32_v_i32m1 (a);
          a+=nl;}
         else{
            vb_t = vle32_v_i32m1 (b_t);}
      va = vle32_v_i32m1 (a);
      vb_t = vmax_vv_i32m1 (va, vb_t);
      vse32_v_i32m1 (b_t, vb_t);
      a += vl;
  b_t += vl;}}}

b_t=b_t_saved;
for (int j=0;j<(nl)/s;j++){
  for (int i=0;i<s-1; i++){
    rows = n2*nl/s;
    b= b_saved+j;
    b_t= b_t_saved+j*s+i+1;
    for (;vl = vsetvl_e32m1 (rows); rows -= vl) {
      if(i==0){
      b_t-=1;
      vb = vlse32_v_i32m1 (b_t, stride_rows);
        b_t+=1;}
      else{
        vb = vlse32_v_i32m1 (b,stride_rows_m);}
      vb_t = vlse32_v_i32m1 (b_t, stride_rows);
      vb = vmax_vv_i32m1 (vb, vb_t);
      vsse32_v_i32m1 (b, stride_rows_s, vb);
      b_t += vl*nl;
      b += vl*nl/s;}}}}
```

Figure 5.18: C function implementing *maxpool2d_totalvec* with RVVE.

of *maxpool_halfvec* instead of *maxpool_totalvec* is very large, having the former an almost constant ideal speedup of around 2x, while the latter has an ideal speedup above 20 already for $n_1 \geq 24$ (14 layers out of 15 in Table 4.3).

Fig. 5.21 show the ideal speedup for layers with $s > 2$ with experimental data from Spike. The number of instructions in this case is highly dependendent on the optimization of the compiler for each value of $s$. Therefore an analytical model for $s > 2$ was not defined. The experimental data shows that *maxpool_halfvec* becomes better with higher

```
for (int j=0;j<(n1)/s;j++){
  for (int i=0;i<s-1; i++){
    rows = n2*n1/s;
    b= b_saved+j;
    b_t= b_t_saved+j*s+i+1;
    for (;vl = vsetvl_e32m1 (rows); rows -= vl) {
      if(i==0){
      b_t-=1;
      vb = vlse32_v_i32m1 (b_t, stride_rows);
        b_t+=1;}
      else{
        vb = vlse32_v_i32m1 (b,stride_rows_m);}
      vb_t = vlse32_v_i32m1 (b_t, stride_rows);
      vb = vmax_vv_i32m1 (vb, vb_t);
      vsse32_v_i32m1 (b, stride_rows_s, vb);
      b_t += vl*n1;
      b += vl*n1/s;}}}
```

Figure 5.19: Loop employed to replace the second loop in *maxpool_totalvec*, to obtain *maxpool2d_halfvec* without non-unit stride load and store.

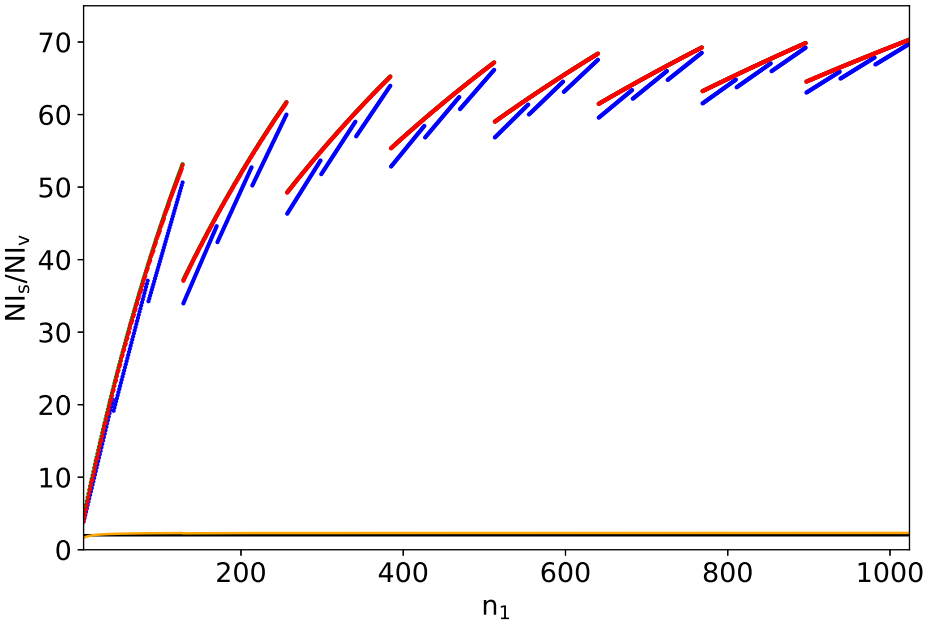


Figure 5.20: In red, ideal speedup ($NI_s/NI_v$) when employing vector instructions instead of scalar instructions for *maxpool_totalvec* for $n2 = n1$ and $m = 2$. In blue, for $n2 = 6$. In green for $n_2 = 1024$. In orange the ideal speedup of *maxpool_halfvec* for $n2 = n1$. In black an horizontal line for $NI_s/NI_v = 2$ is plotted for comparison.

$s$, going from around 25 times slower for $s = 2$ to 4 times slower for $s = 16$.

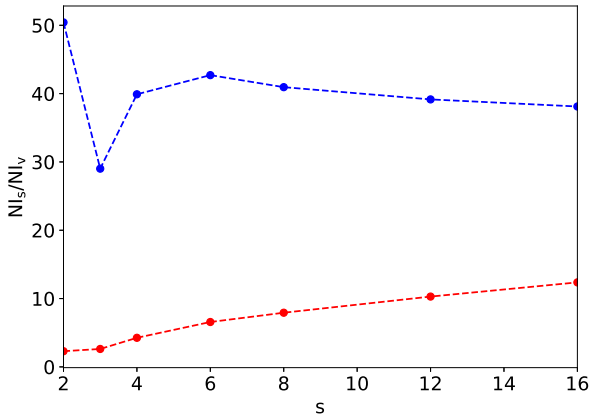


Figure 5.21: Ideal speedup ($NI_s/NI_v$) when employing vector instructions instead of scalar instructions for *maxpool2d_totalvec* (in blue) and for *maxpool2d_halfvec* for different value of $s$.



### ACTIVATION LAYERS

To implement *relu_vec* with the RVVE, a loop based on the instruction *vmax.vx* can be employed, i.e. the instruction comparing all the elements of a vector with the content of a scalar register and providing as an output a vector of the same size where each element is the original element of the input vector or the scalar [336]. As shown in Fig. 5.22 *relu_vec* can be easily obtained setting the value in the scalar register to 0. Also in this case a good degree of parallelizzation can be obtained, because the vector loop operates on ${n_1}^2 \cdot n_2$ vectors. The ideal speedup is:

```
void relu_vec(int *a, int *b, size_t n) {
  size_t vl;
  int c = 0;
  vint32m1_t va, vb;
  for (;vl = vsetvl_e32m1 (n);n -= vl) {
    vb = vle32_v_i32m1 (b);
    va = vmax_vx_i32m1 (vb, c);
    vse32_v_i32m1 (a, va);
    a += vl;
    b += vl;}}
```

Figure 5.22: C function implementing *relu* with RVVE.

$$\frac{NI_s}{NI_v} = \frac{NI_{1s} + n_2 \cdot [(NI_{2s} + n_1(NI_{3s} + n_1 \cdot NI_{Loop,s})]}{NI_{1v} + \left\lceil \frac{{n_1}^2 n_2}{VLMAX} \right\rceil \cdot \left(NI_{2v} + NI_{Loop,v}\right)}. \tag{5.39}$$

Similarly to *batchnorm*, the ideal speedup tends to $(NI_{loop,s}/NI_{loop,v}) \cdot VLMAX$.

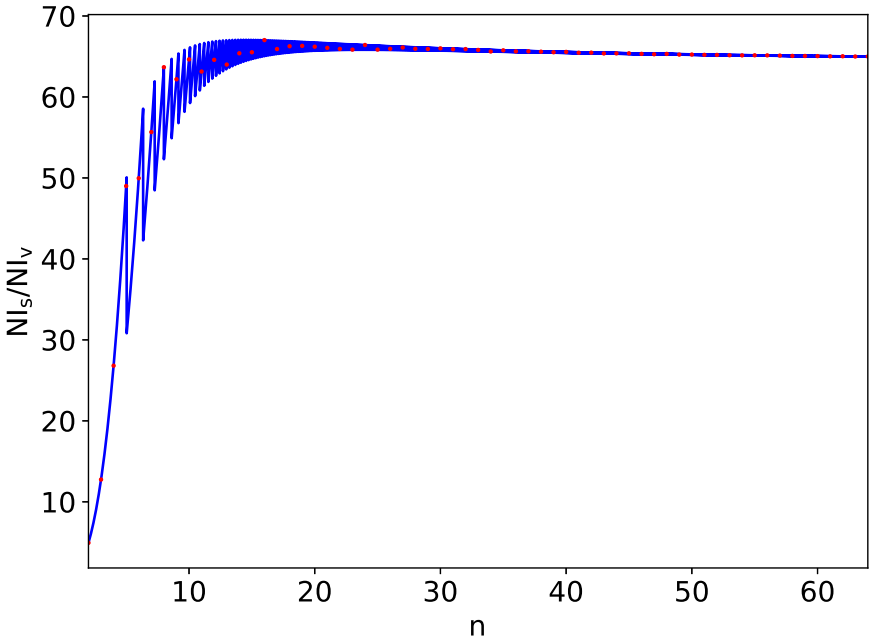


Figure 5.23: Ideal speedup $(NI_s/NI_v)$ when employing vector instructions instead of scalar instructions for *relu*. In red dots, performance for integer $n$ (Eq. 5.39). In blue, the associated continuous function.



### ADDITION LAYERS

Fig. 5.24 shows implementations for vector processors for 2 and 3 operands ($m$). Although this kernel has low OI, it can be heavily parallelized, as the addition of two 3D matrices of length $n_1 \times n_1 \times n_2$ can be seen as an addition between two unit-stride vectors of size ${n_1}^2 n_2$. According to data from Table 4.6 the minimum length of such contiguous vector is of 36864 elements. This very high value shows that increasing *VLMAX* will decrease the number of instructions required to execute *madd* even for very high values of *VLMAX*. The ratio between instructions of the scalar and vector implementation can be found with the equation below (valid for $m \geq 2$):

$$\frac{NI_s}{NI_v} = \frac{NI_{1fs}H(m-3) + (m-1)[NI_{2fs} + NI_{1s} + n_1 n_2 (NI_{2s} + n_1 NI_{Loop,s})]}{NI_{1fv}H(m-3) + (m-1)\left(NI_{2fv} + NI_{1v} + \left\lceil \frac{{n_1}^2 n_2}{VLMAX} \right\rceil NI_{Loop,v}\right)}, \tag{5.40}$$

where $H(m)$ is a unitary step function, i.e. $H(m) = 1$ for $m \geq 0$ and $H(m) = 0$ otherwise. It should be noted that in this case, two instructions are executed to initialize the vector loop and are also executed at each loop iteration. Therefore, they will be included both in $NI_{1v}$ and $NI_{Loop,v}$.

```
void  madd2_vec(int *a, int *b, int *c, size_t length) {
  size_t vl;
  vint32m1_t va, vb, vc;
  for (;vl = vsetvl_e32m1 (length);length -= vl) {
    va = vle32_v_i32m1 (a);
    vb = vle32_v_i32m1 (b);
    vc = vadd_vv_i32m1 (va, vb);
    vse32_v_i32m1 (c, vc);
    a += vl; b += vl; c += vl;}
}

void  madd3_vec(int *a, int *b, int *c, int *d, size_t length) {
  madd2_vec(a, b, d, length);
  madd2_vec(c, d, d, length);
}
```

Figure 5.24: C function employed for addition layers on the RISC-V vector processor. The function for $m = 2$ and $m = 3$ is shown. Those with $m > 3$ are composed in a similar way of the one with $m = 3$, cascading several calls to *madd2_vec*.



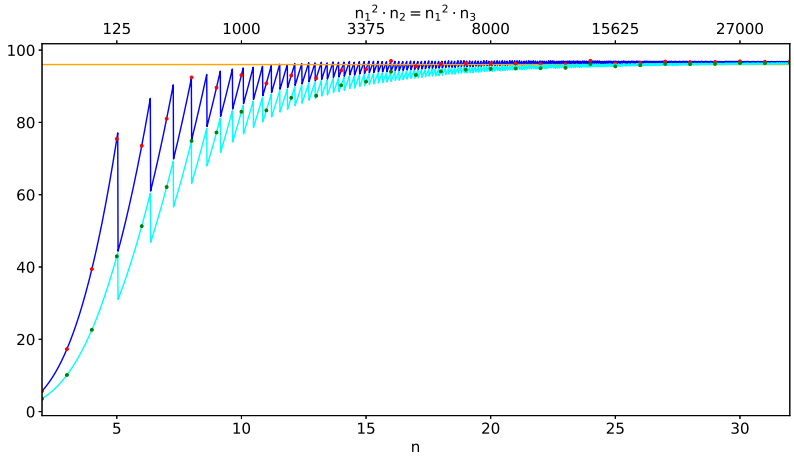


Figure 5.25: Ideal speedup ($NI_s/NI_v$) when employing vector instructions instead of scalar instructions for *matrixadd*. In blue and cyan, the continuous function associated with Eq. 5.40 for $m = 2$ and $m = 4$ respectively. In red and green, $NI_s/NI_v$ for integer values of $n$ for $m = 2$ and $m = 4$ respectively.

## 5.6. SUMMARY

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 a 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 microarchitecures with low ILP. Furthermore, it is possible to implement different approaches in terms of redundancy on the decoupled vector pipelines and on the scalar pipeline, reducing 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), while 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.



Furthermore, this chaper investigated the performance and dependability characteristics of the main memory, one of the most important trade-offs 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 missions. 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. on-line processing) may be unfeasible.

Finally, the impact of the use of instructions from the RVVE on the performance for kernels of CloudNet is investigated, showing that a large increase of performance is possible for all kernels with a minimal set of instructions.

# 6

# Design of a Vector Processor Based on the NOEL-V Platform

*Att våga är att förlora fotfästet en liten stund.*
*Att inte våga är att förlora sig själv.*
*To dare is to lose the foothold for a while,*
*not to dare is to lose oneself.*

Swedish proverb

*This chapter describes the work carried out to extend the NOEL-V platform to include Data-Level Parallelism (DLP) by implementing an integer subset of the RISC-V Vector Extension (RVVE). The performance and resource utilization efficiency of the resulting vector processor for different levels of DLP (i.e. number of lanes) has been compared to the baseline scalar processor on a Xilinx Kintex Ultrascale Field-Programmable Gate Array (FPGA), employing typical kernels for compute-intensive applications. The role of the memory subsystem has also been investigated, comparing the results obtained with both a low-latency and a high-latency main memory. The results show that the speedup due to the use of the vector pipeline increases with the number of lanes in the vector processor, speeding up the execution of the kernels of CloudNet by* 19.6× *(employing 128 lanes in the vector pipeline) with only* 3.9× *the resources of the baseline scalar processor.*

## 6.1. INTRODUCTION

As shown in Chapter 5, state-of-the-art processors for space embedded systems are based on simple microarchitectures, for instance capable of executing a maximum of one instruction per CC, with simple or no speculation (e.g. static branch prediction), and without DLP [30]. Recently, the HPP64 NOEL-V [353], a synthesizable VHDL model of a processor targeting space applications and based on the RISC-V ISA, has been released open source to the public together with the GRLIB VHDL IP Core Library[1]. As opposed to its predecessors, it has a dual-issue pipeline, i.e. it is capable of executing up to two instructions per CC [354]. Also, it has a two-level adaptive branch predictor with history buffer, a two-way branch target buffer, and a return-address-stack [354]. Furthermore, late-ALUs and a late branch unit are implemented to allow ALU and branch operations normally done in the execution stage to be deferred to a later stage. This reduces the latency required to execute some instructions once they are fetched, as well as allowing some dependent instructions to be issued simultaneously [354].

These new features have pushed the CoreMark for a single core [33] from around 2 CoreMark/MHz of previous generations (Sec. 1.4) to more than 4 CoreMark/MHz [353]. However, the limited amount of functional units (4 ALUs and one multiplier/divider) sets an upper bound on the achievable performance for workloads performing a large amount of operations, e.g. matrix multiplications.

As shown in Chapter 4, while this limitation was considered acceptable in previous generations of space processors, nowadays compute-intensive workloads are becoming of interest also in space applications.



### 6.1.1. GOAL, METHODOLOGY AND OUTLINE

Although vector processors are often seen as processors fit for power-hungry high-performance applications [136, 335, 355], this chapter aims to show that vector processors can be employed efficiently in highly constrained embedded systems as well. Space embedded systems are an extreme example of this type of systems.

In order to do this, in Sec. 6.1.2 the requirements for the vector processor prototype are formulated, according to the research carried out in previous chapters. Then, in Sec. 6.2 the design of the vector processor is described, together with the resource utilization on the FPGA and the effect of different main memory technologies. An integer subset of the RVVE was implemented in synthesizable VHDL and its effect on performance evaluated, comprising an immediate configuration instruction (*vsetvli*), a unit-stride load instruction (*vle32.v*), a unit-stride store instruction (*vse32.v*), a vector-vector addition instruction (*vadd.vv*), a whole vector register move instruction (*vmv1r.v*), a multiplication-accumulation on the addend (*vmacc.vx*) and a multiplication-addition overwriting the multiplicand (*vmadd.vx*). The use of integer instructions is common for implementations targeting applications where the precision of the floating-point formats is not needed and fixed-point formats can be employed, resulting in processors with small footprints for low-power applications [137, 302].

The performance of the highly optimized dual-issue scalar pipeline of the HPP64 (configured to support the RV64IM subset and referred to as *scalar* processor) will be

[1] https://www.gaisler.com/index.php/downloads/leongrlib

compared to the performance of an implementation where the scalar processor is extended with a simple and modular Vector Processing Unit (VPU)[2]. To have an estimation of the resources required (i.e. resource utilization) by configurations with different numbers of lanes, syntheses for the Xilinx Kintex Ultrascale (XCKU040) field-programmable gate array (FPGA) were run for different configurations. A template design of the GRLIB 2020.2 was employed as baseline prototype, removing the Ethernet controller to provide more resources for the processor and allow the prototype to reach 100 MHz of clock frequency. Although a vector processor is best suited for ASIC implementations and the final goal of this development is to have a rad-tol or rad-hard ASIC based on this design, the study of this prototype will provide insights on the resource utilization and prevalidation to minimize the design risk of the ASIC. Nevertheless, a rad-hard version of the Kintex Ultrascale has been proposed and the RISC-V vector processor may be employed on a similar platform.

The benchmarking in Sec. 6.3 is divided in two phases: in a first phase, two extreme cases in terms of OI are taken. This type of benchmark was carried out periodically during the design to track the increase of performance due to optimizations. In a second phase, the performance when running kernels from CloudNet is measured. In this case also an extended version of the prototype is considered to speedup some of the kernels not considered in the first phase of the benchmarks. In Sec. 6.4 the requirements defined in Sec. 6.1.2 are verified. In Sec. 6.5 the performance of the vector processor designed in this chapter are compared with other data-parallel RISC-V processors. Finally, in Sec. 6.6 the findings of this chapter are summarized.



### 6.1.2. REQUIREMENTS

The design of the vector extension of the NOEL-V processor was carried out during this PhD according to the requirements formulated below. Further works may change them (e.g. higher TRL) or add new ones (e.g. being able to operate on different element sizes).

**R1: Software support for the processor shall be available from developments of institutions outside the space industry.**

*Rationale: Space industry lacks the manpower and funding needed to solve complex problems like setting up and maintaining a complex and dependable toolchain for a complex processor for AI. A corollary of this requirement is that no custom non-standard extensions can be employed (see discussion in Sec. 2.1.3).*

**R2: The processor shall be configurable at HDL compile time to satisfy a large range (one order of magnitude) of peak performance..**

*Rationale: Space industry lacks the manpower and funding needed to have a different hardware design for each processing requirement.*

[2] This extended processor, comprising the baseline scalar processor and the vector extension, will be called *vector* processor.

**R3: The processor shall achieve a speedup of at least 8× compared to the scalar baseline for the kernels of CloudNet with less than 9× the resources of the baseline scalar processor.**

*Rationale: The most powerful space processor based on LEON is the GR740, a quad-core processor. In Chapter 3 a quad-core processor was estimated to have an area of 9× the one of a single core (included a 1 MiB L2C in the quad-core processor) and it can achieve up to 4× the performance of the baseline scalar processor. If this requirement is satisfied, then a single vector processing core is more than twice as fast as the state of the art when executing CloudNet, more than twice as efficient and will fit in dies of sizes and technologies comparable to state-of-the-art ASICs for space applications.*

**R4: The vector processor shall achieve the same maximum clock frequency as the baseline scalar processor.**

*Rationale: In this way the scalar pipeline can be used for workloads where the vector processor is slower, preserving the performance of the baseline processor.*

**R5: The vector processor shall achieve a TRL of 4.**

*Rationale: In order to build confidence around this innovative solution.*

**R6: The vector processor shall support integer instructions on 32-bit elements.**

*Rationale: In order to avoid large FPUs, operations will be performed on integer (see discussion in Sec. 5.3.2). This approach allows the execution of fixed-point algorithms. The choice of 32-bit elements is due to have a fair comparison with the baseline scalar processor, which doesn't have specific instructions to handle efficiently smaller element sizes.*

## 6.2. IMPLEMENTATION OF THE VECTOR PROCESSING UNIT

The vector processor is implemented as a decoupled vector pipeline that can be enabled at HDL compile time, to allow the use of the selective redundancy approach proposed in Sec.5.3.3, by simply replacing the code of the open source NOEL-V with the FT version from Cobham Gaisler (when it will be available).

Fig. 6.1 shows the complete vector processor, with a modified HPP64 interfaced with the VPU. The VPU comprises three elements: a sequencer, the lanes and a VLSU. The sequencer interfaces with the modified pipeline of the NOEL-V, receiving scalar parameters (e.g. base addresses of vectors) and vector instructions, while sending to the scalar pipeline the scalar results (when applicable). Each lane consists of a slice of the VRF, containing a single 32-bit element of the vector for each of the 32 vector registers defined by the RVVE, and the relative combinational paths to execute the implemented RVVE instructions. The *vmadd* and *vmacc* instructions are executed in two CCs (pipelining) to avoid penalties on the maximum frequency compared to the scalar processor when using large amounts of DLP required for DNNs (as shown analyzing DNNs in [288]). For

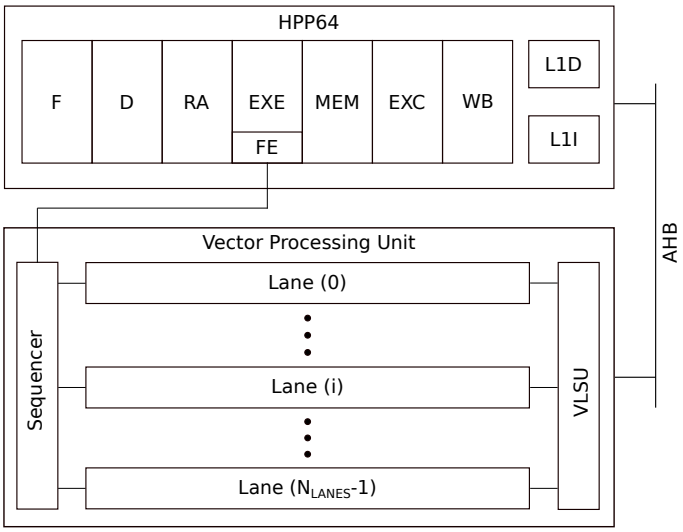


Figure 6.1: The HPP64 NOEL-V interfaced with the VPU. Fetch stage (F), Decode stage (D), Register Access stage, Execution stafe, Front End (FE), Memory stage (MEM), Exception stage (EXC) and Write Back (WB) stage are shown.



maximum flexibility of the prototype and quick design exploration, the VLSU uses a generic bus master from the GRLIB to read and write data on a 128-bit-wide AHB bus. Both the sequencer and the VLSU are designed to handle a configurable number of lanes $N_{Lanes}$ (i.e. *VLMAX*) that can be set at compile time with a *generic* in VHDL. For the convolutional layers of the DNN analyzed in Chapter 4, only 18.2% (24 out of 132) of the dimensions of the multiplied matrices are smaller than 128. For this reason, the focus in this chapter will be on improving performance of kernels with 128-element parallelism, as this seems a good tiling size to speed up a large majority of matrix multiplications in DNNs. Therefore, only consider configurations with $N_{Lanes} \leq 128$ will be considered, as configurations with a level of DLP larger than allowed by the algorithm do not provide any advantage (unless software solutions like batching are employed [186]).

Furthermore, the integration of the VPU required slight modifications to the scalar baseline HPP64 NOEL-V processor. The instruction Decode stage (D) was modified to access the required scalar registers for vector instruction reading or writing scalar values:

- *vsetvli* returns a scalar value to be stored in a scalar register.
- *vle32.v* and *vse32.v* require a scalar value as a base address.
- *vmadd.vx* and *vmacc.vx* require a scalar value as a coefficient.

In the Register Access stage (RA), where instructions are issued, only a vector instruction can be issued per time and no other instruction can be issued simultaneously (to avoid conflicts). Therefore, when vector instructions are issued, the dual-issue capability of the scalar pipeline is disabled. The Execution stage (EXE) was modified to include also a

VPU Front End (FE), as shown in Fig. 6.1. The other stages, i.e. Fetch stage (F), Memory stage (MEM), Exception stage (EXC) and WB stage remained unchanged.

### 6.2.1. RESOURCE UTILIZATION

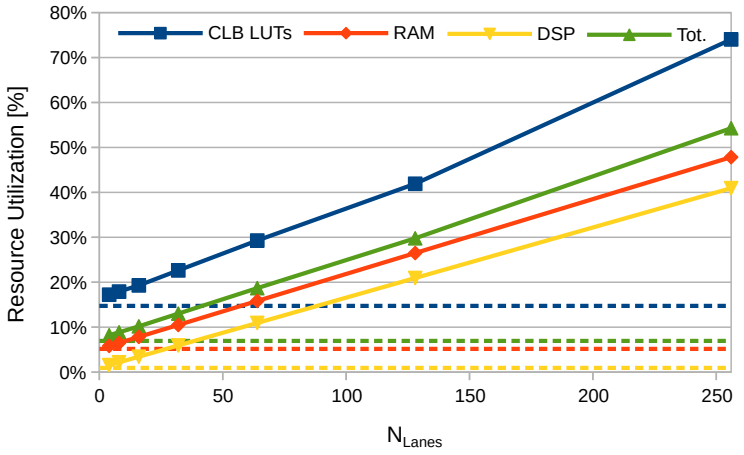




Figure 6.2: Resource utilization on the XCKU040 for the scalar processor (dashed lanes) and vector processors with 4, 8, 16, 32, 64, 128 and 256 lanes (solid lines with data points).

To have an estimation of the resources required (i.e. resource utilization) by configurations with different numbers of lanes, syntheses for the Xilinx Kintex Ultrascale (XCKU040) field-programmable gate array (FPGA) were run for different configurations. Resource utilization from implementation reports is shown in Fig. 6.2 for configurable logic blocks look-up tables (LUTs), block random-access memory (RAM) tiles and digital signal processing (DSP) blocks (of the respectively 242400, 600 and 1920 available in the selected FPGA). Furthermore, a total resource utilization (Tot.) has been calculated assuming that all the resources have the same importance (i.e. with an average of the three utilizations). The increase in terms of resource utilization is linear with the number of lanes, showing good scalability up to 128 lanes. For 256 lanes an increased effort by the tool is required during the implementation to meet the requirement on the clock frequency, resulting in a superlinear increase of required resources, especially in terms of LUTs. The vector processor with $N_{Lanes}$ = 128 requires 4.29x the resources of the baseline scalar processor. Furthermore, all configurations are able to reach a frequency target of 100 MHz (the same as the scalar processor), which will be used as clock frequency of the processors in the remainder of the chapter.

### 6.2.2. MEMORY SUBSYSTEM

When using the scalar pipeline, the data is cached in a 16 KiB L1 DC. This is not the case for the data used by the vector pipeline. On the other hand, also operations on vectors are sped up by the L1 IC in the HPP64. In order to investigate the role of the memory subsystem in the performance of scalar and vector processors, two different memory

configurations will be investigated:

- SRAM configuration: instructions and data are read from and written to a 4 MiB SRAM array on the AHB bus. This ensures a low latency of access from the processor. It is representative of memory hierarchies where tiling is taken care of (similarly to what done in [302]) or caching is implemented at L2.
- Synchronous Dynamic Random-Access Memory (SDRAM) configuration: instructions and data are read from and written to an external 2 GB Double Data Rate 4 (DDR4) SDRAM module. This is representative of how the vector processor would operate at the current development stage in real-world applications, where the large amount of data involved typically requires an external main memory with relatively high latency of access.

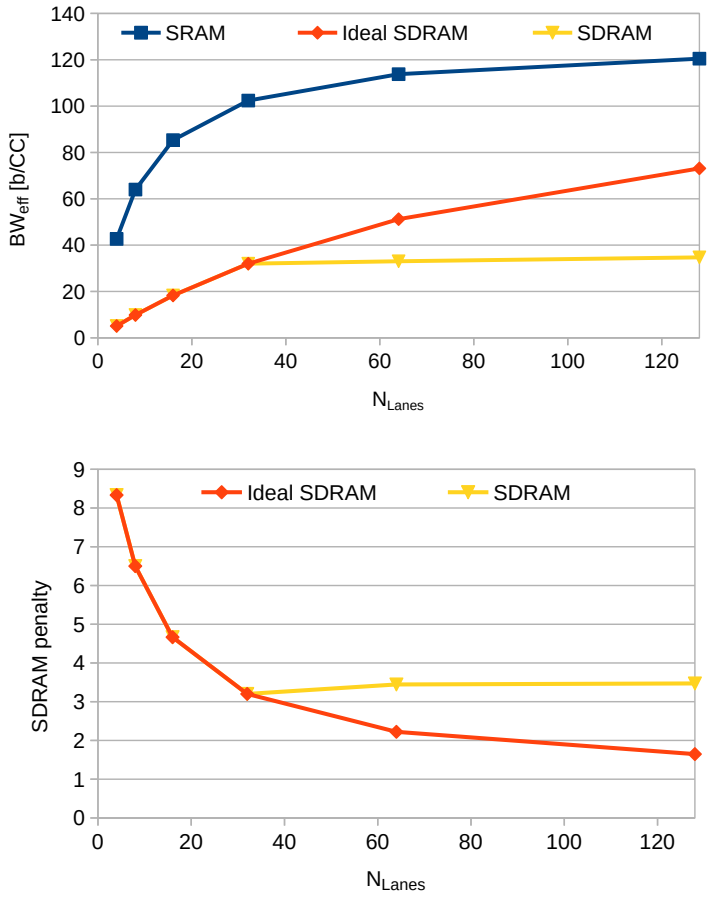


Figure 6.3: Top: Effective bandwidth. Bottom: SDRAM penalty in terms of $BW_{eff}$ for vector loads with different $N_{Lanes}$. Data points represent measured data ($N_{Lanes}$ = 4, 8, 16, 32, 64, 128).

To compare quantitatively the two memory configurations, the bandwidth of both the SRAM and SDRAM configurations are investigated when using the vector pipeline to read the first $N_{Lanes}$ elements of a vector, as shown in Fig. 6.3 (top). To investigate further the behavior of the SDRAM module, both an ideal model of the SDRAM (assuming that the SDRAM module is equivalent to the SRAM except for a longer latency) and real measurements from the SDRAM module on the KCU105 Evaluation Board by Xilinx [356] are considered. The CCs between the read request by the vector processor on the bus and the data available on the bus for the vector processor ($CC_{lat}$) are counted, with simulations for the SRAM configuration and with an on-chip logic analyzer provided with the GRLIB (LOGAN) for the SDRAM on the KCU105 board. This value was two CCs for the SRAM configuration and 24 CCs for the SDRAM configuration.

Assuming a single burst with an initial latency (SRAM and ideal SDRAM), this can be seen by the core as a lower effective bandwidth ($BW_{eff}$), given by:

$$BW_{eff} = \frac{D_{read}}{CC_{lat} + CC_{read}}, \tag{6.1}$$

where $D_{read}$ is the amount of data to be read and $CC_{read}$ is the number of CCs to read the data on the bus once it is available. For instance, considering a load involving all the lanes of a configuration with $N_{Lanes} = 4$ ($D_{read} = 128\ b$), where $CC_{read} = 1$ for both SRAM and SDRAM, $BW_{eff,SRAM} = 42.67\ b/CC$ and $BW_{eff,SDRAM} = 5.12\ b/CC$ (8.33x lower). Fig. 6.3 (bottom) shows that the penalty in terms of $BW_{eff}$ is reduced when the number of lanes is incremented, as vector processors use a single, longer access to mitigate the latency of memories. This effect smoothly saturates for the ideal SDRAM for larger $N_{Lanes}$ and the SDRAM penalty tends asymptotically to 1. For instance, $N_{Lanes} = 128$ is already enough to achieve a penalty of just 1.65. The increase of performance with the increase of $N_{Lanes}$ saturates abruptly instead after the maximum burst size of the SDRAM is reached in real measurements (in this case the maximum burst is $128B$, i.e. the bandwidth saturates for $N_{Lanes} > 32$) and more than one burst read access is required to read the first $N_{Lanes}$ elements of the vector. In this case $CC_{lat}$ is the sum of the latency of each read burst access (two burst reads for 64 lanes, four for 128 lanes). The slight increase of $BW_{eff}$ after $N_{Lanes} = 32$ is due to the fact that the following read burst accesses are in the same SDRAM page of the first one and for this reason $CC_{lat}$ is reduced to 21. In the following sections more complex SDRAM penalty plots will be proposed to investigate the effects of the two memory configurations for several benchmarks and hardware configurations.

## 6.3. BENCHMARKING

The roofline model [198] shows that increasing computational capabilities with DLP speeds up very effectively kernels with high $OI$. On the other hand, performance of kernels with low $OI$ are limited by memory bandwidth. Therefore, improving computational capability in this case provides little or no speedup.

In HPC, optimized Basic Linear Algebra Subroutines (BLAS) routines [357, 358] are typically available for each HPC platform, so that it is possible to map software to highly-optimized routines. One of the most popular BLAS routines is *gemm*, already introduced

```
void iaxpy (int32_t x[], int32_t y[], int32_t a, size_t N) {
  int i;
  for (i=0; i<N; i++) {
    y[i]=(a*x[i])+y[i];}}
```

Figure 6.4: C function employed for *iaxpy* on the scalar processor.

```
void iaxpy_rvv(int32_t *x, int32_t *y, int32_t a, size_t N) {
  size_t vl;
  vint32m1_t vx, vy;
  for (int n = N; vl = vsetvl_e32m1(n); n -= vl) {
    vx = vle32_v_i32m1 (x);
    vy = vle32_v_i32m1 (y);
    vy = vmacc_vx_i32m1 (vy, a, vx);
    vse32_v_i32m1 (y, vy);
    x += vl;
    y += vl;}}
```

Figure 6.5: C function employed for *iaxpy* on the vector processor.

in Sec. 4.4.1.

The reason behind the use of this routine in the benchmarking is that it achieves the peak performance of a given platform for large-enough matrices. This is due to the fact that its *OI* increases with the size of the matrices involved. For instance, assuming matrices composed of 32-bit integers (*igemm*), $\alpha = 1$, $\beta = 1$ and $n_1 = n_2 = n_3 = n$, the operational intensity is $OI = (2n+1)/16$ $OP/B$ ($n$ will be referred to as "problem size"). The C functions used for the scalar and vector processor are reported respectively in Fig. 4.5 (an integer version will be considered) and Fig. 5.10.

On the other hand, other BLAS kernels have low *OI* and even increasing the problem size will not make the workload compute-bound. A typical example is *axpy* ("ax plus y"), which implements the algorithm below:

$$y \leftarrow ax + y \tag{6.2}$$

where $a$ is a scalar, and $x$ and $y$ are vectors of size $n$. In this case, *OI* for 32-bit integers (*iaxpy*) is stuck at $1/6$ $OP/B$. The C functions used for the scalar and vector processor are reported respectively in Fig. 6.4 and 6.5. The choice of this kernel instead of $madd2$ is due to the fact that, in order to comply with Requirement **R4**, FMA operations have been implemented with two CCs instead of one.

### 6.3.1. RESULTS

The benchmarking was conducted exploring the scaling of performance both changing the number of lanes (while keeping the size of the problem constant) and changing the size of the problem (while keeping the number of lanes constant). The results are reported in the remainder of this section.

### SCALABILTY IN TERMS OF NUMBER OF LANES

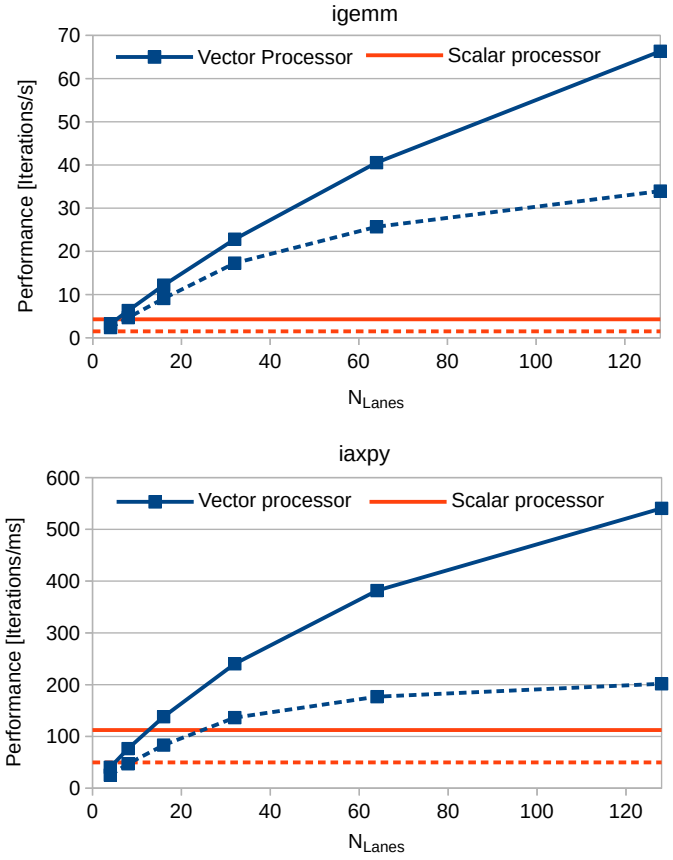


Figure 6.6: Performance (iterations/s) of the vector processor as a function of the number of lanes and of the scalar processor for *igemm* (top) and *iaxpy* (bottom) with $n = 128$ when using the AHB RAM (solid lines) and the external SDRAM (dashed lines). Data points represent measured data ($N_{Lanes} = 4, 8, 16, 32, 64, 128$).

Fig. 6.6 shows the performance of the scalar processor and vector processor with several $N_{Lanes} \leq n$ (in the case of $n = 128$) for both memory configurations. Thanks to the VLA nature of the RVVE, the same executable of the vector function with $n = 128$ was used for all the hardware configurations ranging from 4 to 128 lanes, exploiting the maximum level of DLP available on each configuration.

Fig. 6.7 shows that increasing the number of lanes in the vector processor increases the speedup compared to the scalar processor with the same memory configuration until $N_{Lanes} = n$ is reached, for both *igemm* and *iaxpy*. The maximum speedup achieved when employing the SRAM configuration is respectively around 15.6× and 4.8× the performance of the scalar processor. For the SDRAM configuration the maximum speedup is higher for *igemm* and lower for *iaxpy*, respectively around 23.0× and 4.1×.

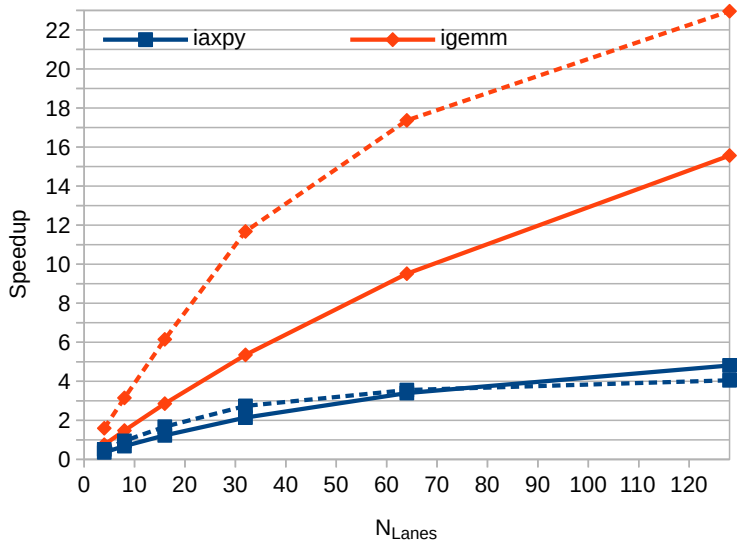


Figure 6.7: Speedup of the vector processor over the scalar processor varying the number of lanes for *iaxpy* and *igemm* with $n = 128$ when using the AHB RAM (solid lines) and the external SDRAM (dashed lines). Data points represent measured data ($N_{Lanes} = 4, 8, 16, 32, 64, 128$).



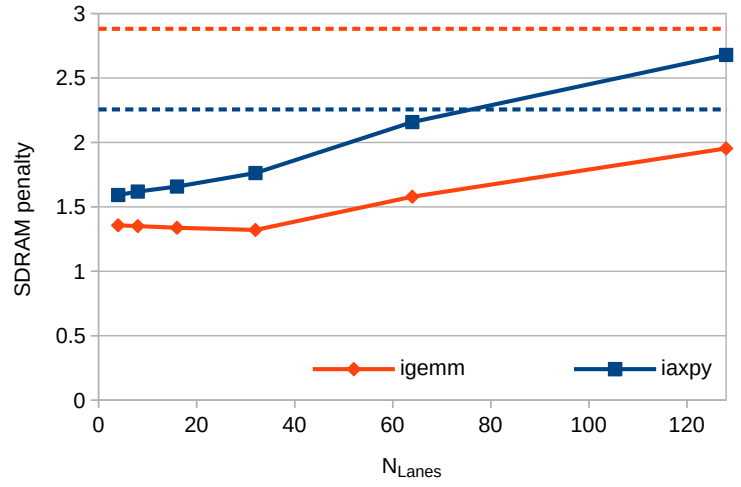


Figure 6.8: SDRAM penalty in terms of performance for the vector processor (solid lines) and for the scalar processor (dashed lines), varying the number of lanes for *iaxpy* and *igemm* with $n = 128$. Data points represent measured data ($N_{Lanes} = 4, 8, 16, 32, 64, 128$).

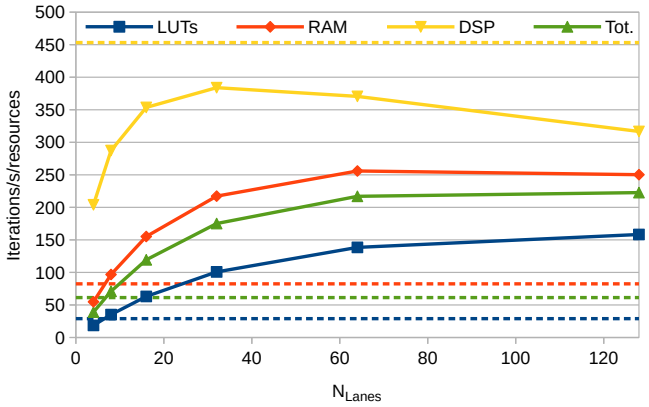


Figure 6.9: Utilization efficiency (performance expressed as number of times the function is executed per second) varying the number of lanes for *igemm* with $n = 128$ when using the SRAM configuration. Solid lines represent data for the vector processor, dashed lines for the scalar processor. Data points represent measured data ($N_{Lanes} = 4, 8, 16, 32, 64, 128$).



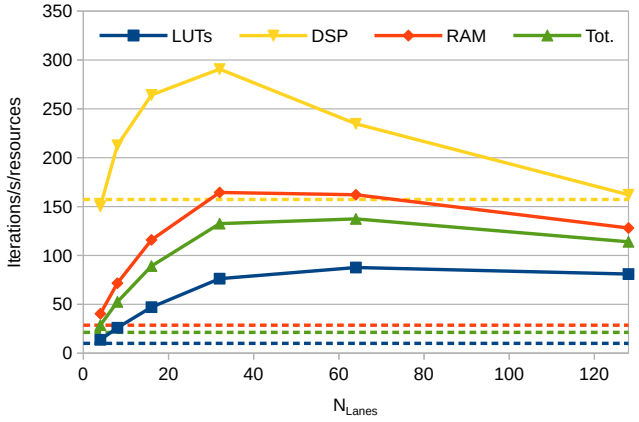


Figure 6.10: Utilization efficiency (performance expressed as number of times the function is executed per second) varying the number of lanes for *igemm* with $n = 128$ when using the SDRAM configuration. Solid lines represent data for the vector processor, dashed lines for the scalar processor. Data points represent measured data ($N_{Lanes} = 4, 8, 16, 32, 64, 128$).

It should be noted that there are some configurations for which the vector processor is slower than the scalar one. For the SRAM configuration this is the case when $N_{Lanes} = 4$ and $N_{Lanes} = 8$ in case of *iaxpy* (respectively 0.36x and 0.68x), while this is the case only for $N_{Lanes} = 4$ for *igemm* (0.75x). For the SDRAM configuration this happens only for *iaxpy* with $N_{Lanes} = 4$ (0.51x) and $N_{Lanes} = 8$ (0.95x). It should also be noted that while the speedup when running *igemm* on the vector processor with the SDRAM configuration is larger than the speedup of the vector processor for the SRAM configuration, this

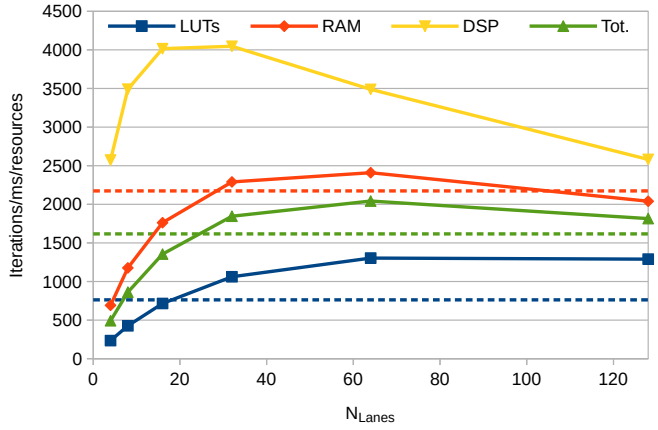


Figure 6.11: Utilization efficiency (performance expressed as number of times the function is executed per ms) varying the number of lanes for *iaxpy* with $n = 128$ when using the SRAM configuration. Solid lines represent data for the vector processor, dashed lines for the scalar processor. Data points represent measured data ($N_{Lanes} = 4, 8, 16, 32, 64, 128$). The DSP utilization efficiency for the scalar version is not reported because it is outside the represented range (11953 *iterations/ ms/ resources*)



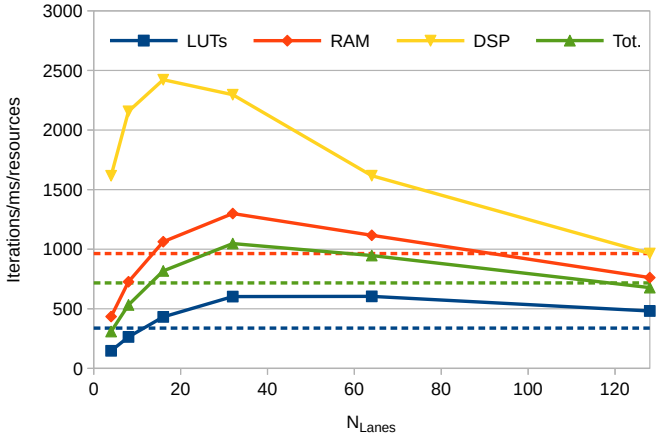


Figure 6.12: Utilization efficiency (performance expressed as number of times the function is executed per ms) varying the number of lanes for *iaxpy* with $n = 128$ when using the SDRAM configuration. Solid lines represent data for the vector processor, dashed lines for the scalar processor. Data points represent measured data ($N_{Lanes} = 4, 8, 16, 32, 64, 128$). The DSP utilization efficiency for the scalar version is not reported because it is outside the represented range (5299 $iterations/ms/resources$)

advantage is reduced for $N_{Lanes} > 32$. In the case of *iaxpy*, there is a similar trend and the relation is even inverted for $N_{Lanes} > 64$. Fig. 6.8 investigates further this aspect, showing that the penalty of using a SDRAM configuration, measured as the ratio between the

execution time with SDRAM and SRAM configuration, is lower for the uncached vector processor than for the cached scalar processor for *igemm* (even though it is clear that the need of several burst read accesses to the SDRAM module to read $N_{Lanes}$ elements of a vector increases the SDRAM penalty of the vector processor for $N_{Lanes} > 32$), while for *iaxpy* it becomes higher after $N_{Lanes} = 64$.

Given that the increase of performance with the number of lanes is typically sub-linear (Fig. 6.6) and that the resource utilization increases linearly instead (Fig. 6.2), there is an optimal number of lanes that maximizes the resource utilization efficiency in terms of performance/resources, where the required resources are expressed as a fraction of those available on the FPGA. Fig. 6.9 shows that for the SRAM configuration this happens for $N_{Lanes} = 32$ in case of *igemm* for the DSP blocks, while the total resource utilization efficiency increases up to 3.63x the one of the scalar processor for $N_{Lanes} = n = 128$. Finally, the *igemm* resource utilization efficiency for the vector processor is higher than the one for the scalar processor only if at least 8 lanes can be implemented, which means that adding the VPU increases resource efficiency only if a resource utilization of 1.28x the scalar processor can be tolerated (see Fig. 6.2). When considering the SDRAM configuration (Fig. 6.10), the efficiencies are smaller in terms of absolute value compared to the SRAM configuration and the total efficiency peaks for $N_{Lanes} = 64$. However, the peak reached has a higher relative value compared to the total utilization efficiency of the respective baseline (6.46x). Furthermore, for the SDRAM configuration all types of resource efficiency of the vector processors (except for DSP blocks and $N_{Lanes} = 4$) are higher compared to the scalar processor (providing higher total efficiency already for 1.18x the resources of the scalar processor).

The scalability of the total resource efficiency of the vector processor with the increase of DLP is, as expected by the roofline model, worse for *iaxpy*. In this case, considering the SRAM configuration, the vector processor has a total resource utilization efficiency better than the scalar processor for $N_{Lanes} > 16$, peaking at 1.26x for $N_{Lanes} = 64$ (Fig. 6.11). Furthermore, as in *iaxpy* the functional units are much less exploited by the software, even for the configuration with the peak DSP utilization efficiency ($N_{Lanes} = 32$), its value is 0.34x the one of the scalar processor. Similarly to *igemm*, the resource efficiency of *iaxpy* for the SDRAM configuration is lower as an absolute value compared to the SRAM configuration, but it is higher compared to the respective baseline. In this case, the vector processor has a total resource utilization efficiency better than the scalar processor already for $N_{Lanes} > 8$, its peak is at $N_{Lanes} = 32$ and it is 1.46x the one of the scalar processor. The following decrease makes the efficiency of the vector processor lower than the one of the scalar processor for $N_{Lanes} = 128$. Furthermore, the peak of the DSP efficiency is at $N_{Lanes} = 16$ and reaches 0.46x.

#### SCALABILITY IN TERMS OF THE PROBLEM SIZE

To show how increasing the size of the problem $n$ increases the advantage of the vector processor over the scalar processor, it is investigated how the two processors behave increasing $n$ above 32 for a vector processor with $N_{Lanes} = 32$. The results are shown in Fig. 6.13.

For both the SRAM and SDRAM configurations, the speedup increases with the *OI* (proportional to $n$) for *igemm* until $n = 128$ (11.70x), after which it saturates. Fig. 6.14

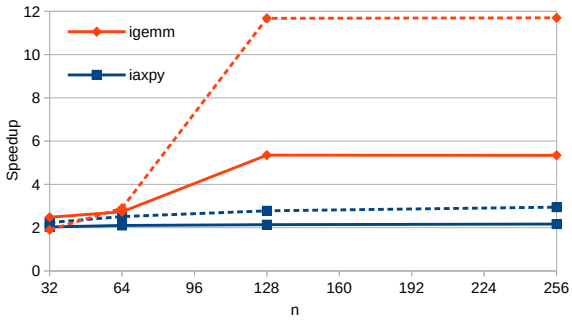


Figure 6.13: Speedup varying the size of the problem $n$ with a SRAM (solid lines) and SDRAM (dashed lines) configuration for both *iaxpy* and *igemm* with $N_{Lanes} = 32$. Data points represent measured data ($n = 32, 64, 128, 256$).

gives more insights on why this happens, considering the effective number of operations per CC ($\#OP/CC$) calculated as the number of operations ($2N^3 + N^2$ for *igemm* and $2N$ for *iaxpy*) for the SRAM configuration. For *igemm*, increasing the size of the problem slightly increases $\#OP/CC$ for the vector processor, as the overhead of calling the function is spread over more calculations. On the other hand, $\#OP/CC$ for the scalar processor decreases when $n$ is increased because the matrices get larger, leading to an increase of cache misses. The same can be deduced by the SDRAM penalty shown in Fig. 6.15, which shows that in the case of *igemm* the SDRAM penalty increases for the scalar processor when going from $n = 32$ to $n = 128$, meaning that the L1D is less capable of masking the latency of the main memory as matrices get larger. The SDRAM penalty is instead roughly constant for the vector processor. It should also be noted that large problem sizes favour the vector processor even for *iaxpy*, increasing the gap in terms of SDRAM penalty as the problem size increases (even if the SDRAM penalty of the scalar processor does not increase).

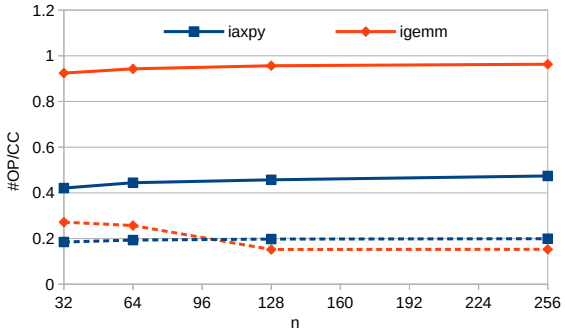


Figure 6.14: Number of operations per CC varying the size of the problem $n$ for both *iaxpy* and *igemm* with $N_{Lanes} = 32$ when using the SRAM configuration. Solid lines represent data for the vector processor, dashed lines for the scalar processor. Data points represent measured data ($n = 32, 64, 128, 256$).

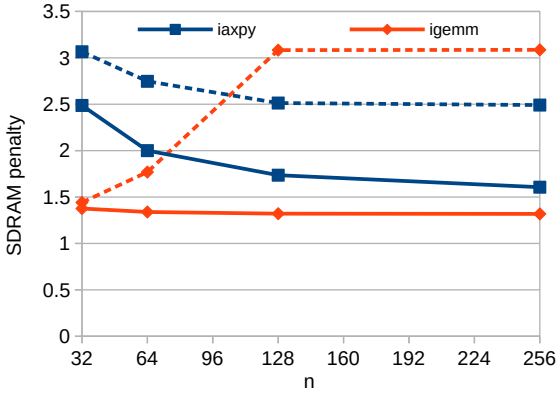


Figure 6.15: SDRAM penalty in terms of performance, varying the size of the problem $n$ for both *iaxpy* and *igemm* with $N_{Lanes} = 32$. Solid lines represent data for the vector processor, dashed lines for the scalar processor. Data points represent measured data ($n = 32, 64, 128, 256$).

### 6.3.2. PERFORMANCE IMPROVEMENTS FOR CLOUDNET KERNELS

In this section, the kernels in Sec. 4.4.1 were run on Spike, with a similar setup as in Sec. 5.5, and on the hardware prototype. In Sec. 5.5, the ideal speedup (i.e. the ratio between number of instructions of the scalar implementation and of the vector implementation) for the kernels in CloudNet was investigated. Although the analysis in Chapter 5 was able to prove that considerable performance improvements are possible employing vector instructions, the speedup of the prototype will differ, as (from Eq. 1.1) it can be written as :

$$Speedup = \frac{NI_s \, T_{clk_s} \, CPI_s}{NI_v \, T_{clk_v} \, CPI_v}, \tag{6.3}$$

where variables with the subscript $s$ refer to the scalar baseline processor and the ones with the subscript $v$ refer to the vector processor. Given that the scalar and the vector prototype run at the same clock frequency, and that in Chapter 5 the ideal speedup was defined as $Ideal\,speedup = NI_s / NI_v$, Eq. 6.3 can be rewritten as:

$$Speedup = Ideal\,Speedup \cdot \frac{CPI_s}{CPI_v}. \tag{6.4}$$

From Eq. 6.4, it is clear that in this case the actual speedup of the hardware will be the Ideal Speedup divided by a $CPI_v/CPI_s$ factor. This factor will be estimated experimentally dividing the Speedup measured by the Ideal Speedup. This analysis will show which kernels require more optimizations in further works and which fraction of the ideal speedup is actually achievable by the prototype. Two configurations will be considered in this section:

1. Minimal configuration: this configuration implements only the instructions in Sec. 6.1.1. Comparing this set of instruction with the vector kernels in Sec. 5.5, it

is clear that this configuration will enable a speedup only for convolutional layers, concatenation layers and addition layers. For the other layers, scalar implementations will be employed (i.e., for these layers, *Speedup* = 1).

2. Extended configuration: this configuration implements additional RVVE instructions compared to the ones described in Sec. 6.1.1, in order to speedup all the layers in CloudNet with the vector kernels in Sec. 5.5. The additional instructions include *vmax.vv* to compare vectors in *maxpool2d*[3], *vmax.vx* to compare matrix elements with 0 in *relu*, and *vmul.vx* and *vadd.vx* to speedup batch normalization layers.

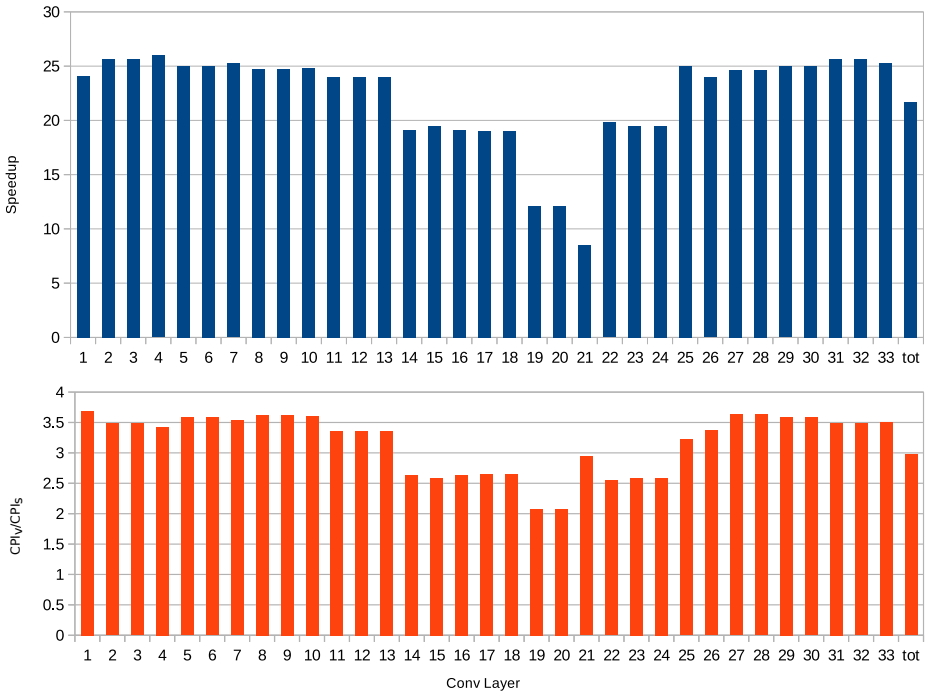


Figure 6.16: Speedup (above) and $CPI_v/CPI_s$ (below) for convolutional layers in CloudNet.

### CONVOLUTIONAL LAYERS

As shown in Fig. 6.16, Convolutional layers achieve the highest speedup among all the layers (the lowest value is 8.56, the average is 21.68, with a peak of 25.97), with a low $CPI_V/CPI_S$ ratio (an average of 2.97, only subsampling layers achieve lower values), meaning that the prototype is well optimized for this type of layers. The value of the speedup is mainly determined by the size of $n_3$ (values of $n3$ for each convolutional layer

[3] Even in this case, non-unit stride load and stores were not implemented. Therefore, the vector kernel employed in this section to speedup subsampling layers is *maxpool2d_halfvec*.

are reported in Tab. 4.1): layers with $n_3 \geq 576$ have a speedup ranging from 23.96 to 25.99, layers with $n_3 = 144$ have a speedup ranging from 18.98 to 19.49 and layers with $n_3 = 36$ have a speedup ranging from 8.56 to 12.13.

### CONCATENATE LAYERS

As shown in Fig. 6.17, concatenation Layers achieve a relatively low speedup (an average of 4.04, only subsampling layers have a lower average). Both speedups and $CPI_v/CPI_s$ are roughly the same for all layers, meaning that all the layers operate in the 'saturation' zone, i.e. matrices are large enough to optimize performance, and the behavior of the prototype has a similar behavior as the one from Spike shown in Fig. 5.14, although scaled by a factor of around 5.

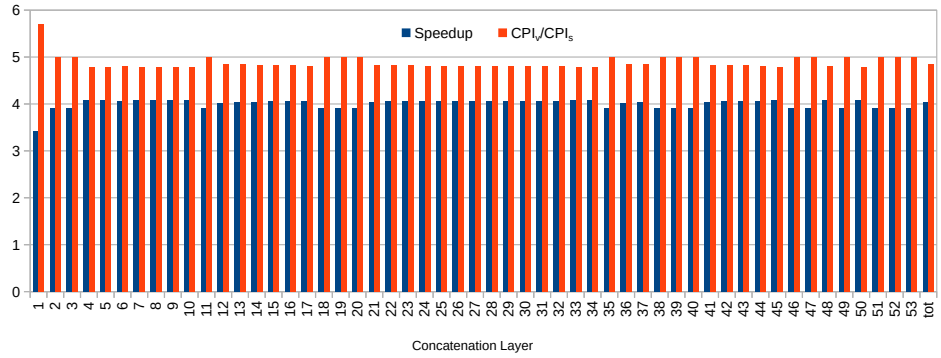


Figure 6.17: Speedup and $CPI_v/CPI_s$ for concatenation layers in CloudNet.

### BATCH NORMALIZATION LAYER

As shown in Fig. 6.18, the speedup is mainly determined by the parallelization factor ${n_1}^2 n_2$. Layers with ${n_1}^2 n_2 \geq 576$ have a speedup between 7 and 8, layers with ${n_1}^2 n_2 = 144$ have a speedup between 5 and 6, layers with ${n_1}^2 n_2 = 36$ have a speedup of ranging from 3.2 to 3.3. Layer 15 confirms that for small matrices there is a large ripple in the ideal speedup, as shown in Fig. 5.17, although the same behavior is not present in the speedup of the prototype.

### SUBSAMPLING LAYERS

As shown in Fig. 6.19, the speedup for subsampling layers has a large variation depending on the layer (the ratio between the minimum and the maximum speedup is 2.7). This is due to the fact that layers have different subsampling factors $s$, and that the speedup increases with the increase of $s$. Layers with $s = 2$ have a speedup ranging from roughly 1 to 2, layers with $s = 4$ (Layer 7 , 11 and 14) have a speedup between 2 and 3, layers with $s = 8$ (Layer 8 and 12). However, there is a saturation for $s = 16$ (Layer 9), where similar values of layers with $s = 8$ are achieved. The average speedup is similar to some of the layers with $s = 4$ (Layer 11 and 14). From Layer 7, 8 and 9 is clear that increasing $s$ also increases $CPI_v/CPI_s$, meaning that the increase of speedup with $s$ is less steep than the one shown in Fig. 5.21 for *maxpool2d_halfvec*.

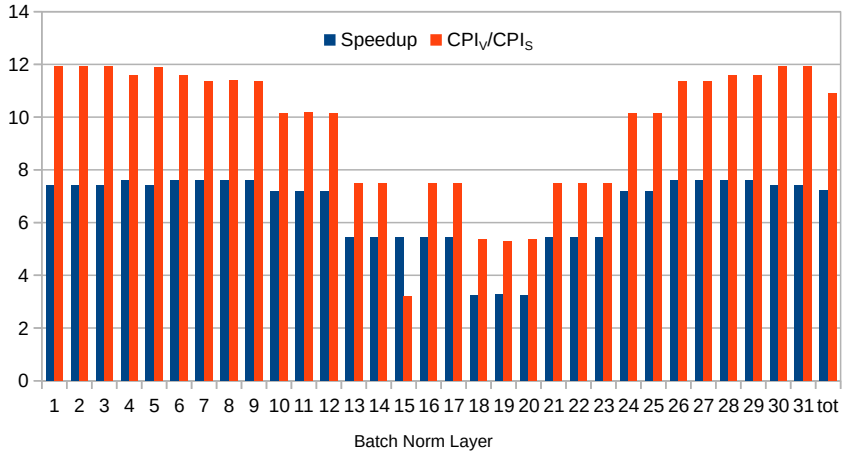


Figure 6.18: Speedup and $CPI_v/CPI_s$ for Batch Normalization layers in CloudNet.

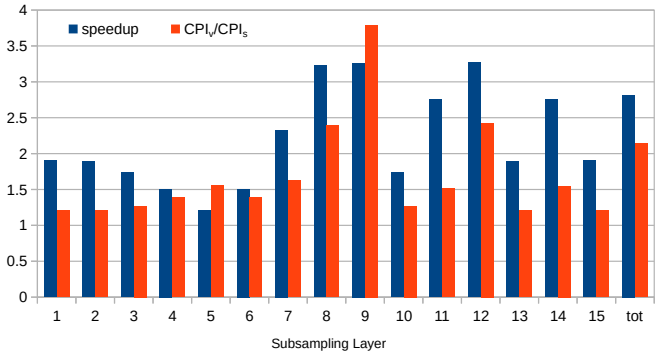


Figure 6.19: Speedup and $CPI_v/CPI_s$ for subsampling layers in CloudNet.



### ACTIVATION LAYERS

As shown in Fig. 6.20, Activation layers have a roughly constant speedup and $CPI_v/CPI_s$ (similarly to concatenation layers, although both values are higher than those of the latter). Also in this case, this indicates that the prototype has a similar behavior to the one found in Spike and reported in Fig. 5.23 and that the matrices in CloudNet are large enough to achieve the saturation part of Fig. 5.23.

### ADDITION LAYERS

As shown in Fig. 6.21, all the addition layers have a speedup value of around 6. This value is roughly independent from the number of matrices $m$ (this is the case for relatively large matrices in Fig. 5.25), while the speedup slightly increases with $n_2$: 6.29 for Layer 6 ($n_2 = 1204$) against 5.93 for Layer 1 ($n_2 = 32$).

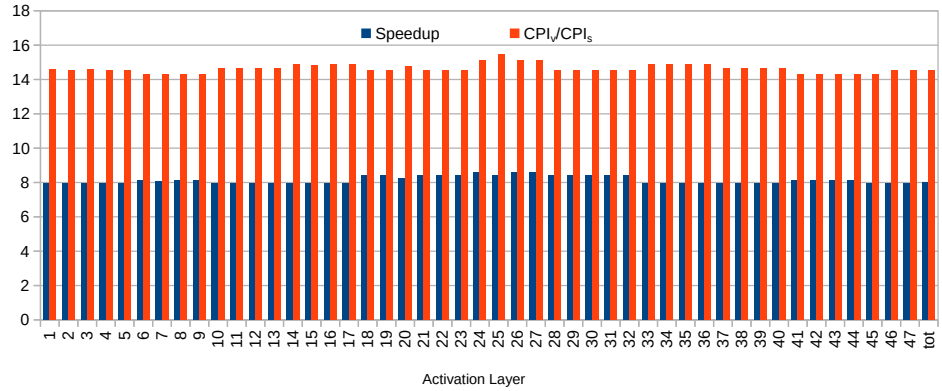


Figure 6.20: Speedup and $CPI_v/CPI_s$ for activation layers in CloudNet.

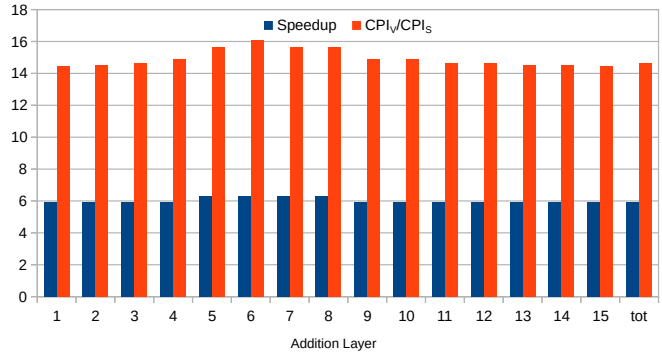


Figure 6.21: Speedup and $CPI_v/CPI_s$ for concatenation layers in CloudNet.



### OVERALL SPEEDUP

A total speedup for both configurations keeping into account all the kernels in CloudNet was calculated summing up all the execution times of the layers implemented with the scalar kernel and the ones of the kernels implemented with the scalar kernel. As most of the execution time is taken by convolution layers, the CloudNet speedup of the minimal configuration (19.6) is close to the one of the complete configuration (20.7), both values close to the total speedup for convolutional layer as shown in Fig. 6.16 (i.e. 21.7). The $CPI_V/CPI_S$ is respectively 1.97 and 2.91 instead. The breakdown for the minimal configuration, shown in Fig. 6.22, is similar to the preliminary one reported in Fig. 4.3 (obtained executing the DNN on a quad-core Intel i7-6600U), with two main differences:

1. In Fig. 6.22 the percentage of time spent in convolution layers is larger than in Fig. 4.3. This is probably due to better mapping to BLAS function on the Intel i7-6600U than on the vector prototype.

2. Although the order of the type of layers from the most time-consuming to the less

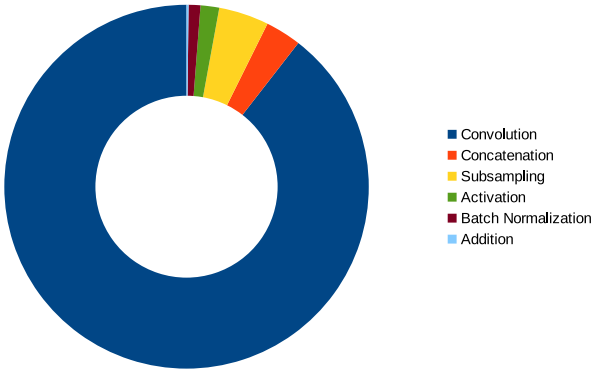


Figure 6.22: Breakdown of execution times for each layer type when executing CloudNet for the minimal configuration prototype.

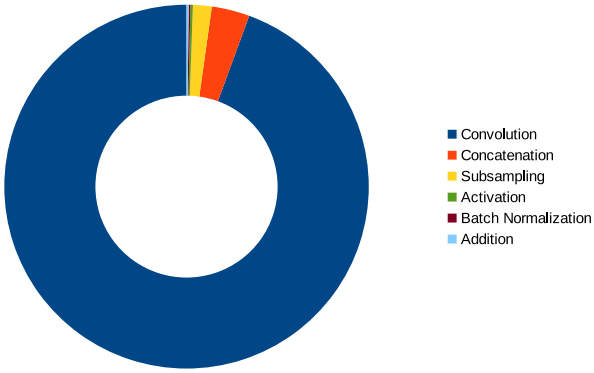


Figure 6.23: Breakdown of execution times for each layer type when executing CloudNet for the extended configuration prototype.



time consuming is similar, the relative importance of activation layers and batch normalization layers is swapped.

The breakdown for the extended configuration is instead reported in Fig. 6.23. In this case the percentage of time spent for convolution layers increases slightly.

## 6.4. REQUIREMENTS VERIFICATION

All the requirements defined in Sec. 6.1.2 have been satisfied, as shown below:

**R1: Software support for the processor shall be available from developments of institutions outside the space industry.**

*Verification: It was possible to compile the code with a patched version of the GCC compiler from the RISC-V foundation. Support in the future will be even more straightforward*

*as it will be integrated in the standard GCC.*

**R2: The processor shall be configurable at HDL compile time to satisfy a large range (one order of magnitude) of peak performance.**

*Verification: The peak performance from 1.60x to 22.96x the performance of the baseline scalar processor (Fig 6.7, igemm with n = 128).*

**R3: The processor shall achieve a speedup of at least 8× compared to the scalar baseline for the kernels of CloudNet with less than 9× the resources of the baseline scalar processor.**

*Verification: The minimal configuration with 128 lanes achieves 19.6x the performance of the baseline scalar processor when executing CloudNet, with 4.29x the resources of the baseline scalar processor.*

**R4: The vector processor shall achieve the same maximum clock frequency as the baseline scalar processor.**

*Verification: All the hardware prototypes achieve 100 MHz as maximum clock frequency on the selected hardware platform, the same as the one of the baseline scalar processor.*

**R5: The vector processor shall achieve a TRL of 4.**

*Verification: The hardware prototype was employed to run and measure performance of all the vector kernels of CloudNet, i.e. the hardware prototype was validated in a laboratory.*

**R6: The vector processor shall support integer instructions on 32-bit elements.**

*Verification: All the selected instruction have been verified to work on vector of 32-bit randomized values.*

## 6.5. RELATED WORK

Although the RVVE is still in the process of being standardized, it plays such a crucial role in state-of-the-art terrestrial applications that already several developments implementing the RVVE are described in literature. One of the most mature implementations described in literature is Xuantie-910 from Alibaba, a 16-core RISC-V processor supporting the RVVE [136]. Its FPGA prototypes are employed in data servers and a 12-nm ASIC clocked up to 2.5 GHz has been taped out. However, this processor clearly targets servers and HPC applications, which have totally different constraints compared to space embedded systems. Also a European consortium [359] is working on a RISC-V Vector accelerator for HPC applications and an ASIC has been recently taped out. Other companies like SiFive have announced commercial cores supporting the RVVE [360].

### 6.5.1. COMPARISON WITH OTHER DLP PROCESSORS

The microarchitecture described in this chapter (called 'vector NOEL-V processor' in this section) can be seen as a simplified version of Ara [19], a RISC-V vector processor based on the scalar Ariane [59] and implementing the RVVE with vector lanes. However, Ara aims at implementing the complete RVVE (excluding fixed-point and vector atomics [361]), while in this chapter it was decided to implement only a reduced integer subset, allowing us to reach massive DLP on a single core with limited resources. For this reason (especially because of floating point units), the scaling of the resources required by Ara with the number of lanes is worse: for instance in this chapter the 16-lane version uses just 24.1% resources more compared to the 4-lane version, while for Ara the area increases by 212.6%. On the other hand, Ara proves that the complete RVVE, although very interesting for HPC applications, is overdimensioned for resource-constrained embedded applications.

The approach followed in this chapter is not the only possible way of implementing the RVVE. For example, in [362] Vicuna is described and benchmarked. Vicuna uses dedicated execution units for different instruction types that process several elements at once instead of vector lanes [362]. Also in that case, only integer and fixed-point instructions are implemented. Although a direct comparison with the implementation described in this chapter is not possible (because Vicuna operates on 8-bit vector elements instead of 32-bit vector elements), observations can be made. The wider memory interface employed in this chapter (128 bits) compared to Vicuna (32 bits) allows for faster execution of memory-bound algorithms. In case of *iaxpy* with $n = 65536$ on 8-bit integer data, Vicuna requires 524.9 µs for its execution. Multiplying by 4 to keep into account the higher memory traffic required, it can be estimated a required time of 2.1 ms for 32-bit elements. The 256-lane implementation described in this chapter executes $n = 65536$ *iaxpy* in 678.8 µs. Furthermore, the modular approach followed in the design of the vector processor in this chapter and the subset of chosen instructions does not cause penalties on the maximum frequency (100 MHz) up to 256 lanes of 32 bits each (8192-bit datapath), while the largest version of Vicuna (1024-bit multiplier) has a 20% penalty over the smallest (32-bit multiplier). A similar penalty would be even larger for execution units operating on 32-bit data with the same DLP (4096-bit multiplier). However, an 8-bit 256×256 *igemm* executes in 8.3 ms on Vicuna. Assuming that it would be possible to implement a 32-bit version of Vicuna without any penalties on the frequency, an equivalent time of 33.3 ms can be estimated for 32-bit elements (x4). This is better than the 256 lanes implementation described in this chapter (81 ms), meaning that a 32-bit version of Vicuna would be faster than the vector NOEL-V processor, if it achieves a clock frequency higher than 33 MHz. A vector processor similar to Vicuna is described in [363]. In this case a 120×120 matrix multiplication executes on this processor capable of executing 8 OP/CC on 8-bit data in 3.0 MCC, equivalent to 12.0 MCC for 32-bit elements. The vector NOEL-V processor with the same potential number of OP/CC (8 lanes) executes a 120×120 *igemm* on 32-bit elements in 13.1 MCC. Although using a different FPGA, the implementation in [363] achieves only 50 MHz, confirming that that approach has worse scalability in terms of maximum clock frequency with the level of DLP implemented compared to the vector NOEL-V processor. The works [362] and [363] show that operating on 8-bit elements is an interesting solution to speed up

calculations and reduce memory traffic when the precision of 32 bits is not needed. Data from [364] shows that 8-bit is usually the smallest factor that allows an acceptable loss of accuracy for DNNs. Another possible approach is the one of Klessydra T13 [365], which is based on a non-standard custom RISC-V vector extension. This extension is comparable to the minimal subset implemented in this chapter (as also in this case the focus is on resource-constrained embedded applications and for instance floating point operations are not implemented), but it is centered around a scratchpad memory instead of a VRF. Furthermore, the Klessydra platform is based on a interleaved multithreated microarchitecture, alternating instructions belonging to different execution threads in the stages of a single-issue in-order pipeline [365]. The data in [365] shows that the single-core implementation using the maximum DLP available (8 elements) executes a 64×64 matrix multiplication on 32-bit values in 484.4 kCC for homogeneous workloads and 414.4 kCC for heterogeneous workloads[4], while the vector NOEL-V processor with 64 lanes runs the 64×64 *igemm* in 314.2 *kCC*.

The vector NOEL-V processor described in this chapter can be compared also with RISC-V implementations using non-standard packed SIMD extensions, like RI5CY [137]. This 32-bit core implements custom packed SIMD instructions to operate on four 8-bit elements or two 16-bit elements with a single instruction [137]. Furthermore, the core implements other instructions to speed up matrix multiplications and other kernels composed by loops of computations (e.g. hardware loops and post-increment addressing modes) [137]. Data for RI5CY in [365] shows that for a 64×64 matrix multiplications the vector NOEL-V processor with 64 lanes is around 4.3x faster than RI5CY operating on 32-bit values (314.3 kCC against 1.4 MCC). This means that the vector extended NOEL-V with 64-lane implementation in this case is roughly as computationally capable as four RI5CY cores and therefore the 8 cores in PULP can be potentially outperformed with a dual-core implementation of the NOEL-V vector processor. However, for smaller matrix multiplications this advantage is reduced. For a 16-bit 32×32 matrix multiplication in [189] a single core RI5CY takes 41.9 kCC, while for the vector NOEL-V 32-lane implementation it takes 70.9 kCC. Even adjusting for the different data element size (x2), the vector NOEL-V processor would be just 1.18x faster. An approach to improve the performance of the vector NOEL-V processor for small matrices is to implement the *vmadd* and *vmacc* instructions in a single clock cycle. This would have little or no impact for a small number of lanes in terms of maximum clock frequency, roughly halving the execution time.

### 6.5.2. BENCHMARKING METHODOLOGY

Considering the methodology employed in the benchmarking, the related works discussed in Sec. 6.5 do not investigate the effects of a fast and a slow memory on the vector processor ([362] provides performance only for a SRAM), some of them ([19] and [365]) not even providing details on the memory used. On the other hand, in this chapter it is proven how cacheless vector processors have an increased advantage over cached scalar processors when dealing with relatively slow, high-latency memories (Fig. 6.7).

[4] Given the interleaved multithreated microarchitecture of the Klessydra cores, data from [365] is reported both in the case the threads are based on the same algorithm operating on different data (homogeneous workload) and in the case where different algorithms are executed as different threads (heterogeneous workload).

## 6.6. SUMMARY

State-of-the-art processors for space embedded systems are based on simple microarchitectures, because of the small footprint required and low power available. In this chapter the implementation of DLP to enable compute-intensive workloads in next-generation space embedded systems with limited resources was described, and benchmarked with relevant software from real-world DNNs for space applications. Several configurations were considered, with high- and low-latency main memory (based respectively on a SDRAM and SRAM memory).

The design meets requirements formulated from the preliminary design in previous chapters, proving that vector processors are fit to speedup the execution of DNNs on space processors. The resources required for the hardware implementation of the selected RVVE subset increase roughly linearly with the number of lanes (i.e. its computational capabilities), with no noticeable penalty in the maximum clock frequency achievable compared to the baseline scalar processor up to 128 lanes. Given a certain size of the problem, the hardware implementation of the selected integer RVVE subset shows good scalability with the number of lanes, especially for compute-intensive workloads (*igemm*), having resource utilization efficiency above the one of the scalar processor for all configurations with more than four lanes, achieving up to 23.0x the performance of the scalar processor with only 4.3× the resources of the scalar prototype, and increasing resource utilization efficiency up to 6.5× the one of the scalar processor. Furthermore, for a fixed configuration, the performance of the vector processor scales better than the performance of the scalar processor with the increase of the problem size. The number of lanes at which the peak resource utilization efficiency happens depend on the latency of the main memory. For *igemm* the peak for the SRAM configuration happens for $N_{Lanes} = 32$, while for the SDRAM configuration at $N_{Lanes} = 64$. Although the resource utilization efficiency of the SDRAM configuration is smaller in terms of absolute value compared to the SRAM configuration, the peak reached has a higher relative value compared to the total utilization efficiency of the respective baseline (6.46×). Finally, the proposed vector processor is capable of achieving a speedup of 19.6× over the scalar baseline when executing a DNN for cloud detection, just employing a minimal subset of the RVVE.

Although this work shows that applying DLP in next-generation processors is feasible and effective for certain workloads, further work is needed to investigate whether a L1 vector data cache can speed up the execution for relatively small number of lanes or small problem size, whether other subsets of the RVVE are fit for implementations with small footprint, and which RVVE instructions not implemented yet are detrimental to the scaling in terms of number of lanes when considering resource efficiency and maximum achievable frequency. Finally, also the implementation of operations on smaller vector elements is an interesting subject to investigate to increase performance when lower precision is acceptable.

# 7

# CONCLUSION

*Il lavoro dovrebbe essere una grande gioia ed è ancora per molti tormento, tormento di non averlo, tormento di fare un lavoro che non serva, non giovi a un nobile scopo.*
*Work should bring joy, while for many people it still brings heartache. Heartache of not having one, heartache of having a useless one, without a noble goal.*

Adriano Olivetti, 1955

*This chapter summarizes the results of the research carried out in this PhD project. The Research Questions formulated in Chapter 1 are answered and the introduced innovations highlighted. Finally, a set of activities are suggested for future research on RISC-V processors for space applications, along with a set of recommendations to be followed during future activities.*

## 7.1. SUMMARY

The main objective of the research presented in this dissertation was to investigate the potential of RISC-V in space applications and define how to exploit it. In order to identify present and future possible uses of RISC-V processors in satellite data systems, in Chapter 2, two different satellite architectures were proposed: a distributed reference satellite data system architecture for medium-large satellites and one for small satellites. A comparison of several RISC-V Intellectual Property (IP) cores against Commercial-Off-The-Shelf (COTS) and space-grade processors for all target applications in each architecture was carried out, along with a critical discussion of benchmarks and metrics to be employed for a fair comparison for each application. From this analysis two sets of activities were identified:

1. Increase the fault tolerance of microarchitectures: In order to do this a literature study was carried out in Chapter 3 and a model to evaluate the vulnerability of microarchitectures to Single Event Effects (SEEs) was developed. Several redundancy techniques were introduced and their effect on microarchitectures similar to state-of-the art and next-generation space processors evaluated with this model.

2. Increase performance of space processors: Instead of focusing on the optimization of existing microarchitecures, the focus here was on identifying new types of applications for space processors for which performance can be dramatically improved by new microarchitectures, as state-of-the-art processors were not designed to execute them efficiently. In Chapter 4 an analysis of the requirements of processors to execute Deep Neural Networks (DNNs) was performed. Then, a study of compute-intensive workloads required for execution of DNNs was carried out, based on the case study of CloudNet, a DNN for cloud detection. Since it was found that the workload is largely composed by matrix operations, in Chapter 5 the RISC-V Vector Extension (RVVE) was proposed to speedup the execution of these matrix operations. The general-purpose processor baseline was then extended by adding vector instructions from the RVVE to achieve performance comparable with those defined in the results of Chapter 2. Finally, in Chapter 6 vector instructions were implemented on a hardware prototype on Field Programmable Gate Array (FPGA) achieving Technology Readiness Level (TRL) 4. Furthermore, a detailed benchmarking with specific kernels and with kernels from CloudNet was carried out to show the benefits of implementing vector instructions.

## 7.2. ANSWERS TO RESEARCH QUESTIONS

The research work presented in this dissertation answered the Research Questions (RQs) formulated in Chapter 1.

**RQ1: What are the advantages of using RISC-V in space applications?**

In Chapter 2, the main features of RISC-V were found to be openness and modularity. The openness of RISC-V already enabled a vast field of research activities for terrestrial

applications, resulting in implementations ranging from tiny microcontrollers to high-performance processors for Artificial Intelligence. Furthermore, many tools (compiler, debuggers, etc.) and models at different level of abstraction are already available (ISA simulators, gem5, IP cores, FPGA prototypes). Therefore, the space industry can spin-in developments from academia and industry, focusing efforts mainly on improvements concerning specific needs in space applications and without wasting efforts on other activities. In order to fully exploit modularity, the need of defining the types of processors required in space application was identified.

**RQ2: How can RISC-V be employed in satellite data systems to solve key issues and enable new capabilities?**

Several applications and processor profiles to address them have been identified in Chapter 2. They are defined by the Instruction Set Architecture (ISA) subset, Instruction-Level Parallelism (ILP), Data-Level Parallelism (DLP), Processor-Level Parallelism (PLP), reference implementation and expected performance; ranging from microcontrollers to general-purpose implementations to high-performance processors for Artificial Intelligence (AI). Finally, a roadmap to bring RISC-V IP cores for terrestrial applications to space level was defined, identifying the steps and models required.

**RQ3: How to design a fault-tolerant high-performance processor for space?**

In order to evaluate the vulnerability of a processor and the effect of including redundancy during the RTL design, a comprehensive model is required. During the creation of this model, it was found that the most impacting factors on the SEE vulnerability are technology, environment and microarchitecture. The focus in this thesis was on the effect of changes in the microarchitecture, varying technology and environment as parameters. The most vulnerable parts identified in both next-generation and state-of-the-art processors were caches, mainly because of the large area occupied. For this reason, addressing the upsets in caches reduces failure rate for state-of-the-art and next-generation processors by 96% and 93% respectively (see Table 3.15). For the rest of the processor, the most cost-effective approach for technologies mostly vulnerable to Single Event Upsets (SEUs) (and when the focus is on dependability instead of performance) is employing FF-level in the logic and Error Correcting Codes (ECC) in the RF. In all the other cases, duplicating the rest of the processor and executing instructions in lockstep on both replicas is more cost-effective. This is applicable to both state-of-the-art processors and next-generation processors, although when dependability is critical the former approach may still be preferable.

**RQ4: How to enhance performance of next-generation processors in satellite data system architectures?**

Part of next-generation processors for space applications will be required to run efficiently new types of workloads compared to state-of-the art processors. The most promising type of new workloads, from a point of view of system capabilities, are DNNs. Their

use in space data system enables higher downlink efficiency and higher dependability, especially in LEO satellites. In Chapter 4, it was shown that a large part of the workload of DNNs is composed of matrix operation. Then, in Chapter 5, it is shown that a vector processor can provide a significant speedup in terms of number of instructions for some of the kernels of DNNs (typically for large matrices), e.g. 88.80× for matrix-multiplications between 128×128 matrices. Finally, in Chapter 6, a TRL 4 hardware prototype achieving 20× the performance of a next-generation scalar processor for space applications was introduced and verified.

## 7.3. INNOVATIONS AND CONTRIBUTIONS

Several innovations and contributions to the body of knowledge were presented in this dissertation:

- **Roadmap for RISC-V processors in space applications (RQ1 and RQ2)**
  In the roadmap proposed in Chapter 2, a clear distinction is made between the contribution to performance due to microarchitecture and the one due to ISA. This is done in order to avoid proliferation of cores which target applications already covered efficiently by processors based on other ISAs or even on the same one, while some space applications may remain unaddressed. ESA employed this study to propose their roadmap in [366]. Cobham Gaisler employed a substantially similar profile classification for their RISC-V cores [367].

- **Open model to estimate vulnerability of IP cores to soft errors and cost-effectiveness of redundancy (RQ3)**
  The amount of redundancy to be employed in an Fault-Tolerant (FT) processor is often decided according to heritage and similarity to proven designs. Even when an analysis of the vulnerability is carried out, often it does not take into account the environment and the technology, only investigating the Architectural Vulnerability Factor (AVF) with a fault injection. The model proposed in Chapter 3 is the first open comprehensive analytical model to allow designers to take into account dependability from the early stages of their design, extending the trade-off between performance, area and power into trade-off between performance, area, power and dependability.

- **Feasibility of the use of machine learning in space applications (RQ4)**
  It was shown that, thanks to available large datasets and given the types of processors required, two types of application of machine learning can be applied successfully in space applications: image segmentation and telemetry analysis. These two applications have a positive impact on the space system, respectively on the downlink efficiency and on the dependability.

- **Vector processor for space applications (RQ4)**
  This dissertation chose a completely different path to satisfy the requirements of Machine Learning (ML) algorithms compared to related work in the space industry (see Sec. 5.1): instead of relying on tools translating the algorithms automatically into hardware designs for FPGA or using COTS component, a processor specifically designed for space was proposed to address ML applications in space.

- **Selective redundancy approach (RQ4)**
  The decoupled pipeline approach, proposed in Sec. 5.3.3, allows the implementation of redundancy only to the scalar pipeline, minimizing the impact on the performance of the scalar pipeline. Cobham Gaisler provides an FT version of NOEL-V. Replacing the non-FT version employed in this work with the FT one will apply this approach.

- **Scalability (RQ4)**
  The "minimal configuration" subset of RVVE identified in Chapter 6 allows an implementation with excellent scalability in terms of resource and frequency utilization up to 256 lanes. Previous vector processors have been presented with up to 16 lanes [19]. This increase of an order of magnitude was due to the selection of a reduced subset of the RVVE.

- **Benchmarking (RQ4)**
  Several novel approaches have been employed to benchmark the vector processor. First of all, the penalty of running algorithms on an external Synchronous Dynamic Random-Access Memory (SDRAM) and on an internal Static Random-Access Memory (SRAM) was investigated, showing that vector processors achieve even lower penalty when the maximum length of a burst of the SDRAM is not exceeded. Furthermore, instead of employing the usual roofline model [198], the actual performance of the prototype is compared with the potential maximum speedup obtained in terms of number of instructions.

- **Advances in computing capabilities of space processors (RQ4)**
  In this dissertation a TRL 4 prototype of a space processor executing a DNN roughly 20x faster than state-of-the-art space processor was presented.



## 7.4. RECOMMENDATIONS AND FUTURE WORK

Given the limited time and manpower available, the research in this dissertation leaves some work to be done and suggests several opportunities to future activities:

- **Optimization of pipeline and software (peak performance)**
  The aim of this work was not to fully optimize a single software kernel on our prototype. In future research, the roofline model could be applied and assembly optimizations carried out as done in [19].

- **Application-Specific Integrated Circuit (ASIC) implementation**
  Although the vector processor was implemented as an FPGA prototype, the final goal of the approach followed in this thesis can only be an ASIC implementation, maximizing power efficiency while preserving the general-purpose nature of a vector unit as an accelerator for DNNs. A 65-nm rad-hard or rad-tolerant technology or smaller, e.g. 22-nm Fully-Depleted Silicon-On-Insulator (FD-SOI), is recommended. It should be verified if the assumption that the area on the die of the ASIC is proportional to the percentage of resources occupied in the FPGA holds.

- **Radiation tests**
  The ASIC should go through a SEE test campaign to validate the selective redundancy

approach. Testing the FPGA is deemed not useful to understand the vulnerability of the final vector processor, because, as shown in the auxiliary work carried out by a Master Student under the supervision of the PhD in [368], most of the failures observed will be due to upsets in the configuration memory, a failure mechanism not present in the final ASIC. A test of the ASIC would also validate further the model in Chapter 3.

- **Additional vector instructions**
  Although additional instructions, as non-unit stride load and store, may be beneficial for some applications, adding floating-point instruction is deemed not fit for space applications and not required if the goal is to run DNNs inference on-board.

- **Element size**
  As the main objective of this thesis was to prove the advantages of a vector pipeline against a scalar pipeline, the vector implementation operates on 32-bit elements, in order to allow a fair comparison of performance with the baseline scalar processor. To further improve performance, operations on 8-bit and 16-bit elements can be employed, expecting roughly 4× and 2× the performance of the 32-bit version (with the approximately the same resources), respectively.

- **Vector data caches**
  Although it was shown that vector data caching can speedup the execution of compute-bound kernels, it would imply a much larger area than the cacheless vector pipeline proposed in this dissertation. Furthermore, it was shown how a vector pipeline overcomes memory latency employing DLP (see Fig. 6.3, bottom), making vector data caching less useful than previously thought (see Fig. 6.8).

# REFERENCES


[1] S. Di Mascio, A. Menicucci, E. Gill, G. Furano, and C. Monteleone. "Leveraging the Openness and Modularity of RISC-V in Space". In: *Journal of Aerospace Information Systems* 16.11 (2019), pp. 454–472. DOI: 10.2514/1.I010735.

[2] S. Di Mascio, A. Menicucci, G. Furano, C. Monteleone, and M. Ottavi. "The Case for RISC-V in Space". In: *International Conference on Applications in Electronics Pervading Industry, Environment and Society*. Springer. 2018, pp. 319–325. DOI: 10.1007/978-3-030-11973-7_37.

[3] D. Girimonte and D. Izzo. "Artificial intelligence for space applications". In: *Intelligent Computing Everywhere*. Springer, 2007, pp. 235–253. DOI: 10.1007/978-1-84628-943-9_12.

[4] S. Chien, R. Doyle, A. G. Davies, A. Jonsson, and R. Lorenz. "The future of AI in space". In: *IEEE Intelligent Systems* 21.4 (2006), pp. 64–69. DOI: 10.1109/MIS.2006.79.

[5] A. Gillette, C. Wilson, and A. D. George. "Efficient and autonomous processing and classification of images on small spacecraft". In: *2017 IEEE National Aerospace and Electronics Conference (NAECON)*. 2017, pp. 135–141.

[6] J. Doubleday, S. Chien, C. Norton, K. Wagstaff, D. R. Thompson, J. Bellardo, C. Francis, and E. Baumgarten. "Autonomy for remote sensing — Experiences from the IPEX CubeSat". In: *2015 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*. July 2015, pp. 5308–5311. DOI: 10.1109/IGARSS.2015.7327033.

[7] D. Izzo and L. Pettazzi. "Autonomous and distributed motion planning for satellite swarm". In: *Journal of Guidance, Control, and Dynamics* 30.2 (2007), pp. 449–459.

[8] G. Lentaris, K. Maragos, I. Stratakos, L. Papadopoulos, O. Papanikolaou, D. Soudris, M. Lourakis, X. Zabulis, D. Gonzalez-Arjona, and G. Furano. "High-Performance Embedded Computing in Space: Evaluation of Platforms for Vision-Based Navigation". In: *Journal of Aerospace Information Systems* 15.4 (2018), pp. 178–192. DOI: 10.2514/1.I010555.

[9] K. Erickson. "Optimal architecture for an asteroid mining mission: equipment details and integration". In: *Space 2006*. 2006, p. 7504.

[10] D. Gaines, R. Anderson, G. Doran, W. Huffman, H. Justice, R. Mackey, G. Rabideau, A. Vasavada, V. Verma, T. Estlin, et al. "Productivity challenges for mars rover operations". In: *Proceedings of 4th Workshop on Planning and Robotics (PlanRob)*. London, UK. 2016, pp. 115–125.

[11] J. Lemley, S. Bazrafkan, and P. Corcoran. “Deep Learning for Consumer Devices and Services: Pushing the limits for machine learning, artificial intelligence, and computer vision.” In: *IEEE Consumer Electronics Magazine* 6.2 (Apr. 2017), pp. 48–56. ISSN: 2162-2248. DOI: 10.1109/MCE.2016.2640698.

[12] P. Dodd and L. Massengill. “Basic mechanisms and modeling of single-event upset in digital microelectronics”. In: *IEEE Transactions on Nuclear Science* 50.3 (2003), pp. 583–602. DOI: 10.1109/TNS.2003.813129.

[13] S. Di Mascio, A. Menicucci, G. Furano, T. Szewczyk, L. Campajola, F. Di Capua, A. Lucaroni, and M. Ottavi. “Towards defining a simplified procedure for COTS system-on-chip TID testing”. In: *Nuclear Engineering and Technology* 50.8 (2018), pp. 1298–1305. ISSN: 1738-5733. DOI: https://doi.org/10.1016/j.net.2018.07.010.

[14] R. Hillman, G. Swift, P. Layton, M. Conrad, C. Thibodeau, and F. Irom. “Space processor radiation mitigation and validation techniques for an 1,800 MIPS processor board”. In: *Proceedings of the 7th European Conference on Radiation and Its Effects on Components and Systems, 2003. RADECS 2003.* Sept. 2003, pp. 347–352.

[15] D. A. Patterson and J. L. Hennessy. *Computer Organization and Design RISC-V Edition: The Hardware/Software Interface*. Elsevier Science, 2017. ISBN: 9780128122761.

[16] P. M. Kogge. *The architecture of pipelined computers*. CRC press, 1981. ISBN: 9780891164944.

[17] Y. Baida. *Pipelining Review and Its Limitations*. 2010. URL: https://mipt.ru/drec/about/ilab/upload/02a/f_4vj711-arpgiu3f1e1.pdf.

[18] D. Dabbelt, C. Schmidt, E. Love, H. Mao, S. Karandikar, and K. Asanovic. “Vector Processors for Energy-Efficient Embedded Systems”. In: *Proceedings of the Third ACM International Workshop on Many-core Embedded Systems*. MES ’16. Seoul, Republic of Korea: ACM, 2016, pp. 10–16. ISBN: 978-1-4503-4262-9. DOI: 10.1145/2934495.2934497.

[19] M. Cavalcante, F. Schuiki, F. Zaruba, M. Schaffner, and L. Benini. “Ara: A 1-GHz+ Scalable and Energy-Efficient RISC-V Vector Processor With Multiprecision Floating-Point Support in 22-nm FD-SOI”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 28.2 (2020), pp. 530–543.

[20] N. Stephens, S. Biles, M. Boettcher, J. Eapen, M. Eyole, G. Gabrielli, M. Horsnell, G. Magklis, A. Martinez, N. Premillieu, A. Reid, A. Rico, and P. Walker. “The ARM Scalable Vector Extension”. In: *IEEE Micro* 37.2 (2017), pp. 26–39.

[21] Imagination Technologies. *An Overview of MIPS Multi-Threading -White Paper*. 2018. URL: https://www.mips.com/wp-content/uploads/2017/10/Overview_of_MIPS_Multi_Threading.pdf.

[22] A. Cheikh, G. Cerutti, A. Mastrandrea, F. Menichelli, and M. Olivieri. “The microarchitecture of a multi-threaded RISC-V compliant processing core family for IoT end-nodes”. In: *ApplePies*. 2017. DOI: 10.1007/978-3-319-93082-4_12.

[23] H. V. Caprita and M. Popa. “Design methods of multithreaded architectures for multicore microcontrollers”. In: *2011 6th IEEE International Symposium on Applied Computational Intelligence and Informatics (SACI)*. May 2011, pp. 427–432. DOI: 10.1109/SACI.2011.5873041.

[24] D. Geer. “Chip makers turn to multicore processors”. In: *Computer* 38.5 (2005), pp. 11–13. DOI: 10.1109/MC.2005.160.

[25] W. Gropp and E. Lusk. “A taxonomy of programming models for symmetric multiprocessors and SMP clusters”. In: *Programming Models for Massively Parallel Computers*. 1995, pp. 2–7. DOI: 10.1109/PMMPC.1995.504335.

[26] J. L. Gustafson. “Reevaluating Amdahl’s Law”. In: *Commun. ACM* 31.5 (May 1988), pp. 532–533. ISSN: 0001-0782. DOI: 10.1145/42411.42415.

[27] A. Varghese, B. Edwards, G. Mitra, and A. P. Rendell. “Programming the Adapteva Epiphany 64-Core Network-on-Chip Coprocessor”. In: *2014 IEEE International Parallel Distributed Processing Symposium Workshops*. 2014, pp. 984–992. DOI: 10.1109/IPDPSW.2014.112.

[28] F. Sanchez, E. Salami, A. Ramirez, and M. Valero. “Performance Analysis of Sequence Alignment Applications”. In: *2006 IEEE International Symposium on Workload Characterization*. 2006, pp. 51–60. DOI: 10.1109/IISWC.2006.302729.

[29] T. Ball and J. R. Larus. “Branch prediction for free”. In: *ACM SIGPLAN Notices* 28.6 (1993), pp. 300–313.

[30] J. Andersson, M. Hjorth, F. Johansson, and S. Habinc. “LEON Processor Devices for Space Missions: First 20 Years of LEON in Space”. In: *2017 6th International Conference on Space Mission Challenges for Information Technology (SMC-IT)*. Sept. 2017, pp. 136–141. DOI: 10.1109/SMC-IT.2017.31.

[31] G. Sun, C. Hughes, C. Kim, J. Zhao, C. Xu, Y. Xie, and Y.-K. Chen. “Moguls: A model to explore the memory hierarchy for bandwidth improvements”. In: *2011 38th Annual International Symposium on Computer Architecture (ISCA)*. 2011, pp. 377–388.

[32] Z. Al-Waisi and M. O. Agyeman. “An overview of on-chip cache coherence protocols”. In: *2017 Intelligent Systems Conference (IntelliSys)*. 2017, pp. 304–309. DOI: 10.1109/IntelliSys.2017.8324309.

[33] S. Gal-On and M. Levy. *Exploring CoreMark - A Benchmark Maximizing Simplicity and Efficacy*. https://www.eembc.org/techlit/articles/coremark-whitepaper.pdf. 2012.

[34] D. Bhandarkar and J. Ding. “Performance characterization of the Pentium Pro processor”. In: *Proceedings Third International Symposium on High-Performance Computer Architecture*. 1997, pp. 288–297. DOI: 10.1109/HPCA.1997.569689.

[35] A. R. Weiss. “Dhrystone Benchmark”. In: (2002).

[36] R. York. “Benchmarking in context: Dhrystone”. In: *ARM, March* (2002).

[37] P. Deitel and H. Deitel. *C How to Program - Seventh Edition*. Prentice Hall, 2013.

[38] *CoreMark*. Accessed on 2021-10-25. URL: https://github.com/eembc/coremark.

[39] A. Avizienis, J. .-. Laprie, B. Randell, and C. Landwehr. "Basic concepts and taxonomy of dependable and secure computing". In: *IEEE Transactions on Dependable and Secure Computing* 1.1 (Jan. 2004), pp. 11–33. ISSN: 1545-5971. DOI: 10.1109/TDSC.2004.2.

[40] V. G. Rao and H. Mahmoodi. "Analysis of reliability of flip-flops under transistor aging effects in nano-scale CMOS technology". In: *2011 IEEE 29th International Conference on Computer Design (ICCD)*. IEEE. 2011, pp. 439–440.

[41] R. Baumann. "Soft errors in advanced computer systems". In: *IEEE Design Test of Computers* 22.3 (May 2005), pp. 258–266. ISSN: 1558-1918. DOI: 10.1109/MDT.2005.69.

[42] M.-L. Li, P. Ramachandran, S. K. Sahoo, S. V. Adve, V. S. Adve, and Y. Zhou. "Understanding the propagation of hard errors to software and implications for resilient system design". In: *ACM Sigplan Notices* 43.3 (2008), pp. 265–276.

[43] V. Sridharan, J. Stearley, N. DeBardeleben, S. Blanchard, and S. Gurumurthi. "Feng Shui of supercomputer memory positional effects in DRAM and SRAM faults". In: *SC '13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. Nov. 2013, pp. 1–11.

[44] T. Siddiqua, V. Sridharan, S. E. Raasch, N. DeBardeleben, K. B. Ferreira, S. Levy, E. Baseman, and Q. Guan. "Lifetime memory reliability data from the field". In: *2017 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. Oct. 2017, pp. 1–6. DOI: 10.1109/DFT.2017.8244428.

[45] J. L. Barth, C. Dyer, and E. Stassinopoulos. "Space, atmospheric, and terrestrial radiation environments". In: *IEEE Transactions on nuclear science* 50.3 (2003), pp. 466–482.

[46] N. A. Dodds. *Single event latchup: hardening strategies, triggering mechanisms, and testing considerations*. Vanderbilt University, 2012.

[47] M. Ebrahimi, A. Evans, M. B. Tahoori, E. Costenaro, D. Alexandrescu, V. Chandra, and R. Seyyedi. "Comprehensive Analysis of Sequential and Combinational Soft Errors in an Embedded Processor". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34.10 (Oct. 2015), pp. 1586–1599. ISSN: 1937-4151. DOI: 10.1109/TCAD.2015.2422845.

[48] A. L. Bogorad, J. J. Likar, R. E. Lombardi, S. E. Stone, and R. Herschitz. "On-orbit error rates of RHBD SRAMs: Comparison of calculation techniques and space environmental models with observed performance". In: *IEEE Transactions on Nuclear Science* 58.6 (2011), pp. 2804–2806. DOI: 10.1109/TNS.2011.2167242.

[49] L. Edmonds, C. Barnes, and L. Scheick. *An introduction to space radiation effects on microelectronics*. Jet Propulsion Laboratory, National Aeronautics and Space Administration, 2000.

[50] D. Heynderickx, B. Quaghebeur, J. Wera, E. J. Daly, and H. D. R. Evans. "New radiation environment and effects models in the european space agency's space environment information system (SPENVIS)". In: *Space Weather* 2.10 (Oct. 2004), pp. 1–4. ISSN: 1542-7390. DOI: 10.1029/2004SW000073.

[51] G. Hubert, L. Artola, and D. Regis. "Impact of scaling on the soft error sensitivity of bulk, FDSOI and FinFET technologies due to atmospheric radiation". In: *Integration* 50 (2015), pp. 39–47. ISSN: 0167-9260. DOI: https://doi.org/10.1016/j.vlsi.2015.01.003.

[52] J. A. Pellish, M. A. Xapsos, C. A. Stauffer, T. M. Jordan, A. B. Sanders, R. L. Ladbury, T. R. Oldham, P. W. Marshall, D. F. Heidel, and K. P. Rodbell. "Impact of Spacecraft Shielding on Direct Ionization Soft Error Rates for Sub-130 nm Technologies". In: *IEEE Transactions on Nuclear Science* 57.6 (Dec. 2010), pp. 3183–3189. ISSN: 0018-9499. DOI: 10.1109/TNS.2010.2084595.

[53] H. Michel, H. Guzmán-Miranda, A. Dörflinger, H. Michalik, and M. A. Echanove. "SEU fault classification by fault injection for an FPGA in the space instrument SOPHI". In: *2017 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. July 2017, pp. 9–15. DOI: 10.1109/AHS.2017.8046353.

[54] C. Fong, S. Yang, C. Chu, C. Huang, J. Yeh, C. Lin, T. Kuo, T. Liu, N. L. Yen, S. Chen, Y. Kuo, Y. Liou, and S. Chi. "FORMOSAT-3/COSMIC Constellation Spacecraft System Performance: After One Year in Orbit". In: *IEEE Transactions on Geoscience and Remote Sensing* 46.11 (Nov. 2008), pp. 3380–3394. DOI: 10.1109/TGRS.2008.2005203.

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

[56] R. García Alía, M. Brugger, E. Daly, S. Danzeca, V. Ferlet-Cavrois, R. Gaillard, J. Mekki, C. Poivey, and A. Zadeh. "Simplified SEE Sensitivity Screening for COTS Components in Space". In: *IEEE Transactions on Nuclear Science* 64.2 (Feb. 2017), pp. 882–890. ISSN: 1558-1578. DOI: 10.1109/TNS.2017.2653863.

[57] B. D. Sierawski, R. A. Reed, K. M. Warren, A. L. Sternberg, R. A. Austin, J. M. Trippe, R. A. Weller, M. L. Alles, R. D. Schrimpf, L. W. Massengill, D. M. Fleetwood, G. W. Buxton, J. C. Brandenburg, W. B. Fisher, and R. Davis. "CubeSat: Real-time soft error measurements at low earth orbits". In: *2017 IEEE International Reliability Physics Symposium (IRPS)*. Apr. 2017, pp. 3D-1.1-3D–1.6. DOI: 10.1109/IRPS.2017.7936291.

[58] G. Furano and A. Menicucci. "Roadmap for on-board processing and data handling systems in space". In: *Dependable Multicore Architectures at Nanoscale*. Springer, 2018, pp. 253–281.

[59] F. Zaruba and L. Benini. "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". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27.11 (2019), pp. 2629–2640.

[60] D. L. Weaver. *OpenSPARC internals: OpenSPARC T1/T2 CMT throughput computing*. Sun Microsystems, 2008.

[61] A. Dixit and A. Wood. "The impact of new technology on soft error rates". In: *2011 International Reliability Physics Symposium*. Apr. 2011, 5B.4.1–5B.4.7. DOI: 10.1109/IRPS.2011.5784522.

[62] S. Buchner, M. Baze, D. Brown, D. McMorrow, and J. Melinger. "Comparison of error rates in combinational and sequential logic". In: *IEEE Transactions on Nuclear Science* 44.6 (Dec. 1997), pp. 2209–2216. ISSN: 0018-9499. DOI: 10.1109/23.659037.

[63] S. Jagannathan, T. D. Loveless, B. L. Bhuva, N. J. Gaspard, N. Mahatme, T. Assis, S. .-. Wen, R. Wong, and L. W. Massengill. "Frequency Dependence of Alpha-Particle Induced Soft Error Rates of Flip-Flops in40-nm CMOS Technology". In: *IEEE Transactions on Nuclear Science* 59.6 (Dec. 2012), pp. 2796–2802. ISSN: 1558-1578. DOI: 10.1109/TNS.2012.2223827.

[64] STMicroelectronics. *A 65nm hardened ASIC technology for Space applications*. 2017. URL: https://indico.esa.int/event/165/contributions/1218/attachments/1205/1425/05b_-_KIPSAT_-_Presentation.pdf.

[65] S. Clerc, F. Abouzeid, G. Gasiot, J. Daveau, C. Bottoni, M. Glorieux, J. Autran, F. Cacho, V. Huard, L. Dugoujon, R. Weigand, F. Malou, L. Hili, and P. Roche. "Space radiation and reliability qualifications on 65nm CMOS 600MHz microprocessors". In: *2013 IEEE International Reliability Physics Symposium (IRPS)*. Apr. 2013, pp. 6C.1.1–6C.1.7. DOI: 10.1109/IRPS.2013.6532051.

[66] R. M. Goodman and M. Sayano. "The reliability of semiconductor RAM memories with on-chip error-correction coding". In: *IEEE Transactions on Information Theory* 37.3 (1991), pp. 884–896.

[67] G. Georgakos, P. Huber, M. Ostermayr, E. Amirante, and F. Ruckerbauer. "Investigation of Increased Multi-Bit Failure Rate Due to Neutron Induced SEU in Advanced Embedded SRAMs". In: *2007 IEEE Symposium on VLSI Circuits*. June 2007, pp. 80–81. DOI: 10.1109/VLSIC.2007.4342774.

[68] M. Raine, G. Hubert, M. Gaillardin, P. Paillet, and A. Bournel. "Monte Carlo Prediction of Heavy Ion Induced MBU Sensitivity for SOI SRAMs Using Radial Ionization Profile". In: *IEEE Transactions on Nuclear Science* 58.6 (Dec. 2011), pp. 2607–2613. ISSN: 0018-9499. DOI: 10.1109/TNS.2011.2168238.

[69] M. J. Gadlage, R. D. Schrimpf, J. M. Benedetto, P. H. Eaton, D. G. Mavis, M. Sibley, K. Avery, and T. L. Turflinger. "Single event transient pulse widths in digital microcircuits". In: *IEEE Transactions on Nuclear Science* 51.6 (Dec. 2004), pp. 3285–3290. ISSN: 1558-1578. DOI: 10.1109/TNS.2004.839174.

[70] R. C. Harrington, J. S. Kauppila, K. M. Warren, Y. P. Chen, J. A. Maharrey, T. D. Haeffner, T. D. Loveless, B. L. Bhuva, M. Bounasser, K. Lilja, and L. W. Massengill. "Estimating Single-Event Logic Cross Sections in Advanced Technologies". In: *IEEE Transactions on Nuclear Science* 64.8 (Aug. 2017), pp. 2115–2121. DOI: 10.1109/TNS.2017.2718517.

[71] R. C. Baumann. “Radiation-induced soft errors in advanced semiconductor technologies”. In: *IEEE Transactions on Device and materials reliability* 5.3 (2005), pp. 305–316. DOI: 10.1109/TDMR.2005.853449.

[72] N. N. Mahatme, S. Jagannathan, T. D. Loveless, L. W. Massengill, B. L. Bhuva, S. .-. Wen, and R. Wong. “Comparison of Combinational and Sequential Error Rates for a Deep Submicron Process”. In: *IEEE Transactions on Nuclear Science* 58.6 (Dec. 2011), pp. 2719–2725. ISSN: 0018-9499. DOI: 10.1109/TNS.2011.2171993.

[73] J. Velamala, R. LiVolsi, M. Torres, and Y. Cao. “Design sensitivity of Single Event Transients in scaled logic circuits”. In: *2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC)*. June 2011, pp. 694–699.

[74] A. Ramos, J. A. Maestro, and P. Reviriego. “Characterizing a RISC-V SRAM-based FPGA implementation against Single Event Upsets using fault injection”. In: *Microelectronics Reliability* 78 (2017), pp. 205–211. ISSN: 0026-2714. DOI: https://doi.org/10.1016/j.microrel.2017.09.007.

[75] C. Bernardeschi, L. Cassano, A. Domenici, and L. Sterpone. “Accurate simulation of SEUs in the configuration memory of SRAM-based FPGAs”. In: *2012 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. Oct. 2012, pp. 115–120. DOI: 10.1109/DFT.2012.6378210.

[76] S. K. Reinhardt and S. S. Mukherjee. “Transient Fault Detection via Simultaneous Multithreading”. In: *Proceedings of the 27th Annual International Symposium on Computer Architecture*. ISCA ’00. Vancouver, British Columbia, Canada: Association for Computing Machinery, 2000, pp. 25–36. ISBN: 1581132328. DOI: 10.1145/339647.339652. URL: https://doi.org/10.1145/339647.339652.

[77] D. Gil, R. Martínez, J. V. Busquets, J. C. Baraza, and P. J. Gil. “Fault Injection into VHDL Models: Experimental Validation of a Fault Tolerant Microcomputer System”. In: *Proceedings of the Third European Dependable Computing Conference on Dependable Computing*. EDCC-3. Berlin, Heidelberg: Springer-Verlag, 1999, pp. 191–208.

[78] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin. “A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor”. In: *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO 36. Washington, DC, USA: IEEE Computer Society, 2003, pp. 29–. ISBN: 0-7695-2043-X. URL: http://dl.acm.org/citation.cfm?id=956417.956570.

[79] R. W. Hamming. “Error detecting and error correcting codes”. In: *The Bell system technical journal* 29.2 (1950), pp. 147–160.

[80] J. Kim, N. Hardavellas, K. Mai, B. Falsafi, and J. Hoe. “Multi-bit Error Tolerant Caches Using Two-Dimensional Error Coding”. In: *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*. Dec. 2007, pp. 197–209. DOI: 10.1109/MICRO.2007.19.

[81] J. Gaisler. “A portable and fault-tolerant microprocessor based on the SPARC v8 architecture”. In: *Proceedings International Conference on Dependable Systems and Networks*. June 2002, pp. 409–415. DOI: 10.1109/DSN.2002.1028926.

[82] H. Q. Le, W. J. Starke, J. S. Fields, F. P. O'Connell, D. Q. Nguyen, B. J. Ronchetti, W. M. Sauer, E. M. Schwarz, and M. T. Vaden. "IBM POWER6 microarchitecture". In: *IBM Journal of Research and Development* 51.6 (Nov. 2007), pp. 639–662. ISSN: 0018-8646. DOI: 10.1147/rd.516.0639.

[83] Manish Shah, J. Barreh, J. Brooks, R. Golla, G. Grohoski, N. Gura, R. Hetherington, P. Jordan, M. Luttrell, C. Olson, Bikram Saha, D. Sheahan, L. Spracklen, and A. Wynn. "UltraSPARC T2: A highly-treaded, power-efficient, SPARC SOC". In: *2007 IEEE Asian Solid-State Circuits Conference*. Nov. 2007, pp. 22–25. DOI: 10.1109/ASSCC.2007.4425786.

[84] Nhon Quach. "High availability and reliability in the itanium processor". In: *IEEE Micro* 20.5 (Sept. 2000), pp. 61–69. ISSN: 0272-1732. DOI: 10.1109/40.877951.

[85] X. Iturbe, B. Venu, E. Ozer, J.-L. Poupat, G. Gimenez, and H.-U. Zurek. "The Arm Triple Core Lock-Step (TCLS) Processor". In: *ACM Trans. Comput. Syst.* 36.3 (June 2019). ISSN: 0734-2071. DOI: 10.1145/3323917. URL: https://doi.org/10.1145/3323917.

[86] I. Alam, C. Schoeny, L. Dolecek, and P. Gupta. "Parity++: Lightweight Error Correction for Last Level Caches". In: *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. June 2018, pp. 114–120. DOI: 10.1109/DSN-W.2018.00048.

[87] M. Richter, K. Oberlaender, and M. Goessel. "New Linear SEC-DED Codes with Reduced Triple Bit Error Miscorrection Probability". In: *2008 14th IEEE International On-Line Testing Symposium*. July 2008, pp. 37–42. DOI: 10.1109/IOLTS.2008.27.

[88] S. Cha and H. Yoon. "Efficient implementation of single error correction and double error detection code with check bit pre-computation for memories". In: *JSTS: Journal of Semiconductor Technology and Science* 12.4 (2012), pp. 418–425.

[89] P. Reviriego, J. A. Maestro, S. Baeg, S. Wen, and R. Wong. "Protection of Memories Suffering MCUs Through the Selection of the Optimal Interleaving Distance". In: *IEEE Transactions on Nuclear Science* 57.4 (Aug. 2010), pp. 2124–2128. DOI: 10.1109/TNS.2010.2042818.

[90] Hsiang-Jen Tsai, Chien-Chih Chen, Keng-Hao Yang, Ting-Chin Yang, Li-Yue Huang, Ching-Hao Chung, Meng-Fan Chang, and Tien-Fu Chen. "Leveraging data lifetime for energy-aware last level non-volatile SRAM caches using redundant store elimination". In: *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*. June 2014, pp. 1–6. DOI: 10.1145/2593069.2593153.

[91] M. Focardi, S. Pezzuto, R. Cosentino, G. Giusi, M. Pancrazzi, V. Noce, R. Ottensamer, M. Steller, A. M. D. Giorgio, E. Pace, P. Plasson, G. Peter, and I. Pagano. "The instrument control unit of the ESA-PLATO 2.0 mission". In: *Space Telescopes and Instrumentation 2016: Optical, Infrared, and Millimeter Wave*. Ed. by H. A. MacEwen, G. G. Fazio, M. Lystrup, N. Batalha, N. Siegler, and E. C. Tong. Vol. 9904. International Society for Optics and Photonics. SPIE, 2016, pp. 962–976. DOI: 10.1117/12.2231658. URL: https://doi.org/10.1117/12.2231658.

[92] A. Evans, C. U. Ortega, K. Marinis, E. Costenaro, H. Laroussi, K.-V. Obbe, G. Magistrati, and V. Ferlet-Cavrois. “Heavy-ion micro beam and simulation study of a flash-based FPGA microcontroller implementation”. In: *IEEE Transactions on Nuclear Science* 64.1 (2017), pp. 504–511. DOI: 10.1109/TNS.2016.2633401.

[93] X. He, X. Huang, and Y. Li. “TMR-Based Soft Error Tolerance Techniques in ASIC Design”. In: *2016 6th International Conference on Advanced Design and Manufacturing Engineering (ICADME 2016)*. Atlantis Press. 2017. DOI: https://doi.org/10.2991/icadme-16.2016.76.

[94] M. M. Ghahroodi, E. Ozer, and D. Bull. “SEU and SET-tolerant ARM Cortex-R4 CPU for space and avionics applications”. In: *Proc. of the Workshop on Manufacturable and Dependable Multi-core Architectures at Nanoscale*. 2013.

[95] R. Yamamoto, C. Hamanaka, J. Furuta, K. Kobayashi, and H. Onodera. “An Area-Efficient 65 nm Radiation-Hard Dual-Modular Flip-Flop to Avoid Multiple Cell Upsets”. In: *IEEE Transactions on Nuclear Science* 58.6 (Dec. 2011), pp. 3053–3059. ISSN: 1558-1578. DOI: 10.1109/TNS.2011.2169457.

[96] K. Kobayashi, K. Kubota, M. Masuda, Y. Manzawa, J. Furuta, S. Kanda, and H. Onodera. “A Low-Power and Area-Efficient Radiation-Hard Redundant Flip-Flop, DICE ACFF, in a 65 nm Thin-BOX FD-SOI”. In: *IEEE Transactions on Nuclear Science* 61.4 (Aug. 2014), pp. 1881–1888. ISSN: 1558-1578. DOI: 10.1109/TNS.2014.2318326.

[97] C. Bottoni, B. Coeffic, J. Daveau, L. Naviner, and P. Roche. “Partial triplication of a SPARC-V8 microprocessor using fault injection”. In: *2015 IEEE 6th Latin American Symposium on Circuits Systems (LASCAS)*. Feb. 2015, pp. 1–4. DOI: 10.1109/LASCAS.2015.7250415.

[98] X. Iturbe, B. Venu, E. Ozer, and S. Das. “A Triple Core Lock-Step (TCLS) ARM Cortex-R5 Processor for Safety-Critical and Ultra-Reliable Applications”. In: *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W)*. June 2016, pp. 246–249. DOI: 10.1109/DSN-W.2016.57.

[99] B. Venu, E. Ozer, X. Iturbe, and A. Robinson. “A fail-functional automotive CPU subsystem architecture for mitigating single point of failures”. In:

[100] X. Iturbe, B. Venu, and E. Ozer. “Soft error vulnerability assessment of the real-time safety-related ARM Cortex-R5 CPU”. In: *2016 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. Sept. 2016, pp. 91–96. DOI: 10.1109/DFT.2016.7684076.

[101] G. Furano, S. Di Mascio, T. Szewczyk, A. Menicucci, L. Campajola, F. Di Capua, A. Fabbri, and M. Ottavi. “A novel method for SEE validation of complex SoCs using Low-Energy Proton beams”. In: *2016 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. Sept. 2016, pp. 131–134. DOI: 10.1109/DFT.2016.7684084.

[102] Á. B. de Oliveira, G. S. Rodrigues, F. L. Kastensmidt, N. Added, E. L. A. Macchione, V. A. P. Aguiar, N. H. Medina, and M. A. G. Silveira. "Lockstep Dual-Core ARM A9: Implementation and Resilience Analysis Under Heavy Ion-Induced Soft Errors". In: *IEEE Transactions on Nuclear Science* 65.8 (Aug. 2018), pp. 1783–1790. ISSN: 1558-1578. DOI: 10.1109/TNS.2018.2852606.

[103] S. Shrimpton. *An Implementation of MIL-STD-1750 Airborne Computer Instruction Set Architecture*. Tech. rep. ROYAL AIRCRAFT ESTABLISHMENT FARNBOROUGH (ENGLAND), 1981.

[104] J. Gaisler. *25 Years of SPARC*. https://indico.esa.int/event/182/contributions/1526/attachments/1400/1625/0905_-_Gaisler.pdf. 2017.

[105] Cobham Gaisler. *GR740 Qualification Results*. 2021. URL: https://escies.org/download/webDocumentFile?id=68442.

[106] Cobham Gaisler. *LEON4 Processor*. https://www.gaisler.com/index.php/products/processors/leon4.

[107] EEMBC. *Coremark scores*. https://www.eembc.org/coremark/scores.php. Accessed: 2018-12-21.

[108] K. Asanovic, D. A. Patterson, and C. Celio. *The berkeley out-of-order machine (boom): An industry-competitive, synthesizable, parameterized risc-v processor*. Tech. rep. University of California at Berkeley Berkeley United States, 2015.

[109] M. Demler. "ARC HS4x and HS4xD CPUs: New Dual-Issue Architecture Boosts Embedded Processor Performance". In: 2018.

[110] P. D. Schiavone, F. Conti, D. Rossi, M. Gautschi, A. Pullini, E. Flamand, and L. Benini. "Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for Internet-of-Things applications". In: *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*. Sept. 2017, pp. 1–8. DOI: 10.1109/PATMOS.2017.8106976.

[111] A. Bardizbanyan and P. Larsson-Edefors. "Exploring early and late ALUs for single-issue in-order pipelines". In: *2015 33rd IEEE International Conference on Computer Design (ICCD)*. Oct. 2015, pp. 543–548. DOI: 10.1109/ICCD.2015.7357163.

[112] M. S. Hrishikesh, N. P. Jouppi, K. I. Farkas, D. Burger, S. W. Keckler, and P. Shivakumar. "The optimal logic depth per pipeline stage is 6 to 8 FO4 inverter delays". In: *Proceedings 29th Annual International Symposium on Computer Architecture*. May 2002, pp. 14–24. DOI: 10.1109/ISCA.2002.1003558.

[113] K. Asanovic, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz, et al. "The rocket chip generator". In: *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17* (2016).

[114] N. Jouppi. "Cache Write Policies And Performance". In: *Proceedings of the 20th Annual International Symposium on Computer Architecture*. 1993, pp. 191–201. DOI: 10.1109/ISCA.1993.698560.

[115] M. Fernández, R. Gioiosa, E. Quiñones, L. Fossati, M. Zulianello, and F. J. Cazorla. "Assessing the Suitability of the NGMP Multi-core Processor in the Space Domain". In: *Proceedings of the Tenth ACM International Conference on Embedded Software*. EMSOFT '12. Tampere, Finland: ACM, 2012, pp. 175–184. ISBN: 978-1-4503-1425-1. DOI: 10.1145/2380356.2380389.

[116] 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". In: *IEEE Transactions on Nuclear Science* 62.6 (Dec. 2015), pp. 2687–2694. ISSN: 0018-9499. DOI: 10.1109/TNS.2015.2492824.

[117] M. Pignol. "COTS-based applications in space avionics". In: *2010 Design, Automation Test in Europe Conference Exhibition (DATE 2010)*. Mar. 2010, pp. 1213–1219. DOI: 10.1109/DATE.2010.5456992.

[118] S. Frasse-Sombet, Y. Brunel, and S. Mariottini. "Bringing high-performance microprocessors up to space level reliability". In: *Aerospace Conference, 2015 IEEE*. IEEE. 2015, pp. 1–8.

[119] K. Asanović and D. A. Patterson. "Instruction sets should be free: The case for RISC-V". In: *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-146* (2014).

[120] D. Margan and S. Čandrlić. "The success of open source software: A review". In: *Information and Communication Technology, Electronics and Microelectronics, 38th International Convention on*. IEEE. 2015, pp. 1463–1468.

[121] A. Waterman and K. Asanovic. "The RISC-V Instruction Set Manual-Volume I: User-Level ISA-Document Version 2.2". In: (2017).

[122] A. Waterman, Y. Lee, R. Avizienis, H. Cook, D. Patterson, and K. Asanovic. "The RISC-V Instruction Set". In: *Hot Chips 25 Symposium (HCS)*. IEEE. 2013, pp. 25–27.

[123] L. G. Salmon. "A Perspective on the Role of Open-Source IP In Government Electronic Systems". In: *7th RISC-V Workshop Proceedings*. 2017.

[124] *RISC-V Exchange: Available Software*. "https://riscv.org/exchange/software/". Accessed: 2021-09-29.

[125] D. Rossi, F. Conti, A. Marongiu, A. Pullini, I. Loi, M. Gautschi, G. Tagliavini, A. Capotondi, P. Flatresse, and L. Benini. "PULP: A parallel ultra low power platform for next generation IoT applications". In: *2015 IEEE Hot Chips 27 Symposium (HCS)*. Aug. 2015, pp. 1–39. DOI: 10.1109/HOTCHIPS.2015.7477325.

[126] M. Fink. "RISC-V: Enabling a New Era of Open Data-Centric Computing Architectures". In: *7th RISC-V Workshop Proceedings*. 2017.

[127] A. Waterman and K. Asanovic. "The RISC-V Instruction Set Manual, Volume II: Privileged architecture". In: (2017).

[128] F. Gómez, M. Masmano, V. Nicolau, J. Andersson, J. Le Rhun, D. Trilla, F. Gallego, G. Cabo, and J. Abella Ferrer. "De-RISC–Dependable Real-Time Infrastructure for Safety-Critical Computer Systems". In: *Ada User Journal* 41.2 (2020), pp. 107–112.

[129] R. Shu, P. Wang, S. A. Gorski III, B. Andow, A. Nadkarni, L. Deshotels, J. Gionta, W. Enck, and X. Gu. "A Study of Security Isolation Techniques". In: *ACM Comput. Surv.* 49.3 (Oct. 2016). ISSN: 0360-0300. DOI: 10.1145/2988545.

[130] S.-Y. Fu, C.-M. Lin, D.-Y. Hong, Y.-P. Liu, J.-J. Wu, and W.-C. Hsu. "Work-in-Progress: Exploiting SIMD Capability in an ARMv7-to-ARMv8 Dynamic Binary Translator". In: *2018 International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES)*. 2018, pp. 1–3. DOI: 10.1109/CASES.2018.8516794.

[131] D. A. Patterson and C. H. Sequin. "A VLSI RISC". In: *Computer* 15.9 (1982), pp. 8–22.

[132] R.-V. Foundation. *RISC-V Foundation Announces Ratification of the RISC-V Base ISA and Privileged Architecture Specifications*. https://riscv.org/announcements/2019/07/risc-v-foundation-announces-ratification-of-the-risc-v-base-isa-and-privileged-architecture-specifications/. Accessed: 2021-01-07.

[133] D. Dechev. "The ABA problem in multicore data structures with collaborating operations". In: *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 7th International Conference on*. IEEE. 2011, pp. 158–167.

[134] E. D. Reilly. *Milestones in Computer Science and Information Technology*. USA: Greenwood Publishing Group Inc., 2003. ISBN: 1573565210.

[135] I. S. Sparc International and S. International. *The SPARC Architecture Manual: Version 8*. SPARC International. Prentice Hall, 1992. ISBN: 9780138250010. URL: https://books.google.nl/books?id=Qo1QAAAAMAAJ.

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

[137] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, and L. Benini. "Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25.10 (Oct. 2017), pp. 2700–2713. DOI: 10.1109/TVLSI.2017.2654506.

[138] D. C. Doolan, S. Tabirca, and L. T. Yang. "Mobile Parallel Computing". In: *2006 Fifth International Symposium on Parallel and Distributed Computing*. 2006, pp. 161–167.

[139] V. Costan, I. A. Lebedev, and S. Devadas. "Sanctum: Minimal Hardware Extensions for Strong Software Isolation." In: *USENIX Security Symposium*. 2016, pp. 857–874.

[140] A. Menon, S. Murugan, C. Rebeiro, N. Gala, and K. Veezhinathan. “Shakti-T: A RISC-V Processor with Light Weight Security Extensions”. In: *Proceedings of the Hardware and Architectural Support for Security and Privacy*. HASP ’17. Toronto, ON, Canada: ACM, 2017. DOI: 10.1145/3092627.3092629.

[141] F. Bezerra, D. Dangla, F. Manni, J. Mekki, D. Standarovski, R. G. Alia, M. Brugger, and S. Danzeca. “Evaluation of an Alternative Low Cost Approach for SEE Assessment of a SoC”. In: *2017 17th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*. Oct. 2017, pp. 1–5. DOI: 10.1109/RADECS.2017.8696219.

[142] M. Maniatakos, M. K. Michael, and Y. Makris. “Investigating the limits of AVF analysis in the presence of multiple bit errors”. In: *2013 IEEE 19th International On-Line Testing Symposium (IOLTS)*. July 2013, pp. 49–54. DOI: 10.1109/IOLTS.2013.6604050.

[143] A. Chatzidimitriou, P. Bodmann, G. Papadimitriou, D. Gizopoulos, and P. Rech. “Demystifying Soft Error Assessment Strategies on ARM CPUs: Microarchitectural Fault Injection vs. Neutron Beam Experiments”. In: *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. June 2019, pp. 26–38. DOI: 10.1109/DSN.2019.00018.

[144] S.-H. Jeon, J.-H. Cho, Y. Jung, S. Park, and T.-M. Han. “Automotive hardware development according to ISO 26262”. In: *13th International Conference on Advanced Communication Technology (ICACT2011)*. 2011, pp. 588–592.

[145] B. Kosinski and K. Dodson. “Key attributes to achieving >99.99 satellite availability”. In: *2018 IEEE International Reliability Physics Symposium (IRPS)*. Mar. 2018, 6A.3-1-6A.3–10. DOI: 10.1109/IRPS.2018.8353620.

[146] D. Oliveira, P. Navaux, and P. Rech. “Increasing the Efficiency and Efficacy of Selective-Hardening for Parallel Applications”. In: *2019 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. Oct. 2019, pp. 1–6. DOI: 10.1109/DFT.2019.8875300.

[147] R. Boi, S. Brigati, F. Francesconi, C. Ghidini, P. Malcovati, F. Maloberti, and M. Poletti. “Switched-capacitor Litton-code matched filter for satellite ODBH bus”. In: *ISCAS’99. Proceedings of the 1999 IEEE International Symposium on Circuits and Systems VLSI (Cat. No.99CH36349)*. Vol. 2. May 1999, 69–72 vol.2. DOI: 10.1109/ISCAS.1999.780621.

[148] C. Henry. *Geostationary satellite orders bouncing back*. 2020. URL: https://spacenews.com/geostationary-satellite-orders-bouncing-back/.

[149] B. Lal, E. Sylak-Glassman, M. Mineiro, N. Gupta, L. Pratt, and A. Azari. “Global trends in space volume 2: Trends by subsector and factors that could disrupt them”. In: *Institute for Defense Analyses, Science & Technology Policy Institute, IDA Paper P-5242* 2 (2015).

[150] ESA. *Onboard Computers*. https://www.esa.int/Enabling_Support/Space_Engineering_Technology/Onboard_Computers_and_Data_Handling/Onboard_Computers. 2021.

[151] D. Selva and D. Krejci. “A survey and assessment of the capabilities of Cubesats for Earth observation”. In: *Acta Astronautica* 74 (2012), pp. 50–68. ISSN: 0094-5765. DOI: https://doi.org/10.1016/j.actaastro.2011.12.014.

[152] S. Tsitas and J. Kingston. “6U CubeSat design for Earth observation with 6.5m GSD, five spectral bands and 14Mbps downlink”. In: *The Aeronautical Journal* 114.1161 (2010), pp. 689–697.

[153] EEMBC. *ARM Cortex-A9 (Exynos4 Quad) CoreMark Score*. https://www.eembc.org/viewer/?benchmark_seq=1484. Accessed: 2021-01-07.

[154] EEMBC. *Intel i7-3612QE CoreMark Score*. https://www.eembc.org/viewer/?benchmark_seq=1461. Accessed: 2021-01-07.

[155] ISISPACE. *ISIS On Board Computer (iOBC) Brochure*. https://www.isispace.nl/wp-content/uploads/2016/02/ISIS-On-board-Computer-Brochure-v2-compressed.pdf. Accessed: 2021-01-07.

[156] AAC Clyde Space. *Sirius OBC LEON3FT*. https://www.aac-clyde.space/assets/000/000/106/SIRIUS-OBC-LEON3FT_original.pdf. Accessed: 2021-01-07.

[157] Nasa. *Command and Data Handling*. https://www.nasa.gov/smallsat-institute/sst-soa-2020/command-and-data-handling. Accessed: 2021-01-07.

[158] EEMBC. *Atmel AT91SAM9G20 CoreMark Score*. https://www.eembc.org/viewer/?benchmark_seq=1111. Accessed: 2021-01-07.

[159] GomSpace. *NanoMind3200 User manual*. https://gomspace.com/UserFiles/Subsystems/datasheet/gs-ds-nanomind-a3200_1006901.pdf. Accessed: 2021-01-07.

[160] AAC Microtec. *Sirius OBC and TCM User Manual Rev. O*. http://repo.aacmicrotec.com/manuals/previous_versions/205065%20Sirius%20Product%20User%20Manual%20Rev%20O.pdf. 2019.

[161] Cobham Gaisler. *LEON3 Processor*. https://www.gaisler.com/index.php/products/processors/leon3.

[162] Airbus. *OSCAR OBC*. https://spaceequipment.airbusdefenceandspace.com/wp-content/uploads/2018/04/OSCAR_GB_2018.pdf. 2018.

[163] G. Furano. “Issues in (very) rad hard systems: an ESA perspective on use of COTS and space grade in JUICE mission”. In: *JUICE Instrument Workshop*. 2011.

[164] European Space Agency. *GOCE*. https://earth.esa.int/web/eoportal/satellite-missions/g/goce. Accessed: 2018-12-21.

[165] RUAG Space AB. *Command & Data Handling*. https://www.ruag.com/system/files/media_document/2017-12/Command%20and%20Data%20Handling.pdf. Accessed: 2021-01-07. 2013.

[166] European Space Agency. *CDF Study Report - GaiaNIR - Study to Enlarge the Achievements of Gaia with NIR Survey*. https://sci.esa.int/documents/34923/36148/1567260328914-GaiaNIR_CDF_study_report_v2.pdf. Accessed: 2021-01-07. 2017.

[167] J.-L. Poupat, A. Lefevre, and F. Koebel. "OSCAR: A Compact, Powerful and Versatile On Board Computer Based on LEON3 Core". In: *DASIA 2011-Data Systems In Aerospace* 694 (2011), p. 28.

[168] R. Heckmann, M. Langenbach, S. Thesing, and R. Wilhelm. "The influence of processor architecture on the design and the results of WCET tools". In: *Proceedings of the IEEE* 91.7 (2003), pp. 1038–1054. DOI: 10.1109/JPROC.2003.814618.

[169] ECSS. *ECSS-E-ST-40C - Space engineering - Software*. 2009.

[170] J. Garrido, D. Brosnan, J. A. de la Puente, A. Alonso, and J. Zamorano. "Analysis of WCET in an experimental satellite software development". In: *12th International Workshop on Worst-Case Execution Time Analysis*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2012.

[171] European Space Agency. *Remote Terminal Units*. https://www.esa.int/Enabling_Support/Space_Engineering_Technology/Onboard_Computers_and_Data_Handling/Remote_Terminal_Units. Accessed: 2018-12-21.

[172] M. Angulo, J. M. Mi, P. de Vicente, M. Prieto, O. Rodriguez, E. de la Fuente, and J. Palau. "Development of the MicroSat programme at INTA". In: *Small Satellites for Earth Observation*. Springer, 2008, pp. 41–54.

[173] M. Fossion, A. Van Esbeen, T. Van Humbeeck, Y. Geerts, E. Geukens, S. Redant, R. Jansen, and C. Monteleone. "A Mixed-Signal Radhard Microcontroller: the Digital Programmable Controller (DPC)". In: *AMICSA 2014* ().

[174] eoPortal. *Delfi-C3 (Triple-unit CubeSat Configuration of TU Delft)*. Accessed: 2021-01-07. URL: https://directory.eoportal.org/web/eoportal/satellite-missions/d/delfi-c3.

[175] F. Bruhn, P. Selin, I. Kalnins, J. Lyke, J. Rosengren-Calixte, and R. Nordenberg. "Quadsat/pnp: A space-plug-and-play architecture (spa) compliant nanosatellite". In: *Infotech@ Aerospace 2011*. 2011, p. 1575.

[176] A. Pullini, D. Rossi, I. Loi, A. D. Mauro, and L. Benini. "Mr. Wolf: A 1 GFLOP/s Energy-Proportional Parallel Ultra Low Power SoC for IOT Edge Processing". In: *ESSCIRC 2018 - IEEE 44th European Solid State Circuits Conference (ESSCIRC)*. Sept. 2018, pp. 274–277. DOI: 10.1109/ESSCIRC.2018.8494247.

[177] A. S. Waterman. "Design of the RISC-V instruction set architecture". PhD thesis. UC Berkeley, 2016.

[178] Z. Cao, Q. Lv, Y. Wang, M. Wen, N. Wu, and C. Zhang. "A Compression Instruction Set Design based on RISC-V for Network Packet Forwarding". In: *Journal of Physics: Conference Series*. Vol. 1026. 1. IOP Publishing. 2018, p. 012001.

[179] S. S. Mukherjee, M. Kontz, and S. K. Reinhardt. “Detailed design and evaluation of redundant multi-threading alternatives”. In: *Proceedings 29th Annual International Symposium on Computer Architecture*. May 2002, pp. 99–110. DOI: 10.1109/ISCA.2002.1003566.

[180] A. G. Dean. “Efficient Real-Time Fine-Grained Concurrency on Low-Cost Microcontrollers”. In: *IEEE Micro* 24 (July 2004), pp. 10–22. ISSN: 0272-1732. DOI: 10.1109/MM.2004.27.

[181] H. Leppinen. “Current use of linux in spacecraft flight software”. In: *IEEE Aerospace and Electronic Systems Magazine* 32.10 (2017), pp. 4–13. DOI: 10.1109/MAES.2017.160182.

[182] European Space Agency. *PhiSat-1 Nanosatellite Mission*. https://directory.eoportal.org/web/eoportal/satellite-missions/p/phisat-1. Accessed: 2021-01-07.

[183] G. Giuffrida, L. Diana, F. de Gioia, G. Benelli, G. Meoni, M. Donati, and L. Fanucci. “CloudScout: A Deep Neural Network for On-Board Cloud Detection on Hyperspectral Images”. In: *Remote Sensing* 12.14 (2020). DOI: 10.3390/rs12142205.

[184] B. Barry, C. Brick, F. Connor, D. Donohoe, D. Moloney, R. Richmond, M. O’Riordan, and V. Toma. “Always-on vision processing unit for mobile applications”. In: *IEEE Micro* 35.2 (2015), pp. 56–66.

[185] *UB0100 CubeSat Board for AI and Computer Vision Acceleration*. https://ubotica.com/ub0100/. Accessed: 2021-01-07.

[186] K. Abdelouahab, M. Pelcat, J. Sérot, and F. Berry. “Accelerating CNN inference on FPGAs: A Survey”. In: *CoRR* abs/1806.01683 (2018). arXiv: 1806.01683. URL: http://arxiv.org/abs/1806.01683.

[187] H.-C. Ng, C. Liu, and H. K.-H. So. “A soft processor overlay with tightly-coupled FPGA accelerator”. In: *arXiv preprint arXiv:1606.06483* (2016).

[188] T. A. K. A.-H. Aporva, A. S. D. S. Davidson, P. G. G. L. A. Rao, A. R. N. S. C. Torng, L. V. B. V. S. Xie, and C. Z. R. Zhao. “Experiences Using the RISC-V Ecosystem to Design an Accelerator-Centric SoC in TSMC 16nm”. In: (2017).

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

[190] D. Moloney. “Embedded deep neural networks: “The cost of everything and the value of nothing””. In: *2016 IEEE Hot Chips 28 Symposium (HCS)*. 2016, pp. 1–20. DOI: 10.1109/HOTCHIPS.2016.7936219.

[191] Y. Lee, A. Waterman, R. Avizienis, H. Cook, C. Sun, V. Stojanović, and K. Asanović. “A 45nm 1.3GHz 16.7 double-precision GFLOPS/W RISC-V processor with vector accelerators”. In: *ESSCIRC 2014 - 40th European Solid State Circuits Conference (ESSCIRC)*. Sept. 2014, pp. 199–202. DOI: 10.1109/ESSCIRC.2014.6942056.

[192] Adapteva. *E16G301 EPIPHANY 16-CORE MICROPROCESSOR Datasheet*. 2013.

[193] Adapteva. *E64G401 EPIPHANY 64 CORE MICROPROCESSOR Datasheet*. 2013.

[194] A. Olofsson. "Epiphany-V: A 1024 processor 64-bit RISC system-on-chip". In: *arXiv preprint arXiv:1610.01832* (2016).

[195] S. Steffl and S. Reda. "LACore: A Supercomputing-Like Linear Algebra Accelerator for SoC-Based Designs". In: *2017 IEEE International Conference on Computer Design (ICCD)*. Nov. 2017, pp. 137–144. DOI: 10.1109/ICCD.2017.29.

[196] S. Steffl and S. Reda. "A RISC-V Based Linear Algebra Accelerator for SoC Designs Samuel Steffl, Brown University". In: *7th RISC-V Workshop Proceedings*. 2017.

[197] V. Aggarwal, Y. Sabharwal, R. Garg, and P. Heidelberger. "HPCC RandomAccess benchmark for next generation supercomputers". In: *2009 IEEE International Symposium on Parallel Distributed Processing*. May 2009, pp. 1–11. DOI: 10.1109/IPDPS.2009.5161019.

[198] S. Williams, A. Waterman, and D. Patterson. "Roofline: An Insightful Visual Performance Model for Multicore Architectures". In: *Commun. ACM* 52.4 (Apr. 2009), pp. 65–76. ISSN: 0001-0782. DOI: 10.1145/1498765.1498785.

[199] G. Ofenbeck, R. Steinmann, V. Caparros, D. G. Spampinato, and M. Püschel. "Applying the roofline model". In: *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. Mar. 2014, pp. 76–85. DOI: 10.1109/ISPASS.2014.6844463.

[200] M. Parker. "A case for user-level interrupts". In: *ACM SIGARCH Computer Architecture News* 30.3 (2002), p. 17.

[201] M. Clark. "rv 8 : a high performance RISC-V to x 86 binary translator". In: 2017.

[202] A. Roelke and M. R. Stan. "Risc5: Implementing the RISC-V ISA in gem5". In: *First Workshop on Computer Architecture Research with RISC-V (CARRV)*. 2017.

[203] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, et al. "The gem5 simulator". In: *ACM SIGARCH Computer Architecture News* 39.2 (2011), pp. 1–7.

[204] T. Ta, L. Cheng, and C. Batten. "Simulating Multi-Core RISC-V Systems in gem5". In: *Workshop on Computer Architecture Research with RISC-V*. 2018.

[205] T. Schuster, R. Meyer, R. Buchty, L. Fossati, and M. Berekovic. "SoCRocket - A virtual platform for the European Space Agency's SoC development". In: *2014 9th International Symposium on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*. May 2014, pp. 1–7. DOI: 10.1109/ReCoSoC.2014.6860690.

[206] V. Herdt, D. Groβe, H. M. Le, and R. Drechsler. "Extensible and configurable RISC-V based virtual prototype". In: *2018 Forum on Specification & Design Languages (FDL)*. IEEE. 2018, pp. 5–16.

[207] A. Charif, G. Busnot, R. Mameesh, T. Sassolas, and N. Ventroux. "Fast virtual prototyping for embedded computing systems design and exploration". In: *Proceedings of the Rapid Simulation and Performance Evaluation: Methods and Tools*. 2019, pp. 1–8.

[208] F. Cordero, J. Mendes, B. Kuppusamy, T. Dathe, M. Irvine, and A. Williams. “A cost-effective Software Development and Validation environment and approach for LEON based satellite amp;amp; payload subsystems”. In: *Proceedings of 5th International Conference on Recent Advances in Space Technologies - RAST2011*. June 2011, pp. 511–516. DOI: 10.1109/RAST.2011.5966889.

[209] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović. “Chisel: Constructing hardware in a Scala embedded language”. In: *DAC Design Automation Conference 2012*. June 2012, pp. 1212–1221. DOI: 10.1145/2228360.2228584.

[210] S. Habinc and P. Sinander. “Using VHDL for board level simulation”. In: *IEEE Design Test of Computers* 13.3 (1996), pp. 66–78. ISSN: 0740-7475. DOI: 10.1109/54.536097.

[211] W. Simon, A. C. Yüzügüler, A. Ibrahim, F. Angiolini, M. Arditi, J. .-. Thiran, and G. De Micheli. “Single-FPGA, scalable, low-power, and high-quality 3D ultrasound beamformer”. In: *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*. 2016, pp. 1–2. DOI: 10.1109/FPL.2016.7577391.

[212] D. S. Lee, G. R. Allen, G. Swift, M. Cannon, M. Wirthlin, J. S. George, R. Koga, and K. Huey. “Single-Event Characterization of the 20 nm Xilinx Kintex Ultra-Scale Field-Programmable Gate Array under Heavy Ion Irradiation”. In: *2015 IEEE Radiation Effects Data Workshop (REDW)*. 2015, pp. 1–6. DOI: 10.1109/REDW.2015.7336736.

[213] F. G. de Lima Kastensmidt, G. Neuberger, R. F. Hentschke, L. Carro, and R. Reis. “Designing fault-tolerant techniques for SRAM-based FPGAs”. In: *IEEE Design Test of Computers* 21.6 (2004), pp. 552–562. DOI: 10.1109/MDT.2004.85.

[214] P. Maillard, M. Hart, J. Barton, P. Chang, M. Welter, R. Le, R. Ismail, and E. Crabill. “Single-Event Upsets Characterization & Evaluation of Xilinx UltraScale™ Soft Error Mitigation (SEM IP) Tool”. In: *2016 IEEE Radiation Effects Data Workshop (REDW)*. July 2016, pp. 1–4. DOI: 10.1109/NSREC.2016.7891745.

[215] E. Kyriakakis, K. Ngo, and J. Öberg. “Implementation of a fault-tolerant, globally-asynchronous-locally-synchronous, inter-chip NoC communication bridge on FPGAs”. In: *2017 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC)*. 2017, pp. 1–6. DOI: 10.1109/NORCHIP.2017.8124972.

[216] *IGLOO2 and SmartFusion2 65nm Commercial Flash FPGAs - Interim Summary of Radiation Test Results*. https://www.microsemi.com/document-portal/doc_download/134103-igloo2-and-smartfusion2-fpgas-interim-radiation-report.

[217] T. D. Loveless, L. W. Massengill, W. T. Holman, B. L. Bhuva, D. McMorrow, and J. H. Warner. “A Generalized Linear Model for Single Event Transient Propagation in Phase-Locked Loops”. In: *IEEE Transactions on Nuclear Science* 57.5 (2010), pp. 2933–2947. DOI: 10.1109/TNS.2010.2066287.

[218] S. C. Davis, R. Koga, and J. S. George. “Proton and Heavy Ion Testing of the Microsemi Igloo2 FPGA”. In: *2017 IEEE Radiation Effects Data Workshop (REDW)*. July 2017, pp. 1–6. DOI: 10.1109/NSREC.2017.8115454.

[219] N. Rezzak, J. Wang, D. Dsilva, and N. Jat. “TID and SEE Characterization of Microsemi’s 4th Generation Radiation Tolerant RTG4 Flash-Based FPGA”. In: *2015 IEEE Radiation Effects Data Workshop (REDW)*. July 2015, pp. 1–6. DOI: 10.1109/REDW.2015.7336739.

[220] Xilinx. *Radiation-Hardened, Space-Grade Virtex-5QV Family Data Sheet: Overview*. 2018. URL: https://www.xilinx.com/support/documentation/data_sheets/ds192_V5QV_Device_Overview.pdf.

[221] A. Haran, J. Barak, D. David, E. Keren, N. Refaeli, and S. Rapaport. “Single Event Hard Errors in SRAM Under Heavy Ion Irradiation”. In: *IEEE Transactions on Nuclear Science* 61.5 (Oct. 2014), pp. 2702–2710. ISSN: 0018-9499. DOI: 10.1109/TNS.2014.2345697.

[222] P. Roche, J. Autran, G. Gasiot, and D. Munteanu. “Technology downscaling worsening radiation effects in bulk: SOI to the rescue”. In: *2013 IEEE International Electron Devices Meeting*. Dec. 2013, pp. 31.1.1–31.1.4. DOI: 10.1109/IEDM.2013.6724728.

[223] J. Karp, M. J. Hart, P. Maillard, G. Hellings, and D. Linten. “Single-Event Latch-Up: Increased Sensitivity From Planar to FinFET”. In: *IEEE Transactions on Nuclear Science* 65.1 (2018), pp. 217–222. DOI: 10.1109/TNS.2017.2779831.

[224] S. Redant, R. Marec, L. Baguena, E. Liegeon, J. Soucarre, B. V. Thielen, G. Beeckman, P. Ribeiro, A. Fernandez-Leon, and B. Glass. “Radiation test results on first silicon in the design against radiation effects (DARE) library”. In: *IEEE Transactions on Nuclear Science* 52.5 (Oct. 2005), pp. 1550–1554. ISSN: 0018-9499. DOI: 10.1109/TNS.2005.855818.

[225] S. Clerc, F. Abouzeid, G. Gasiot, J. Daveau, C. Bottoni, M. Glorieux, J. Autran, F. Cacho, V. Huard, L. Dugoujon, R. Weigand, F. Malou, L. Hili, and P. Roche. “Space radiation and reliability qualifications on 65nm CMOS 600MHz microprocessors”. In: *2013 IEEE International Reliability Physics Symposium (IRPS)*. Apr. 2013, pp. 6C.1.1–6C.1.7. DOI: 10.1109/IRPS.2013.6532051.

[226] Department of Defense. *MIL-PRF-38535J, Performance Specification, Integrated Circuits (Microcircuits) Manufacturing, General Specification for*. 2010.

[227] Microchip. *ATmegaS128 Datasheet*. https://ww1.microchip.com/downloads/en/DeviceDoc/atmegas128_ds.pdf.

[228] A. (Microchip). *AT697F Datasheet*. https://ww1.microchip.com/downloads/en/DeviceDoc/doc7703.pdf.

[229] ECSS. “ECSS-E-HB-11A Technology Readiness Level (TRL) guidelines,” in: (2016).

[230] European Space Agency. *LEON’s First Flights*. http://www.esa.int/Our_Activities/Space_Engineering_Technology/LEON_s_first_flights. Accessed: 2018-12-21.

[231] S. Di Mascio, A. Menicucci, E. Gill, G. Furano, and C. Monteleone. “Open-source IP cores for space: A processor-level perspective on soft errors in the RISC-V era”. In: *Computer Science Review* 39 (2021), p. 100349. ISSN: 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2020.100349.

[232] S. Di Mascio, A. Menicucci, E. Gill, G. Furano, and C. Monteleone. “On the Criticality of Caches in Fault Tolerant Processors for Space”. In: *2019 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. Oct. 2019.

[233] H. Cho, E. Cheng, T. Shepherd, C. Cher, and S. Mitra. “System-Level Effects of Soft Errors in Uncore Components”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36.9 (Sept. 2017), pp. 1497–1510. ISSN: 0278-0070. DOI: 10.1109/TCAD.2017.2651824.

[234] M. Fazeli, A. Namazi, and S. G. Miremadi. “An energy efficient circuit level technique to protect register file from MBUs and SETs in embedded processors”. In: *2009 IEEE/IFIP International Conference on Dependable Systems Networks*. June 2009, pp. 195–204. DOI: 10.1109/DSN.2009.5270337.

[235] S. Wang, J. Hu, and S. G. Ziavras. “Replicating Tag Entries for Reliability Enhancement in Cache Tag Arrays”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 20.4 (2012), pp. 643–654. ISSN: 1063-8210. DOI: 10.1109/TVLSI.2011.2111469.

[236] E. Cheng, S. Mirkhani, L. G. Szafaryn, C.-Y. Cher, H. Cho, K. Skadron, M. R. Stan, K. Lilja, J. A. Abraham, P. Bose, and S. Mitra. “CLEAR: Cross-Layer Exploration for Architecting Resilience - Combining Hardware and Software Techniques to Tolerate Soft Errors in Processor Cores”. In: *Proceedings of the 53rd Annual Design Automation Conference*. DAC ’16. Austin, Texas: Association for Computing Machinery, 2016. DOI: 10.1145/2897937.2897996.

[237] A. Savino, A. Vallero, and S. Di Carlo. “ReDO: Cross-Layer Multi-Objective Design-Exploration Framework for Efficient Soft Error Resilient Systems”. In: *IEEE Transactions on Computers* 67.10 (2018), pp. 1462–1477.

[238] J. A. Butts and G. Sohi. “Dynamic dead-instruction detection and elimination”. In: *ACM SIGPLAN Notices* 37.10 (2002), pp. 199–210.

[239] H. Cho, S. Mirkhani, C. Cher, J. A. Abraham, and S. Mitra. “Quantitative evaluation of soft error injection techniques for robust system design”. In: *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*. May 2013, pp. 1–10.

[240] S. S. Mukherjee, J. Emer, and S. K. Reinhardt. “The soft error problem: an architectural perspective”. In: *11th International Symposium on High-Performance Computer Architecture*. Feb. 2005, pp. 243–247. DOI: 10.1109/HPCA.2005.37.

[241] G. S. Rodrigues, F. Rosa, Á. B. de Oliveira, F. L. Kastensmidt, L. Ost, and R. Reis. “Analyzing the Impact of Fault-Tolerance Methods in ARM Processors Under Soft Errors Running Linux and Parallelization APIs”. In: *IEEE Transactions on Nuclear Science* 64.8 (Aug. 2017), pp. 2196–2203. ISSN: 1558-1578. DOI: 10.1109/TNS.2017.2706519.

[242] H. Cho. “Impact of Microarchitectural Differences of RISC-V Processor Cores on Soft Error Effects”. In: *IEEE Access* 6 (2018), pp. 41302–41313.

[243] M. A. Breuer. “Multi-media applications and imprecise computation”. In: *8th Euromicro Conference on Digital System Design (DSD'05)*. Aug. 2005, pp. 2–7. DOI: 10.1109/DSD.2005.58.

[244] D. D. Thaker, D. Franklin, J. Oliver, S. Biswas, D. Lockhart, T. Metodi, and F. T. Chong. “Characterization of Error-Tolerant Applications when Protecting Control Data”. In: *2006 IEEE International Symposium on Workload Characterization*. Oct. 2006, pp. 142–149. DOI: 10.1109/IISWC.2006.302738.

[245] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler. “Understanding Error Propagation in Deep Learning Neural Network (DNN) Accelerators and Applications”. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '17. Denver, Colorado: Association for Computing Machinery, 2017. DOI: 10.1145/3126908.3126964.

[246] J. Perez Acle, M. S. Reorda, and M. Violante. “Implementing a safe embedded computing system in SRAM-based FPGAs using IP cores: A case study based on the Altera NIOS-II soft processor”. In: *2011 IEEE Second Latin American Symposium on Circuits and Systems (LASCAS)*. Feb. 2011, pp. 1–5. DOI: 10.1109/LASCAS.2011.5750278.

[247] N. J. Wang and S. J. Patel. “ReStore: symptom based soft error detection in microprocessors”. In: *2005 International Conference on Dependable Systems and Networks (DSN'05)*. June 2005, pp. 30–39. DOI: 10.1109/DSN.2005.82.

[248] D. S. Kim, S. M. Lee, J.-H. Jung, T. H. Kim, S. Lee, and J. S. Park. “Reliability and availability analysis for an on board computer in a satellite system using standby redundancy and rejuvenation”. In: *Journal of Mechanical Science and Technology* 26.7 (July 2012), pp. 2059–2063. ISSN: 1976-3824. DOI: 10.1007/s12206-012-0512-6.

[249] F. M. David, J. C. Carlyle, E. M. Chan, D. K. Raila, and R. H. Campbell. “Exception Handling in the Choices Operating System”. In: *Advanced Topics in Exception Handling Techniques*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 42–61. ISBN: 978-3-540-37445-9. DOI: 10.1007/11818502_3.

[250] I. Joe and S. C. Lee. “Bootup time improvement for embedded Linux using snapshot images created on boot time”. In: *The 2nd International Conference on Next Generation Information Technology*. June 2011, pp. 193–196.

[251] N. N. Sadler and D. J. Sorin. “Choosing an Error Protection Scheme for a Microprocessor's L1 Data Cache”. In: *2006 International Conference on Computer Design*. Oct. 2006, pp. 499–505. DOI: 10.1109/ICCD.2006.4380862.

[252] M. Maniatakos, P. Kudva, B. M. Fleischer, and Y. Makris. “Low-Cost Concurrent Error Detection for Floating-Point Unit (FPU) Controllers”. In: *IEEE Transactions on Computers* 62.7 (July 2013), pp. 1376–1388. ISSN: 0018-9340. DOI: 10.1109/TC.2012.81.

[253] S. Tselonis, M. Kaliorakis, N. Foutris, G. Papadimitriou, and D. Gizopoulos. “Microprocessor reliability-performance tradeoffs assessment at the microarchitecture level”. In: *2016 IEEE 34th VLSI Test Symposium (VTS)*. Apr. 2016, pp. 1–6. DOI: 10.1109/VTS.2016.7477300.

[254] X. Fu, T. Li, and J. Fortes. “Sim-SODA: Unified framework for architectural level software reliability analysis”. In: *Workshop on Modeling, Benchmarking and Simulation, 2006*. 2006.

[255] A. Chatzidimitriou and D. Gizopoulos. “Anatomy of microarchitecture-level reliability assessment: Throughput and accuracy”. In: *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. Apr. 2016, pp. 69–78. DOI: 10.1109/ISPASS.2016.7482075.

[256] L. Duan, L. Peng, and B. Li. “Predicting Architectural Vulnerability on Multithreaded Processors under Resource Contention and Sharing”. In: *IEEE Transactions on Dependable and Secure Computing* 10.2 (Mar. 2013), pp. 114–127. ISSN: 1545-5971. DOI: 10.1109/TDSC.2012.87.

[257] S. Gayathri and T. C. Taranath. “RTL synthesis of case study using design compiler”. In: *2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT)*. 2017, pp. 1–7. DOI: 10.1109/ICEECCOT.2017.8284603.

[258] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi. “CACTI 6.0: A tool to model large caches”. In: *HP laboratories* 27 (2009), p. 28.

[259] A. Naithani, S. Eyerman, and L. Eeckhout. “Optimizing Soft Error Reliability Through Scheduling on Heterogeneous Multicore Processors”. In: *IEEE Transactions on Computers* 67.6 (June 2018), pp. 830–846. ISSN: 0018-9340. DOI: 10.1109/TC.2017.2779480.

[260] L. Duan, Y. Zhang, B. Li, and L. Peng. “Comprehensive and Efficient Design Parameter Selection for Soft Error Resilient Processors via Universal Rules”. In: *IEEE Transactions on Computers* 63.9 (Sept. 2014), pp. 2201–2214. ISSN: 0018-9340. DOI: 10.1109/TC.2013.24.

[261] A. Biswas, C. Recchia, S. S. Mukherjee, V. Ambrose, L. Chan, A. Jaleel, A. E. Papathanasiou, M. Plaster, and N. Seifert. “Explaining cache SER anomaly using DUE AVF measurement”. In: *HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*. Jan. 2010, pp. 1–12. DOI: 10.1109/HPCA.2010.5416629.

[262] Y. Cheng, A. Ma, Y. Tang, and M. Zhang. “Accurate vulnerability estimation for cache hierarchy”. In: *The 7th International Conference on Networked Computing and Advanced Information Management*. June 2011, pp. 7–14.

[263] M. Kooli, G. Di Natale, and A. Bosio. “Memory-Aware Design Space Exploration for Reliability Evaluation in Computing Systems”. In: *Journal of Electronic Testing* 35.2 (Apr. 2019), pp. 145–162. ISSN: 1573-0727. DOI: 10.1007/s10836-019-05785-0.

[264] Xin Fu, J. Poe, Tao Li, and J. A. B. Fortes. "Characterizing Microarchitecture Soft Error Vulnerability Phase Behavior". In: *14th IEEE International Symposium on Modeling, Analysis, and Simulation*. Sept. 2006, pp. 147–155. DOI: 10.1109/MASCOTS.2006.18.

[265] Wangyuan Zhang and Tao Li. "Managing multi-core soft-error reliability through utility-driven cross domain optimization". In: *2008 International Conference on Application-Specific Systems, Architectures and Processors*. July 2008, pp. 132–137. DOI: 10.1109/ASAP.2008.4580167.

[266] T. M. Jones, M. F. O'boyle, and O. Ergin. "Evaluating the effects of compiler optimisations on AVF". In: *Workshop on interaction between compilers and computer architecture (INTERACT-12)*. 2008.

[267] G. E. Medeiros, F. T. Bortolon, R. Reis, and L. Ost. "Evaluation of Compiler Optimization Flags Effects on Soft Error Resiliency". In: *2018 31st Symposium on Integrated Circuits and Systems Design (SBCCI)*. 2018, pp. 1–6. DOI: 10.1109/SBCCI.2018.8533246.

[268] A. Serrano-Cases, Y. Morilla, P. Martín-Holgado, S. Cuenca-Asensi, and A. Martínez-Álvarez. "Nonintrusive Automatic Compiler-Guided Reliability Improvement of Embedded Applications Under Proton Irradiation". In: *IEEE Transactions on Nuclear Science* 66.7 (2019), pp. 1500–1509. DOI: 10.1109/TNS.2019.2912323.

[269] C. Cher, M. S. Gupta, P. Bose, and K. P. Muller. "Understanding Soft Error Resiliency of Blue Gene/Q Compute Chip through Hardware Proton Irradiation and Software Fault Injection". In: *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. Nov. 2014, pp. 587–596. DOI: 10.1109/SC.2014.53.

[270] Y. Chou, B. Fahs, and S. Abraham. "Microarchitecture Optimizations for Exploiting Memory-Level Parallelism". In: *Proceedings of the 31st Annual International Symposium on Computer Architecture*. ISCA '04. München, Germany: IEEE Computer Society, 2004, pp. 76–. ISBN: 0-7695-2143-6. URL: http://dl.acm.org/citation.cfm?id=998680.1006708.

[271] N. J. Wang, A. Mahesri, and S. J. Patel. "Examining ACE Analysis Reliability Estimates Using Fault-injection". In: *Proceedings of the 34th Annual International Symposium on Computer Architecture*. ISCA '07. San Diego, California, USA: ACM, 2007, pp. 460–469. ISBN: 978-1-59593-706-3. DOI: 10.1145/1250662.1250719.

[272] N. Wang and M. F. and. "Y-branches: when you come to a fork in the road, take it". In: *2003 12th International Conference on Parallel Architectures and Compilation Techniques*. Sept. 2003, pp. 56–66. DOI: 10.1109/PACT.2003.1238002.

[273] A. Biswas, P. Racunas, J. Emer, and S. Mukherjee. "Computing Accurate AVFs using ACE Analysis on Performance Models: A Rebuttal". In: *IEEE Computer Architecture Letters* 7.1 (Jan. 2008), pp. 21–24. ISSN: 2473-2575. DOI: 10.1109/L-CA.2007.19.

[274] K. Tanikella, Y. Koy, R. Jeyapaul, Kyoungwoo Lee, and A. Shrivastava. “gemV: A validated toolset for the early exploration of system reliability”. In: *2016 IEEE 27th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. 2016, pp. 159–163.

[275] P. Ramachandran, P. Kudva, J. Kellington, J. Schumann, and P. Sanda. “Statistical Fault Injection”. In: *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*. June 2008, pp. 122–127. DOI: 10.1109/DSN.2008.4630080.

[276] H. Ziade, R. Ayoubi, and R. Velazco. “A survey on fault injection techniques”. In: *International Arab Journal of Information Technology* Vol. 1, No. 2, July (2004), pp. 171–186.

[277] P. Montesinos, W. Liu, and J. Torrellas. “Using register lifetime predictions to protect register files against soft errors”. In: *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN’07)*. IEEE. 2007, pp. 286–296.

[278] A.-G. Ma, Y. Cheng, and Z.-C. Xing. “Accurate and Simplified Prediction of AVF for Delay and Energy Efficient Cache Design”. In: *Journal of Computer Science and Technology* 26.3 (May 2011), pp. 504–519. ISSN: 1860-4749. DOI: 10.1007/s11390-011-1150-7. URL: https://doi.org/10.1007/s11390-011-1150-7.

[279] J. A. Blome, S. Gupta, S. Feng, and S. Mahlke. “Cost-efficient Soft Error Protection for Embedded Microprocessors”. In: *Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*. CASES ’06. Seoul, Korea: ACM, 2006, pp. 421–431. ISBN: 1-59593-543-6. DOI: 10.1145/1176760.1176811.

[280] X. Li, S. V. Adve, Pradip Bose, and J. A. Rivers. “SoftArch: an architecture-level tool for modeling and analyzing soft errors”. In: *2005 International Conference on Dependable Systems and Networks (DSN’05)*. June 2005, pp. 496–505. DOI: 10.1109/DSN.2005.88.

[281] F. A. Endo, D. Couroussé, and H.-P. Charles. “Micro-architectural Simulation of Embedded Core Heterogeneity with Gem5 and McPAT”. In: *Proceedings of the 2015 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*. RAPIDO ’15. Amsterdam, Holland: ACM, 2015, 7:1–7:6. ISBN: 978-1-60558-699-1. DOI: 10.1145/2693433.2693440. URL: http://doi.acm.org/10.1145/2693433.2693440.

[282] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. “McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures”. In: *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Dec. 2009, pp. 469–480.

[283] Wei Zhang. “Computing cache vulnerability to transient errors and its implication”. In: *20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT’05)*. Oct. 2005, pp. 427–435. DOI: 10.1109/DFTVS.2005.23.

[284] D. L. Hansen, E. J. Miller, A. Kleinosowski, K. Kohnen, A. Le, D. Wong, K. Amador, M. Baze, D. DeSalvo, M. Dooley, K. Gerst, B. Hughlock, B. Jeppson, R. D. Jobe, D. Nardi, I. Ojalvo, B. Rasmussen, D. Sunderland, J. Truong, M. Yoo, and E. Zayas. "Clock, Flip-Flop, and Combinatorial Logic Contributions to the SEU Cross Section in 90 nm ASIC Technology". In: *IEEE Transactions on Nuclear Science* 56.6 (Dec. 2009), pp. 3542–3550. ISSN: 1558-1578. DOI: 10.1109/TNS.2009.2031972.

[285] Freescale. *"Safety Manual for Qorivva MPC5643L"*.

[286] F. Sturesson, J. Gaisler, R. Ginosar, and T. Liran. "Radiation characterization of a dual core LEON3-FT processor". In: *2011 12th European Conference on Radiation and Its Effects on Components and Systems*. Sept. 2011, pp. 938–944. DOI: 10.1109/RADECS.2011.6131334.

[287] P. N. Sanda, J. W. Kellington, P. Kudva, R. Kalla, R. B. McBeth, J. Ackaret, R. Lockwood, J. Schumann, and C. R. Jones. "Soft-error resilience of the IBM POWER6 processor". In: *IBM Journal of Research and Development* 52.3 (May 2008), pp. 275–284. ISSN: 0018-8646. DOI: 10.1147/rd.523.0275.

[288] S. Di Mascio, A. Menicucci, E. Gill, G. Furano, and C. Monteleone. "On-Board Decision Making in Space with Deep Neural Networks and RISC-V Vector Processors". In: *Journal of Aerospace Information Systems* 18.8 (2021), pp. 553–570. DOI: 10.2514/1.I010916.

[289] J. R. Schwank, M. R. Shaneyfelt, and P. E. Dodd. "Radiation Hardness Assurance Testing of Microelectronic Devices and Integrated Circuits: Radiation Environments, Physical Mechanisms, and Foundations for Hardness Assurance". In: *IEEE Transactions on Nuclear Science* 60.3 (2013), pp. 2074–2100.

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

[291] Z. Katona, M. Gräßlin, A. Donner, N. Kranich, H. Brandt, H. Bischl, and M. Brück. "A flexible LEO satellite modem with Ka-band RF frontend for a data relay satellite system". In: *International Journal of Satellite Communications and Networking* 38.3 (2020), pp. 301–313. DOI: https://doi.org/10.1002/sat.1333. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/sat.1333. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/sat.1333.

[292] G. Maral, M. Bousquet, and Z. Sun. *Satellite communications systems: systems, techniques and technology*. John Wiley & Sons, 2020.

[293] J. Radtke, C. Kebschull, and E. Stoll. "Interactions of the space debris environment with mega constellations—Using the example of the OneWeb constellation". In: *Acta Astronautica* 131 (2017), pp. 55–68. ISSN: 0094-5765. DOI: https://doi.org/10.1016/j.actaastro.2016.11.021. URL: http://www.sciencedirect.com/science/article/pii/S009457651630515X.

[294] C. OMeara, L. Schlag, and M. Wickler. "Applications of deep learning neural networks to satellite telemetry monitoring". In: *2018 SpaceOps Conference*. 2018, p. 2558.

[295] S. N. Goward, J. G. Masek, D. L. Williams, J. R. Irons, and R. Thompson. "The Landsat 7 mission: Terrestrial research and applications for the 21st century". In: *Remote Sensing of Environment* 78.1 (2001). Landsat 7, pp. 3–12. ISSN: 0034-4257. DOI: https://doi.org/10.1016/S0034-4257(01)00262-0.

[296] S. M. Guertin and M. Amrbar. "Single Event Testing of SDRAM, DDR2 and DDR3 Memories". In: *2016 IEEE Radiation Effects Data Workshop (REDW)*. 2016, pp. 1–7.

[297] ISSI. *"IS43/46DR81280B(L), IS43/46DR16640B(L) datasheet*. 2015. URL: http://www.issi.com/WW/pdf/43-46DR81280B-16640B.pdf.

[298] S. Bianco, R. Cadene, L. Celona, and P. Napoletano. "Benchmark Analysis of Representative Deep Neural Network Architectures". In: *IEEE Access* 6 (2018), pp. 64270–64277.

[299] S. Mohajerani and P. Saeedi. "Cloud-Net: An End-To-End Cloud Detection Algorithm for Landsat 8 Imagery". In: *IGARSS 2019 - 2019 IEEE International Geoscience and Remote Sensing Symposium*. 2019, pp. 1029–1032.

[300] M. Y. Saifi, J. Singla, and Nikita. "Deep Learning based Framework for Semantic Segmentation of Satellite Images". In: *2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC)*. 2020, pp. 369–374. DOI: 10.1109/ICCMC48092.2020.ICCMC-00069.

[301] M. Kartal and O. Duman. "Ship Detection from Optical Satellite Images with Deep Learning". In: *2019 9th International Conference on Recent Advances in Space Technologies (RAST)*. 2019, pp. 479–484. DOI: 10.1109/RAST.2019.8767844.

[302] D. Cappellone, S. Di Mascio, G. Furano, A. Menicucci, and M. Ottavi. "On-Board Satellite Telemetry Forecasting with RNN on RISC-V Based Multicore Processor". In: *2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. IEEE. 2020, pp. 1–6.

[303] V. Sze, Y. Chen, T. Yang, and J. S. Emer. "Efficient Processing of Deep Neural Networks: A Tutorial and Survey". In: *Proceedings of the IEEE* 105.12 (2017), pp. 2295–2329.

[304] C. Luo, X. Li, L. Wang, J. He, D. Li, and J. Zhou. "How Does the Data set Affect CNN-based Image Classification Performance?" In: *2018 5th International Conference on Systems and Informatics (ICSAI)*. 2018, pp. 361–366.

[305] P. Singh, N. Singh, K. K. Singh, and A. Singh. "Chapter 5 - Diagnosing of disease using machine learning". In: *Machine Learning and the Internet of Medical Things in Healthcare*. Ed. by K. K. Singh, M. Elhoseny, A. Singh, and A. A. Elngar. Academic Press, 2021, pp. 89–111. ISBN: 978-0-12-821229-5. DOI: https://doi.org/10.1016/B978-0-12-821229-5.00003-3. URL: https://www.sciencedirect.com/science/article/pii/B9780128212295000033.

[306] D. Phiri and J. Morgenroth. "Developments in Landsat land cover classification methods: A review". In: *Remote Sensing* 9.9 (2017), p. 967.

[307] T. Yairi, Y. Kawahara, R. Fujimaki, Y. Sato, and K. Machida. "Telemetry-mining: a machine learning approach to anomaly detection and fault diagnosis for space systems". In: *2nd IEEE International Conference on Space Mission Challenges for Information Technology (SMC-IT'06)*. 2006, 8 pp.–476. DOI: 10.1109/SMC-IT.2006.79.

[308] L. Li, M. Gariel, R. J. Hansman, and R. Palacios. "Anomaly detection in onboard-recorded flight data using cluster analysis". In: *2011 IEEE/AIAA 30th Digital Avionics Systems Conference*. 2011, 4A4-1-4A4–11. DOI: 10.1109/DASC.2011.6096068.

[309] E. Shelhamer, J. Long, and T. Darrell. "Fully Convolutional Networks for Semantic Segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.4 (2017), pp. 640–651.

[310] K. Chellapilla, S. Puri, and P. Simard. "High Performance Convolutional Neural Networks for Document Processing". In: *Tenth International Workshop on Frontiers in Handwriting Recognition*. Oct. 2006. URL: https://hal.inria.fr/inria-00112631.

[311] V. Dumoulin and F. Visin. "A guide to convolution arithmetic for deep learning". In: *arXiv preprint arXiv:1603.07285* (2016).

[312] A. Heinecke, K. Vaidyanathan, M. Smelyanskiy, A. Kobotov, R. Dubtsov, G. Henry, A. G. Shet, G. Chrysos, and P. Dubey. "Design and Implementation of the Linpack Benchmark for Single and Multi-node Systems Based on Intel® Xeon Phi Coprocessor". In: *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. 2013, pp. 126–137.

[313] K. He, X. Zhang, S. Ren, and J. Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[314] L. Lai, N. Suda, and V. Chandra. "Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus". In: *arXiv preprint arXiv:1801.06601* (2018).

[315] S. Ioffe and C. Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *arXiv preprint arXiv:1502.03167* (2015).

[316] E. Phaisangittisagul. "An Analysis of the Regularization Between L2 and Dropout in Single Hidden Layer Neural Network". In: *2016 7th International Conference on Intelligent Systems, Modelling and Simulation (ISMS)*. 2016, pp. 174–179.

[317] Keras. *BatchNormalization layer*. https://keras.io/api/layers/normalization_layers/batch_normalization/. Accessed: 2021-12-21.

[318] J. Cong and B. Xiao. "Minimizing Computation in Convolutional Neural Networks". In: *Artificial Neural Networks and Machine Learning – ICANN 2014*. Ed. by S. Wermter, C. Weber, W. Duch, T. Honkela, P. Koprinkova-Hristova, S. Magg, G. Palm, and A. E. P. Villa. Cham: Springer International Publishing, 2014, pp. 281–290. ISBN: 978-3-319-11179-7.

[319] K. Cho et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078* (2014).

[320] A. Graves. “Supervised Sequence Labelling with Recurrent Neural Networks [Ph. D. dissertation]”. In: *Technical University of Munich, Germany* (2008).

[321] A. Graves, A.-r. Mohamed, and G. Hinton. “Speech recognition with deep recurrent neural networks”. In: *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE. 2013, pp. 6645–6649.

[322] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon. “In-Datacenter Performance Analysis of a Tensor Processing Unit”. In: *SIGARCH Comput. Archit. News* 45.2 (June 2017), pp. 1–12. ISSN: 0163-5964. DOI: 10.1145/3140659.3080246.

[323] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. “A density-based algorithm for discovering clusters in large spatial databases with noise.” In: *kdd*. Vol. 96. 34. 1996, pp. 226–231.

[324] P. Blacker, C. P. Bridges, and S. Hadfield. “Rapid Prototyping of Deep Learning Models on Radiation Hardened CPUs”. In: *2019 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. 2019, pp. 25–32.

[325] L. Lai and N. Suda. “Enabling Deep Learning at the LoT Edge”. In: *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2018, pp. 1–6.

[326] G. Furano, J. Byrne, D. Moloney, A. Dunne, L. Buckley, M. Psarakis, A. Tavoularis, G. Meoni, and L. Fanucci. “Towards the use of Artificial Intelligence on the Edge in Space Systems: Challenges and Opportunities”. In: *IEEE AEROSPACE AND ELECTRONIC SYSTEMS MAGAZINE* (2020). Under review.

[327] E. Del Sozzo, A. Solazzo, A. Miele, and M. D. Santambrogio. “On the Automation of High Level Synthesis of Convolutional Neural Networks”. In: *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 2016, pp. 217–224.

[328] Y. Yao, K. Bhardwaj, P. Whatmough, G.-Y. Wei, D. Brooks, et al. “SMAUG: End-to-End Full-Stack Simulation Infrastructure for Deep Learning Workloads”. In: *arXiv preprint arXiv:1912.04481* (2019).

[329] D. Lopez, J. Llosa, E. Ayguade, and M. Valero. “Impact on performance of fused multiply-add units in aggressive VLIW architectures”. In: *Proceedings of the 1999 International Conference on Parallel Processing*. 1999, pp. 22–29.

[330] K. Asanovic and J. Wawrzynek. *Vector microprocessors*. University of California, Berkeley, 1998.

[331] S.-J. Lee, S.-S. Park, and K.-S. Chung. "Efficient SIMD Implementation for Accelerating Convolutional Neural Network". In: *Proceedings of the 4th International Conference on Communication and Information Processing*. ICCIP '18. Qingdao, China: Association for Computing Machinery, 2018, pp. 174–179. DOI: 10.1145/3290420.3290444.

[332] A. Peleg and U. Weiser. "MMX technology extension to the Intel architecture". In: *IEEE Micro* 16.4 (1996), pp. 42–50.

[333] S. Thakkur and T. Huff. "Internet Streaming SIMD Extensions". In: *Computer* 32.12 (1999), pp. 26–34.

[334] T. Shimizu. *Post-K Supercomputer Development*. 2019. URL: https://www.fujitsu.com/global/imagesgig5/post-k-supercomputer-development.pdf.

[335] Y. Lee, A. Ou, C. Schmidt, S. Karandikar, H. Mao, and K. Asanovic. "The Hwacha Microarchitecture Manual, Version 3.8." In: *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-263* (2015).

[336] *RISC-V "V" Vector Extension, Version 0.9*. Accessed on 02.07.2020. 2020. URL: https://github.com/riscv/riscv-v-spec/releases/download/0.9/riscv-v-spec-0.9.pdf.

[337] M. S. Louis, Z. Azad, L. Delshadtehrani, S. Gupta, P. Warden, V. J. Reddi, and A. Joshi. "Towards deep learning using tensorFlow lite on RISC-V". In: *Third Workshop on Computer Architecture Research with RISC-V (CARRV)*. 2019.

[338] C. Y. Lo and C.-W. Sham. "Energy Efficient Fixed-point Inference System of Convolutional Neural Network". In: *2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS)*. 2020, pp. 403–406. DOI: 10.1109/MWSCAS48704.2020.9184436.

[339] A. Ilic, F. Pratas, and L. Sousa. "Cache-aware Roofline model: Upgrading the loft". In: *IEEE Computer Architecture Letters* 13.1 (2014), pp. 21–24. DOI: 10.1109/L-CA.2013.6.

[340] S. Petit, J. P. David, D. Falguere, S. Duzellier, C. Inguimbert, T. Nuns, and R. Ecoffet. "Memories Response to MBU and Semi-Empirical Approach for SEE Rate Calculation". In: *IEEE Transactions on Nuclear Science* 53.4 (2006), pp. 1787–1793.

[341] A. Samaras, F. Bezerra, E. Lorfevre, and R. Ecoffet. "CARMEN-2: In flight observation of non destructive single event phenomena on memories". In: *2011 12th European Conference on Radiation and Its Effects on Components and Systems*. 2011, pp. 839–848.

[342] A. Bacchini, G. Furano, M. Rovatti, and M. Ottavi. "Total Ionizing Dose Effects on DRAM Data Retention Time". In: *IEEE Transactions on Nuclear Science* 61.6 (2014), pp. 3690–3693.

[343] A. Kumar and S. Sawitzki. "High-Throughput and Low-Power Architectures for Reed Solomon Decoder". In: *Conference Record of the Thirty-Ninth Asilomar Conference on Signals, Systems and Computers, 2005*. 2005, pp. 990–994.

[344] A. N. Udipi, N. Muralimanohar, N. Chatterjee, R. Balasubramonian, A. Davis, and N. P. Jouppi. "Rethinking DRAM Design and Organization for Energy-Constrained Multi-Cores". In: *SIGARCH Comput. Archit. News* 38.3 (June 2010), pp. 175–186. ISSN: 0163-5964. DOI: 10.1145/1816038.1815983. URL: https://doi.org/10.1145/1816038.1815983.

[345] Hanho Lee. "High-speed VLSI architecture for parallel Reed-Solomon decoder". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 11.2 (2003), pp. 288–294.

[346] Y. R. Shayan and T. Le-Ngoc. "A cellular structure for a versatile Reed-Solomon decoder". In: *IEEE Transactions on Computers* 46.1 (1997), pp. 80–85.

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

[348] J. D. Gee and A. J. Smith. *Vector Processor Caches*. Tech. rep. UCB/CSD-92-707. EECS Department, University of California, Berkeley, Oct. 1992. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/1992/6251.html.

[349] Quing Yang. "Introducing a new cache design into vector computers". In: *IEEE Transactions on Computers* 42.12 (1993), pp. 1411–1424.

[350] T. Abtahi, C. Shea, A. Kulkarni, and T. Mohsenin. "Accelerating Convolutional Neural Network With FFT on Embedded Hardware". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26.9 (2018), pp. 1737–1749.

[351] S. Wang, J. Hu, and S. G. Ziavras. "On the Characterization of Data Cache Vulnerability in High-Performance Embedded Microprocessors". In: *2006 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation*. 2006, pp. 14–20.

[352] *RISC-V Vector Extension Intrinsic Document*. Accessed: 2021-01-07. URL: https://github.com/riscv/rvv-intrinsic-doc.

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

[354] Cobham Gaisler AB. *GRLIB IP core user's manual*. Jun 2020, Version 2020.2. URL: https://www.gaisler.com/products/grlib/grip.pdf.

[355] A. Rico, J. A. Joao, C. Adeniyi-Jones, and E. Van Hensbergen. "ARM HPC Ecosystem and the Reemergence of Vectors: Invited Paper". In: *Proceedings of the Computing Frontiers Conference*. CF'17. Siena, Italy: Association for Computing Machinery, 2017, pp. 329–334. ISBN: 9781450344876. DOI: 10.1145/3075564.3095086. URL: https://doi.org/10.1145/3075564.3095086.

[356] Xilinx. *KCU105 Board - User Guide (UG917)*. 2019.

[357] *BLAS (Basic Linear Algebra Subprograms)*. http://www.netlib.org/blas/. Accessed: 2021-01-07.

[358] M. Nakata. “Basics and Practice of Linear Algebra Calculation Library BLAS and LAPACK”. In: *The Art of High Performance Computing for Computational Science, Vol. 1: Techniques of Speedup and Parallelization for General Purposes*. Ed. by M. Geshi. Singapore: Springer Singapore, 2019, pp. 83–112. ISBN: 978-981-13-6194-4. DOI: 10.1007/978-981-13-6194-4_6. URL: https://doi.org/10.1007/978-981-13-6194-4_6.

[359] J. Labarta. “The RISC-V Vector Processor in EPI”. In: *3rd RISC-V Meeting*. March 2021. URL: https://open-src-soc.org/2021-03/media/slides/3rd-RISC-V-Meeting-2021-03-31-11h30-Jes%5C%C3%5C%BAs-Labarta.pdf.

[360] SiFive. *SiFive Intelligence X280*. Accessed: 2021-09-15. URL: https://www.sifive.com/cores/intelligence-x280.

[361] M. Cavalcante. “Ara v2.0: RISC-V Vector Processor Implementation in GlobalFoundries 22FDX”. In: *RISC-V Summit Proceedings*. December 2019. URL: https://riscv.org/wp-content/uploads/2019/12/12.11-16.40a-Ara-v2.0-RISC-V-Vector-Processor.pdf.

[362] M. Platzer and P. Puschner. “Vicuna: A Timing-Predictable RISC-V Vector Coprocessor for Scalable Parallel Computation”. In: *33rd Euromicro Conference on Real-Time Systems (ECRTS 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2021.

[363] M. Johns and T. J. Kazmierski. “A Minimal RISC-V Vector Processor for Embedded Systems”. In: *2020 Forum for Specification and Design Languages (FDL)*. 2020, pp. 1–4. DOI: 10.1109/FDL50818.2020.9232940.

[364] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang. “[DL] A Survey of FPGA-Based Neural Network Inference Accelerators”. In: *ACM Trans. Reconfigurable Technol. Syst.* 12.1 (Mar. 2019). ISSN: 1936-7406. DOI: 10.1145/3289185.

[365] A. Cheikh, S. Sordillo, A. Mastrandrea, F. Menichelli, G. Scotti, and M. Olivieri. “Klessydra-T: Designing Vector Coprocessors for Multithreaded Edge-Computing Cores”. In: *IEEE Micro* 41.2 (2021), pp. 64–71. DOI: 10.1109/MM.2021.3050962.

[366] G. Furano, S. Di Mascio, A. Menicucci, and C. Monteleone. “A European Roadmap to Leverage RISC-V in Space Applications”. In: *2022 IEEE Aerospace Conference*. Mar. 2022.

[367] J. Andersson. “Development of a NOEL-V RISC-V SoC Targeting Space Applications”. In: *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. Los Alamitos, CA, USA: IEEE Computer Society, July 2020, pp. 66–67. DOI: 10.1109/DSN-W50199.2020.00020.

[368] T. Hendrix. “Single Event Upset Characterisation of NOEL-V soft processor”. Master’s thesis. Delft University of Technology, 2022.

# ACKNOWLEDGEMENTS

First of all, I would like to thank my promotors. The collaboration with Alessandra Menicucci goes back to my MSc thesis in 2016 and she has been the ideal supervisor for me because she left me the freedom to chose what to do with my time and resources in this four years. Furthermore, I think our backgrounds of a physicist (also) working on radiation effects in semiconductors and of an electronic engineer designing digital subsystems (also) considering radiation effects nicely complement each other. A similar consideration holds for Eberhard Gill, who taught me in his Space Systems Engineering course how to properly formulate and analyze high-level requirements from a mission need, complementing my previous formation as a technologist working only up to the last layer of which I had relevant technical knowledge. In my PhD I aimed to combine his systematic top-down approach with my technologist know-how in electronic systems and my previous bottom-up approach.

I would also like to thank my funders and former colleagues at Cobham Gaisler, who made this PhD possible and collaborated with me on the design of the vector processor. A big thank you to Jan Anderson, Sandi Habinc, Alen and Niels.

A big thank you also to the other funders of my PhD, ESA. Furthermore, the people at the On-Board Computers and Data Handling section helped me to collect new ideas at the beginning of my PhD and to direct my efforts towards industry-relevant application. I would like to thank in particular my Master's Thesis supervisor Gianluca Furano and my PhD Supervisor Claudio Monteleone.

Finally, I would like to thank my colleagues and friends at TU Delft and at ESA for their support and advice, and for exchanging (not only) technical ideas with me.

Corona has hit hard most foreign PhDs at TU Delft. Working from home, alone, in a small, overpriced apartment is mentally wearing. This was not my case, as I had the most beautiful person I ever met by my side, who gifted me contentment during the last two years spent together in Leiden.

Infine, vorrei ringraziare i miei genitori che mi hanno permesso di studiare fino ai miei 24 anni e scusarmi per essere presente solo un mesetto all'anno. Spero che in futuro le cose possano cambiare.

# CURRICULUM VITÆ

## Stefano DI MASCIO

11-10-1991 Born in Rome, Italy.

## EDUCATION

2013–2016 MSc. Electronic Engineering
University of Rome "Tor Vergata"
*Thesis:* Low-Energy Proton Test of Safety-Critical Microcontrollers for Space Applications
*Supervisors:* Marco Ottavi, Gianluca Furano

2010–2013 BSc. Electronic Engineering
University of Rome "Tor Vergata"
*Thesis:* Design, Manufacturing and Characterization in C Band (4-8 GHz) of Passive Microwave Circuits
*Supervisor:* Lucio Scucchia

2005–2010 Electronics and Telecommunications
Istituto Tecnico Industriale Statale "G. Vallauri", Rome, Italy

## EXPERIENCE

2019–2022 NPI (On-Board Computers and Data Handling)
European Space Agency, Noordwijk, The Netherlands

2016-2018 Hardware Engineer
Cobham Gaisler AB, Gothenburg, Sweden

2016 Intern (On-Board Computers and Data Handling)
European Space Agency, Noordwijk, The Netherlands

## AWARDS

2020 DFTS Best paper award

# LIST OF PUBLICATIONS

## JOURNAL PAPERS

6. **S. Di Mascio**, A. Menicucci, E. Gill, G. Furano, C. Monteleone *Extending the NOEL-V Platform with a RISC-V Vector Processor for Space Applications*, Under review.

5. **S. Di Mascio**, A. Menicucci, E. Gill, G. Furano, C. Monteleone *On-board decision making in space with deep neural networks and risc-v vector processors*, Journal of Aerospace Information Systems 18, 8 (2021).

4. **S. Di Mascio**, A. Menicucci, E. Gill, G. Furano, C. Monteleone *Open-source IP cores for space: A processor-level perspective on soft errors in the RISC-V era* , Computer Science Review 39, 2021.

3. **S. Di Mascio**, A. Menicucci, E. Gill, G. Furano, C. Monteleone *Leveraging the Openness and Modularity of RISC-V in Space*, Journal of Aerospace Information Systems 16, 11 (2019).

2. F. Di Capua, L. Campajola, P. Casolaro, M. Campajola, A. Aloisio, A. Lucaroni, G. Furano, A. Menicucci, **S. Di Mascio**, F. Malatesta, M. Ottavi *Full characterization of a compact 90Sr/90Y beta source for TID radiation testing*, Advances in Space Research 63, 10 (2019).

1. **Stefano Di Mascio**, Alessandra Menicucci, Gianluca Furano, Tomasz Szewczyk, Luigi Campajola, Francesco Di Capua, Andrea Lucaroni, Marco Ottavi. *Towards defining a simplified procedure for COTS system-on-chip TID testing* , Nuclear Engineering and Technology **50**, 8 (2018).

## PEER-REVIEWED CONFERENCES

6. D. Cappellone, **S. Di Mascio**, G. Furano, A. Menicucci, M. Ottavi *On-Board Satellite Telemetry Forecasting with RNN on RISC-V Based Multicore Processor*, IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2020.

5. **S. Di Mascio**, A. Menicucci, E. Gill, G. Furano, C. Monteleone, M. Ottavi *On the Criticality of Caches in Fault-Tolerant Processors for Space*, IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2019.

4. **Stefano Di Mascio**, Alessandra Menicucci, Gianluca Furano, Claudio Monteleone, Marco Ottavi *The Case for RISC-V in Space*, International Conference on Applications in Electronics Pervading Industry, Environment and Society, 2018.

3. Alessandra Menicucci, Fabio Malatesta, Francesco Di Capua, Luigi Campajola, Pierluigi Casolaro, Gianluca Furano, **Stefano Di Mascio**, Marco Ottavi *Simplified Procedures for COTS TID Testing: A Comparison Between 90Sr and 60Co* , IEEE Radiation Effects Data Workshop (REDW), 2018.

2. Gianluca Furano, **Stefano Di Mascio**, Tomasz Szewczyk, Alessandra Menicucci, Luigi Campajola, Francesco Di Capua, Andrea Fabbri, Marco Ottavi *A novel method for SEE validation of complex SoCs using Low-Energy Proton beams*, IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2016).

1. **Stefano Di Mascio**, Marco Ottavi, Gianluca Furano, Tomasz Szewczyk, Alessandra Menicucci, Luigi Campajola, Francesco Di Capua, *Qualitative techniques for System-on-Chip test with low-energy protons* , International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS), 2016.