

Exploiting non-conventional DVFS on GPUs: application to Deep Learning

Francisco Soares Mendes

Thesis to obtain the Master of Science Degree in
Electrical and Computer Engineering

Supervisor(s): Doutor Nuno Filipe Valentim Roma
Doutor Pedro Filipe Zeferino Tomás

Examination Committee

Chairperson: Doutora Teresa Maria Sá Ferreira Vazão Vasques
Supervisor: Doutor Nuno Filipe Valentim Roma
Member of the Committee: Doutor Luís Miguel Teixeira D'Avila Pinto da Silveira

October 2020

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

I would like to express my sincere gratitude to my mentors and thesis supervisors, Professor Nuno Roma and Professor Pedro Tomás, for all the discussions about computer hardware, for always setting the bar high and for all their helpful insight and guidance through the development of the work presented in this thesis. Furthermore, I would like to thank INESC-ID for providing me the tools to develop the experimental results for this dissertation.

First, I must thank my grandfathers Filipe and António, my uncle João and my father for always inspiring me to get my "hands dirty" and be practical and objective when trying to solve a problem.

I would also like to thank my dear friends André Meneses, André Pereira, Edgar, Gonçalo, Ivan, Pedro and Rodrigo for all the good times that we spent, and for each of them always having an interesting and different topic to talk about.

Finally, I must express my very profound gratitude to my family, who always believed in me, especially: to my sister, Rita, for all the insightful conversations and especially, for all the bits of advice about really everything that I need; to my parents, for all the love, support, care, attention and opportunities that you provide me; and to my girlfriend, Margarida, for tickling me on her unique way to aspire higher.

I've got a sense that all of you build a little bit of a stepping stone on which I can climb to look further ahead.

This work was partially supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) under projects UIDB/CEC/50021/2020 and PTDC/EEI-HAC/30485/2017.

Resumo

Atualmente, as unidades de processamento gráfico (do inglês GPUs) são um dos principais dispositivos computacionais usados para acelerar aplicações de cariz paralelo. Contudo, esse imenso desempenho acarreta um alto consumo energético. Diversas soluções podem ser adotadas para aumentar a eficiência energética desses dispositivos. Porém, o escalonamento de tensão-frequência (T-F) tem sido a solução que obtém o melhor resultado, permitindo melhorar as métricas de eficiência energética de forma automática e independente do tipo de aplicações a serem executadas.

As implementações atuais de escalonamento dinâmico de tensão e frequência (do inglês DVFS) em GPUs são unidimensionais, ajustando a frequência dentro dos pares padrão de tensão-frequência. No entanto, este ajuste é insuficiente, pois não garante o par T-F mais adequado à aplicação a ser executada. Esta dissertação apresenta uma nova metodologia para caracterizar o impacto do DVFS não convencional em GPUs, capaz de colmatar o carácter unidimensional das implementações actuais. Para atingir este objetivo, a abordagem proposta define um espaço de parametrização que determina a faixa de tensão tolerável para cada frequência. A mesma foi testada em duas GPUs da AMD com os resultados a mostrarem que estes dispositivos, em particular, são ambos capazes de operar em segurança com até menos 20% do valor padrão de tensão. Este espaço de parametrização definido foi então usado pelo mecanismo de otimização T-F desenvolvido para selecionar automaticamente a configuração de maior eficiência energética. Quando aplicado a aplicações de Aprendizagem Profunda e, especificamente, Redes Neurais Convolucionais, o mecanismo de otimização proposto demonstra ser capaz de melhorar a eficiência energética da GPU em até 44% sem qualquer deterioração relevante da precisão do modelo de rede neural a ser treinado.

Palavras-chave: Unidade de Processamento Gráfico, Escalonamento Dinâmico de Tensão e Frequência, Redução da Tensão, Mecanismo de Otimização, Aprendizagem Profunda, Redes Neurais Profundas.

Abstract

Nowadays, Graphics Processing Units (GPUs) are the primary computational devices used to accelerate highly parallel applications. However, this immense performance comes at the cost of high energy consumption. Several solutions can be adopted to increase the energy-efficiency of these devices. Though, Voltage-Frequency (V-F) scaling has been the one that achieves the best results, by allowing to automatically improve this metric and independently of the workload. However, current implementations of Dynamic Voltage and Frequency Scaling (DVFS) on GPUs are still one-dimensional, by simply adjusting frequency while relying on default voltage settings. To overcome this limitation, this dissertation introduces a new methodology to fully characterize the impact of non-conventional DVFS on GPUs. To attain this objective, the proposed approach defines a Usable Execution Space (UES) that determines the tolerable voltage range allowed by each frequency. The conducted experimental evaluation, using two out-of-the-shelf AMD GPUs, demonstrated that these particular devices are able to be safely undervolted by more than 20%. The devised UES is then used by a conceived V-F optimization mechanism, which was created to automatically select the most energy-efficiency configuration. When applied to Deep Learning applications and, specifically, Convolutional Neural Networks (CNNs), the proposed optimization mechanism can improve the GPU energy efficiency by up to 44% without any measured deterioration of the CNN model accuracy.

Keywords: Graphics Processing Unit, Dynamic Voltage and Frequency Scaling, Undervoltage, Optimization Mechanism, Deep Learning, Deep Neural Networks.

Contents

Acknowledgments	v
Resumo	vii
Abstract	ix
List of Tables	xv
List of Figures	xvii
List of Acronyms	xxi
1 Introduction	1
1.1 Objectives	2
1.2 Main Contributions	2
1.3 Dissertation Outline	3
2 Background	5
2.1 General Purpose Computing on GPUs	5
2.1.1 General Overview of a GPU Architecture	6
2.1.2 GPU programming model	7
2.2 CMOS Circuit Characterization	7
2.2.1 Propagation delay and circuit critical path	8
2.2.2 Voltage guardband, PVT Variation and Aging	10
2.2.3 Power Consumption	12
2.3 Dynamic Voltage and Frequency Scaling	13
2.3.1 Control Mechanism	14
2.3.2 GPU DVFS Characterization	17
2.3.3 DVFS Optimization	19
2.3.4 Decoupled V-F - Non-conventional DVFS optimization	21
2.4 Undervoltage and Imprecision tolerant applications	23
2.5 Summary	24
3 GPU architectural characterization to decoupled V-F	25
3.1 Characterization Benchmarks	26
3.2 Experimental Setup and Methodology	32
3.2.1 Voltage and Frequency control API	33

3.2.2	Testing procedure	34
3.3	Limiting components to the voltage exploration space	34
3.3.1	DRAM	35
3.3.2	Cache	35
3.3.3	Shared Memory	36
3.3.4	Arithmetic and Logic Unit	36
3.3.5	Non-linear Operations	37
3.3.6	Branches	39
3.3.7	Reduction	39
3.3.8	General Comments and Remarks	40
3.4	Effect of a decoupled V-F scaling on performance and energy consumption	42
3.4.1	DRAM	42
3.4.2	Cache and Shared Memory	43
3.4.3	Arithmetic and Logic Unit	44
3.4.4	Non-linear Operations	46
3.5	Temperature Model	47
3.6	Summary	48
4	Decoupled V-F Optimization Mechanism	49
4.1	Decoupled V-F Optimization Mechanism description	50
4.1.1	Architecture and Execution Overview	50
4.2	Optimization Mechanism Implementation	55
4.3	Library description	56
4.4	Summary	58
5	Application to Deep Learning	59
5.1	Deep Learning Overview	59
5.1.1	Deep Neural Networks	60
5.1.2	DNN Architectures	60
5.1.3	Training and Inference	62
5.1.4	High-Level Libraries and Software Frameworks	62
5.2	DNN Performance and Energy Efficiency Improvement	63
5.3	Non-conventional V-F on CNNs	64
5.3.1	Convolution Layer	64
5.3.2	Fully-Connected Layer	67
5.3.3	Error Analysis	68
5.4	CNN training and inference with non-conventional V-F	70
5.4.1	Feasibility Assessment	70
5.4.2	Optimization algorithm adaptation to CNN training	72
5.4.3	Experimental Results	76

5.5 Summary	78
6 Conclusions	81
6.1 Future Work	82
Bibliography	85
A How to control and set the desired V-F pair with rocm-smi	93
A.1 AMD Vega 10 Frontier Edition	94
A.2 AMD Radeon 5700 XT	95

List of Tables

2.1	GPU core and memory performance levels for the AMD Vega 10 Frontier Edition GPU. . .	14
3.1	Devised set of kernels to characterize GPU to Non-Conventional DVFS	26
3.2	Considered GPUs in the conducted experimental characterization.	32
5.1	Convolution Parameters.	64
5.2	Comparing CNN training test set accuracy with the application of different undervoltage levels.	71
5.3	Evaluation of performance, energy, and EDP when applying non-conventional DVFS in the training of neural networks.	74
5.4	Mode of selected V-F configurations at the end of each optimization stage.	78
A.1	Vega 10 GPU core performance levels general configuration.	94
A.2	Radeon 5700 XT GPU core V-F pairs general configuration.	96

List of Figures

2.1	AMD's Graphics Core Next logical organization.	6
2.2	a) CMOS inverter. b) Noise margin definitions: $NM_L = V_{IL} - V_{OL}$ and $NM_H = V_{OH} - V_{IH}$	8
2.3	Logic gate propagation delay: t_{pHL} - propagation delay high to low; t_{pLH} - propagation delay low to high.	8
2.4	First order RC circuit and its corresponding temporal response to step input.	9
2.5	Critical path between two registers (red dashed line).	9
2.6	Voltage guardband ensures reliability by making the transistors switching faster.	10
2.7	Voltage guardband ensures operation reliability by effectively inserting some extra timing margin.	11
2.8	GPU Core voltage frequency curve, red dots indicate the default user-defined voltage frequency pair - AMD Radeon 5700 XT.	14
2.9	Interval-based DVFS procedure.	15
2.10	AMD/NVIDIA DVFS control mechanism.	16
3.1	Overview of the methodology and its objective outputs.	26
3.2	Vega 10 - DRAM domain - Usable voltage for each frequency configuration.	35
3.3	Core domain - Cache L2 - Usable voltage for each frequency configuration with varying cache stress (<i>OPS</i> value).	36
3.4	Core domain - Shared Memory - Usable voltage range for each frequency configuration with varying shared memory stress (<i>OPS</i> parameter).	37
3.5	Core domain - ALU-MAC - Usable voltage margins for the different data types and operand's precision.	38
3.6	Core domain - Special Function Unit - Usable voltage for each frequency configuration with varying operand type: SP - Single Precision Floating-Point, DP - Double Precision Floating-Point.	38
3.7	Core domain - Branches - Usable voltage for each frequency configuration with varying number of branches per iteration.	39
3.8	Core domain - Reduction benchmark - Usable voltage for each frequency configuration with varying operand type: Int - 32-bit Integer, SP - Single Precision Floating-Point, DP - Double Precision Floating-Point.	39

3.9	Comparison of usable GPU core voltage ranges for all the considered architectural components of the GPU.	41
3.10	Vega 10 - DRAM domain - DRAM benchmark - Normalized energy consumption and execution time. Each connected data point represents a $50mV$ undervoltage in relation to the previous one.	42
3.11	Vega 10 - Core domain - DRAM - Normalized energy consumption and execution time. The dashed line connects the default V-F configurations and each connected data point represents a $50mV$ undervoltage in relation to the previous one.	43
3.12	Core domain - Cache L2 - Normalized energy and performance variations with OPS=0. The dashed line connects the default F-V configurations.	44
3.13	Core domain - Cache L2 - Obtained normalized Energy-Delay Product (EDP) with OPS=0.	44
3.14	Core domain - Shared Memory - Normalized energy consumption and execution time. The dashed line connects the default F-V configurations and each connected data point represents a $50mV$ undervoltage in relation to the previous one.	45
3.15	Core domain - Shared Memory - Obtained normalized Energy-Delay Product (EDP) with OPS=0.	45
3.16	Core domain - ALU-MAC - Normalized energy and performance chart for the benchmark setup with different values of d for single precision floating-point (see Listing 3.5). The dashed lines connect the results for default F-V configurations.	45
3.17	Core domain - ALU-MAC - Obtained normalized Energy-Delay Product (EDP) for $d=1, 3$	46
3.18	Core domain - Special Function Unit - Normalized energy and performance for single-precision floating-point data type. The dashed lines connect the results for default F-V configurations.	46
3.19	Core domain - Special Function Unit - Obtained Energy-Delay Product (EDP) for single-precision floating-point data type.	47
3.20	Undervoltage capabilities with changing temperature condition and defined temperature model.	47
4.1	V-F Optimization Mechanism Block Diagram.	51
4.2	User provided input example for the optimization mechanism (left) and correspondent usable execution space chart (right). Black-shaded regions represent unusable operation points (GPU crash), while green shaded regions represent tested DVFS operating points.	51
4.3	Flowchart of each iteration of the Optimization Algorithm Execution.	52
4.4	Layer diagram of the developed V-F Optimization Mechanism. The blocks colored in blue represent the developed parts of the optimization mechanism developed in the context of this dissertation.	56
5.1	Model of fully-connected (feed-forward) DNN	60
5.2	Core Domain - Convolution Layer - Usable voltage range.	65

5.3	Core domain - Convolution Layer normalized energy and performance chart to non-conventional V-F configurations.	66
5.4	Core Domain - Convolution Layer - Obtained normalized Energy-Delay Product heat-map for the three algorithms.	67
5.5	Core Domain - Fully-Connected Layer - Usable GPU core voltage range. $[a]$, $[b]$ and $[c]$ values represent matrix sizes (example $A_{a \times b} \cdot B_{b \times c}$).	68
5.6	Core Domain - Fully-Connected Layer normalized energy and performance chart to non-conventional V-F configurations.. . . .	68
5.7	Core Domain - Fully-Connected Layer - Obtained normalized Energy-Delay Product heat-map.	69
5.8	Convolution Layer - Average percentage of accurate results and relative error distribution of non-accurate outputs for the minimum usable core voltage across all considered core frequency values.	70
5.9	Fully Connected Layer- Average percentage of accurate results and relative error distribution of non-accurate outputs for the minimum usable core voltage across all considered core frequency values.	70
5.10	Core domain - CNN models - superposition of all V-F configuration test set loss and model accuracy.	72
5.11	Core domain - CNN models - Normalized energy consumption and execution time for training + inference. The dashed line connects the default V-F pairs, and the diagonal striped pattern indicates the plateau of minimum energy consumption.	73
5.12	Core domain - CNN models - Obtained normalized Energy-Delay Product (EDP) for training + inference.	73
5.13	Core domain - CNN models - Obtained normalized Energy-Delay Product (EDP) for inference phase.	74
5.14	Graphical representation of the power consumption overtime of the first ten training epochs of a CNN model (numbers 0 to 9 represent the execution of training epochs). The orange line represents the case where the training procedure is started at the Space Exploration stage, and the green line represents the case where the training procedure is only started on the Fine-Tuning stage.	75
5.15	Usable Execution Space for the Vega 10 GPU.	76
5.16	WideResNet <i>Space Exploration</i> example - normalized EDP values for 20 epochs. Initial value is 1600MHz and 1.2V.	77
5.17	Median normalized EDP of default DVFS vs training CNN with V-F optimization model. . .	78
A.1	Rocm-smi interface.	93
A.2	Change in voltage curve to select a given V-F pair.	95

List of Acronyms

- AFVS** Adaptive Frequency and Voltage Scaling.
- ALU** Arithmetic and Logic Unit.
- API** Application Protocol Interface.
- ApSA** Application-aware Scalable Architecture.
- BTI** Bias temperature instability.
- CMOS** Complementary metal-oxide-semiconductor.
- CNN** Convolutional Neural Network.
- CPU** Central Processing Unit.
- CU** Computing Unit.
- DL** Deep Learning.
- DNN** Deep Neural Network.
- DRAM** Dynamic Random-Access Memory.
- DVFS** Dynamic Voltage and Frequency Scaling.
- EDP** Energy Delay Product.
- GAN** Generative Adversarial Network.
- GPGPU** General Purpose Graphical Processing Unit.
- GPU** Graphical Processing Unit.
- HCI** Hot Carrier Injection.
- HSA** Heterogeneous System Architecture.
- MAC** Multiply and Accumulate.
- NCU** Next Compute Unit.
- PVT** Process, Voltage and Temperature.
- RISC** Reduced Instruction Set Computer.
- RMSE** Root Means Square Error.
- RNN** Recurrent Neural Network.
- ROC** Radeon Open Computing platform.
- SFU** Special Function Unit.
- SGD** Stochastic Gradient Descent.
- SIMT** Single Instruction Multiple Threads.
- SM** Streaming Multiprocessor.

UPT Unsupervised Pre-trained Network.

Chapter 1

Introduction

Graphical Processing Units (GPUs) were brought to life in the advent of computer graphics in the hopes of improving the capabilities of 3D games. However, the increased parallel computing capabilities exhibited by these devices made them highly appealing to running more general computational demanding applications, coining the term GPGPU - General-Purpose Graphical Processing Units.

Along this path, the development of new architectural generations of GPU devices has traditionally aimed at improving performance and throughput. However, having already reached the maximum supported power consumption and with device fabrication techniques being harder and harder to minimize, extra performance gains need to come while keeping the pace for a greater GPU energy-efficiency.

Researchers have been looking for ways to decrease power consumption to improve energy efficiency with some degree of success, with new architectural designs targeting, for example, operation with lower precision encoding (reduced number of bits) to reduce the number of combinatorial logic. Nevertheless, techniques such as Dynamic Voltage and Frequency Scaling (DVFS) are still the ones that are having a more direct impact on the industry, by being widely available on current out-of-the-shelf devices. These systems' working principle is to carefully select from a list of voltage-frequency (V-F) pairs, the one that should be used, in order to target the current GPU state. Since the dissipated power is directly proportional to the applied frequency and to the square of the applied voltage, the variation of these parameters directly and significantly impacts the devices' power and energy consumption.

In theory, DVFS systems are able to find the V-F configuration that achieves better energy efficiency. Nevertheless, practical implementations of such mechanisms seem to exhibit two flaws: i) they do not consider the running application characteristics and ii) use pre-defined V-F pairs that are conservatively selected by manufacturers. In particular, at each frequency level, the chosen voltage supply is set at a level that leaves a significant voltage margin in relation to the necessary one. Such margin is put in place to guarantee a safe operation under all conditions. However, recent studies prompt the idea that such conservative voltage margin is leaving a significant energy-efficiency gain on the table [1].

Supported on this observation, it has been recognized that the correct utilization of non-conventional V-F pairs may improve the energy-efficiency of already deployed GPUs.

One particular application domain that can significantly benefit from such energy-efficiency improvements is Deep Learning, and more specifically, Deep Neural Networks (DNNs). These algorithms are having a significant impact on industry and society by allowing for important breakthroughs in many application domains, such as computer vision, speech recognition, natural language processing, drug

discovery, genomics, etc [2]. However, DNNs are usually characterized by significant computational burdens, particularly when considering the training of very deep and complex networks, dealing with high dimensional data, such as images and videos. Another important characteristic of DNNs is their tolerance to a certain degree of computation errors [3], without any significant change in the training and inference results.

Therefore, the creation of a novel DVFS system that explores and selects non-conventional V-F pairs by considering the specific target application characteristics can allow for a substantial decrease in the energy consumed to train and deploy Deep Learning applications [4].

1.1 Objectives

To uncover the use of non-conventional V-F scaling, this thesis focuses on the following objectives:

- Access the viability of using non-conventional V-F pairs on regular GPUs.
- Characterize the behavior of GPU architectures to non-conventional V-F pairs.
- Develop a dynamic non-conventional V-F controlling and optimization mechanism that improves the *performance, energy consumption or energy-efficiency* of GPUs.
- Safely apply non-conventional V-F scaling on Deep Learning applications, characterizing the behavior of the training procedure.

1.2 Main Contributions

The work conducted in the scope of this dissertation contributes to open the discussion of further improving the current GPU DVFS systems, by exploring the conservative voltage guardband put in place by device manufacturers. The developed work demonstrates that non-conventional voltage-frequency (V-F) pairs are mostly allowed by out-of-the-shelf devices, leading to great energy-efficiency benefits and, in some cases, even performance enhancements.

To demonstrate and validate the use of those new V-F configurations, a new methodology was developed to test each architectural component of the GPU, analyzing and evaluating its specific voltage margin and characterizing their energy-performance behavior when subject to such configurations. The methodology was tested in two different architectural generation devices, both of them exhibiting a high tolerance to undervoltage.

Furthermore, to more easily benefit from the usable execution space that was defined by executing the set of benchmarks that compose the formulated methodology, a new V-F optimization mechanism was devised. The mechanism makes use of the native code repetition patterns, usually observed in GPGPU applications, to improve the applied V-F pair iteratively, taking into account not only the GPU power consumption, utilization and temperature (as it is done by current DVFS systems) but also the running application characteristics. With the use of such a conceived optimization mechanism, it is possible to perform dynamic voltage and frequency scaling while taking into account each application's intrinsic behavior and benefiting from the energy-efficiency improvements granted by the non-conventional V-F pairs.

To evaluate the proposed methodology and optimization mechanism, it was applied to the training procedure of deep learning applications, specifically the training of convolutional neural networks. The conducted evaluation explored the voltage margins and behavior of the fundamental algorithms that compose this type of operations, the distribution of the computational error that appears when using the lowest allowed voltage by the architecture and finally, the energy-efficiency improvement that is achieved by the V-F optimization mechanism when targeting the execution of this type of algorithms. In particular, the description of the performed work targeting the training of neural networks helps to understand the complete flow of actions that should be taken to take full advantage of exploring the conservative voltage guardband put in place by device manufacturers, and with that, improve the energy-efficiency of current out-of-the-shelf GPUs.

The main scientific contributions of this work have been published for communication in the following conference:

- F. Mendes, P. Tomás and N. Roma, "Exploiting non-conventional DVFS on GPUs: application to Deep Learning", IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'2020), 2020.

1.3 Dissertation Outline

This dissertation is organized in 5 chapters and one appendix, with the following outline:

- **Chapter 2 - Background:** This chapter presents a summary of the current state-of-the-art related to the subject in focus on this dissertation. First, it introduces an overview of general-purpose computing on GPUs, providing a summary of their architecture and programming model. It presents the key concepts that should be present when developing for this kind of devices and the standard programming interfaces between the GPU and the host CPU. A bottom-up approach is also drawn to present the fundamental concepts of frequency and voltage scaling on CMOS devices, and how they translate to the current DVFS systems presented in out-of-the-shelf GPUs. It finalizes by granting a sketch of how better exploring and going against the conventional voltage-frequency scaling allows for better energy efficiency of GPUs, being that the stepping stone of this dissertation.
- **Chapter 3 - GPU architectural characterization to decoupled V-F:** In this chapter, the developed methodology to characterize the use of non-conventional V-F scaling is presented. The chapter starts by exploring each developed stressing component benchmark's motivation and objectives, and it follows by testing the methodology in two different GPU architectures. The methodology is used to: determine the minimum voltage allowed by each architectural component and evaluate the performance, energy consumption and energy efficiency, when prompting non-conventional V-F pairs on them. It finishes by experimentally testing the effects of temperature on the undervoltage capabilities of the GPU architecture.
- **Chapter 4 - V-F Optimization Mechanism:** This chapter analyses and proposes a solution to dynamically adjust the frequency and voltage using the newly tested non-conventional V-F pairs. Programmers can execute their algorithms using this tool, while the most energy-efficient V-F

configuration is being discovered, targeting the current GPU state and performance metrics, and application output.

- **Chapter 5 - Application to Deep Learning:** In this chapter, the previous two chapters' results are combined and used to apply the developed methodology on a deep learning application. This chapter has three objectives: showcase the practical advantages of extending the default voltage-frequency scaling with non-conventional V-F pairs on a target application; evaluate the use of the developed optimization mechanism, and, overall, the chapter indicates how a user should proceed to safely and beneficially use the work developed in this dissertation.
- **Chapter 6 - Conclusions:** This final chapter presents the accomplished results from this work and the possible research directions to take in future work.
- **Appendix A:** This appendix presents all the necessary steps and commands to control the rocm-smi tool with the objective of setting the desired voltage-frequency pair on the target GPUs.

Chapter 2

Background

This chapter provides an overview of modern GPU architecture and respective programming models that allow the extensive use of these devices to execute more computationally intensive applications, followed by an analysis of techniques to improve the energy efficiency of the same.

The following sections present a bottom-up sequence of background and related work that supports this thesis — starting by the physical analysis of the digital circuits and the effects of V-F scaling and temperature on them. It continues by presenting the most common procedure to improve GPUs energy efficiency (DVFS) and finishes by exploring the techniques that will allow for further improvements to these device based on a complete decoupling between the applied voltage supply and operating frequency.

The relevance of the presented work emerges from the reduced number of studies on the effects of decoupled voltage scaling on GPUs, one of the objectives of this dissertation. The main reason is probably mainly due to lack of support for independently controlling these parameters on NVIDIA GPUs (until recently, the dominant player on the market [5, 6]) that is now allowed by the novel AMD software stack used on this work.

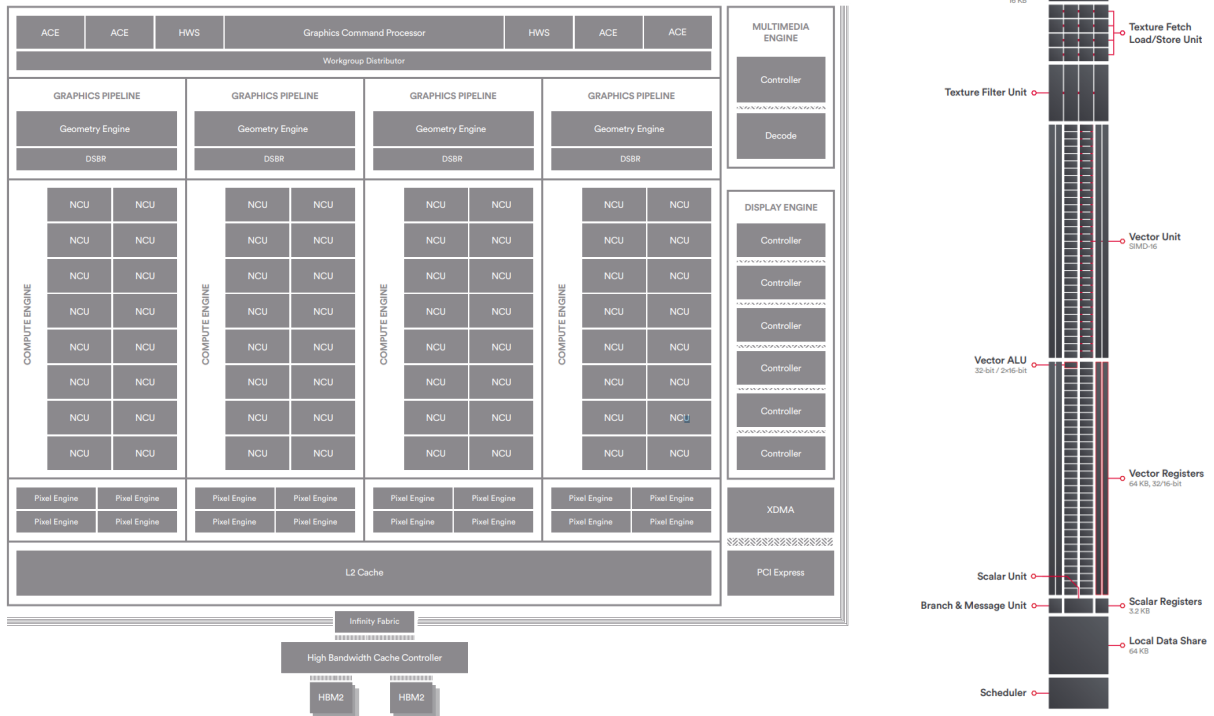
2.1 General Purpose Computing on GPUs

A GPU is a highly parallel programmable processor, that favours the execution of the same instruction on multiple data elements, belonging to the category of *Single Instruction Multiple Threads* - SIMT processors. When referring to GPUs, it is still common to be talking about their graphics capabilities. However, more and more programs are taking advantage of their highly parallel architecture to accelerate general purpose applications, leading to the connotation of this device as a GPGPU - General Purpose Graphical Processing Unit.

The development and deployment of GPGPU applications are only possible with the creation and adoption of standardized programming models and APIs that allow for hardware abstraction. This section provides a general overview of the GPU architecture, followed by the presentation of the most common and used software tools available for GPGPU programming.

2.1.1 General Overview of a GPU Architecture

The architecture of a modern GPU, depicted in figure 2.1, can be roughly divided into computation and memory components. The computation part is usually composed of the vertex shader, the rendering engine, and the RISC processors. The vertex shader and rendering engine are included on the graphics pipeline and are not generally used on GPGPU applications. The RISC processors are responsible for the GPU programmable calculations and depending on the manufacturer, they are denoted as streaming multiprocessors (SM) in NVIDIA GPUs [7] or computing units (CU) in AMD GPUs [8].



(a) Chip block diagram, example with 4 Compute Engines (RISC multi-processors), each with 16 NCU (Compute Units)

(b) NCU

Figure 2.1: AMD's Graphics Core Next logical organization.

As it was referred before, one of the significant benefits of GPUs is, the ability to concurrently execute multiple threads. To accomplish that, each SM or CU is made up of hundreds of execution units. However, to manipulate such number of threads, it is necessary to have a significantly large and fast memory system, not only to save the context of each thread, but also to provide low-overhead context switching between the different sets of threads being executed. In that sense, modern GPU architectures include a large register file on each SM/CU (when compared to the number of registers presented in a CPU core). For reference, in the AMD GNC architecture, each CU has 49,152 (32-bit) registers [9].

In terms of memory, both the AMD and the NVIDIA GPUs present a 3 level hierarchy system: a global memory, accessible by all SM/CU (generally referred to as video memory); a shared memory associated to each SM or CU, accessible by the threads running on that SM or CU; and a set of read-only caches for constants and textures, specific to each execution unit.

A GPU device from AMD will be used to conduct the experimental part of this dissertation. For that reason, the terminology used by AMD will be adopted herein. However, the presented work is independent of the hardware and terminology itself, and could equally be applied to NVIDIA GPUs.

2.1.2 GPU programming model

The development of CUDA [10] (by NVIDIA) and OpenCL [11] (by Khronos Group) were the driving force to using GPUs in general programming. CUDA and OpenCL are both parallel computing platforms and application programming interfaces (API) that allow developers to create GPU-accelerated applications, splitting the computations between the CPU and GPU. The first versions of these frameworks treated the GPU as an accelerating slave device, providing a set of directives that allow the CPU (master device) to transfer data, synchronize and control the GPU. Though, to take full advantage of the GPU architecture and create a true heterogeneous system, the CPU and GPU have to collaborate more efficiently. The creation of the Heterogeneous System Architecture (HSA) [12] framework acts on improving this problem by acting as a low-level intermediary API to provide improved coordination and communication for heterogeneous computing systems. More recently, AMD introduced the Radeon Open Computing platform (ROC) [13]. Like CUDA and OpenCL, ROC provides a set of tools that allow developers to create heterogeneous applications. Being a newer software stack, already built on the notions and added benefits of HSA runtime API, ROC allows the use of a wider set of programming frameworks like OpenCL¹, HCC², and HIP³.

This programming model is rather similar across the different platforms, with developers traditionally programming the GPUs using general-purpose languages like C, C++ and Fortran. More recently, the manufacturers are also starting to provide direct access to the GPU through higher-level languages like Python⁴, allowing for an easier development and adoption of GPUs as accelerating devices.

Overall, when executing a program on GPUs, using either of the previously mentioned platforms, a kernel is invoked on the GPU, that will execute across several parallel threads. Following this work split, the frameworks expose three levels of abstraction: *threads*, *thread blocks* and *block grid*. The number of *threads* to be executed can be explicitly defined by the programmer, or implicitly set by the compiler. The *threads* are grouped in *thread blocks* containing an amount of *threads*, and the *thread blocks* are in turn arranged in a *block grid*. When the Kernel is executed, the scheduler maps each *thread block* onto a CU. Due to this mapping, *threads* within a *thread block* are able to communicate over the CU shared memory and their execution is synchronized through programmable directives.

During the execution of each *thread*, an identifier presents the location of the running *thread* within the *thread blocks* - `ThreadIdx` and within the *block grid* - `BlockIdx`. The *thread block* size can be obtained with the `BlockSize` directive.

2.2 CMOS Circuit Characterization

For the last 40 years, CMOS (Complementary metal-oxide-semiconductor) has been the most used technology in the creation of digital circuits and processors in general [14].

There are two defined logic levels in digital circuits: logic level 0 and 1, each represented by an analog voltage range. To ensure its operations, the circuit requires a DC voltage value V_{DD} . The logic gates are excited through an input voltage V_i and an output voltage V_o , corresponding to the logic level resulting from their logic function (see Figure 2.2(a)). As it is illustrated in Figure 2.2(b), the

¹github.com/RadeonOpenCompute/ROCm-OpenCL-Runtime

²github.com/RadeonOpenCompute/hcc

³github.com/ROCm-Developer-Tools/HIP

⁴developer.nvidia.com/how-to-cuda-python

voltage range at the input of the logic gates is correctly interpreted if it falls inside a given range. Logic level 0 corresponds to the interval between GND ($0V$) to V_{IL} , while logic level 1 corresponds to the interval between V_{IH} to V_{DD} . In turn, the output is considered 0 if it goes from GND ($0V$) to V_{OL} and logic level 1 from V_{OH} to V_{DD} . The limits of the input and output logic levels depend on the intrinsic characteristics of the transistors, such as their dimensions and transconductance values. In addition to the transistors characteristics and operating voltage, any digital circuit is also characterized by their frequency of operation.

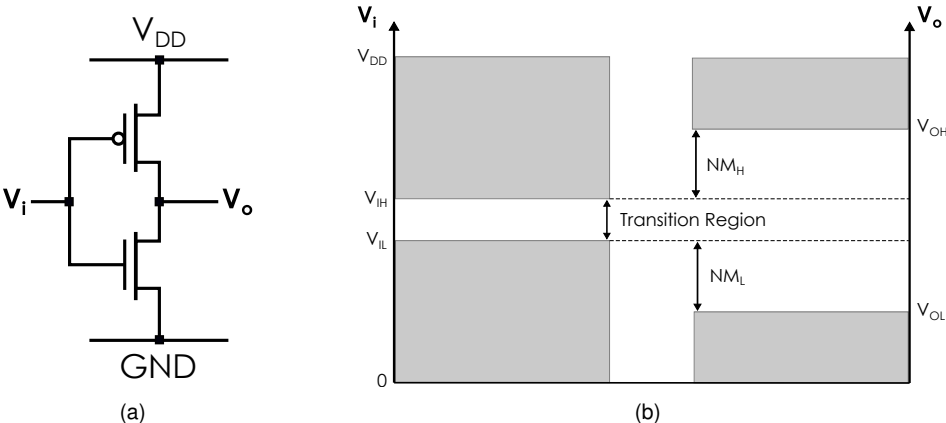


Figure 2.2: a) CMOS inverter. b) Noise margin definitions: $NM_L = V_{IL} - V_{OL}$ and $NM_H = V_{OH} - V_{IH}$.

The remaining of this section provides an introduction to the effects of voltage and frequency scaling on the transistor and circuit level operation.

2.2.1 Propagation delay and circuit critical path

The propagation delay of a logic gate (e.g., inverter) corresponds to the time interval (calculated at 50% of low/high transition) between the application of an input signal and corresponding output switching, as illustrated in Figure 2.3. Considering both transitions low to high and high to low, the logic gate propagation time (t_p) corresponds to the average value of the two propagation delays, as defined in Equation 2.1.

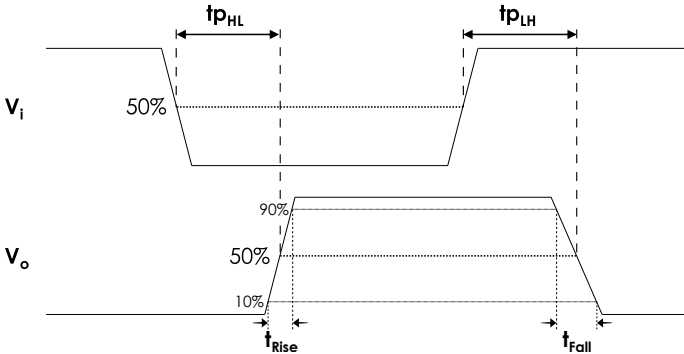


Figure 2.3: Logic gate propagation delay: t_{pHL} - propagation delay high to low; t_{pLH} - propagation delay low to high.

$$t_p = t_{p_{avg}} = \frac{t_{pHL} + t_{pLH}}{2} \tag{2.1}$$

This metric is related to the time that the logic gate takes to switch its output logic level (t_{Fall} for the high to low transition and t_{Rise} for the low to high transition). This transition can be modeled as a first-order RC circuit (see Figure 2.4) with the transient response following Equation 2.2, where τ corresponds to the time constant. This time constant reflects the intrinsic characteristics of the transistors that make up the logic gate, being $\tau = RC$, where R represents the average output resistance of the transistor when it is turned 'ON' and C the output capacitance that the logic gate is driving.

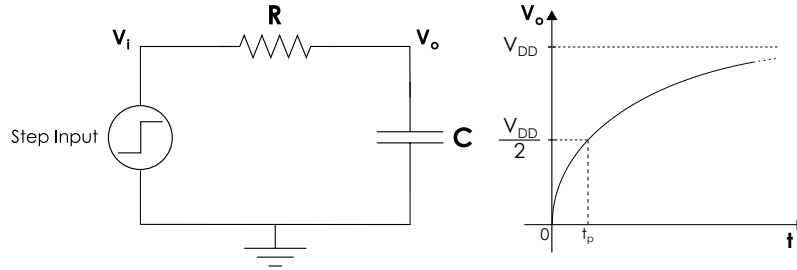


Figure 2.4: First order RC circuit and its corresponding temporal response to step input.

$$V_o = V_{DD} \cdot (1 - e^{-t/\tau}) \quad (2.2)$$

By solving Equation 2.2 for $V_o = \frac{V_{DD}}{2}$, it is observed that the propagation delay is a function of the supplied voltage and τ (with $\tau = RC$), as depicted in Equation 2.3.

$$t_p = -\ln\left(1 - \frac{V_{DD}}{2 \cdot V_{DD}}\right) \cdot \tau = -\ln(0.5) \cdot \tau = \ln(2) \cdot \tau \quad (2.3)$$

From all the logic paths (sequence of logic gates) that connect any two registers, the one which presents the largest sum of the propagation delay limits the overall maximum frequency that the CMOS circuit can operate, establishing itself as the critical path of the circuit (see Figure 2.5). However, in the particular case of a microprocessor circuit, depending on the operation (instruction) being performed (and so, of the used architectural component), the circuit's critical path can change.

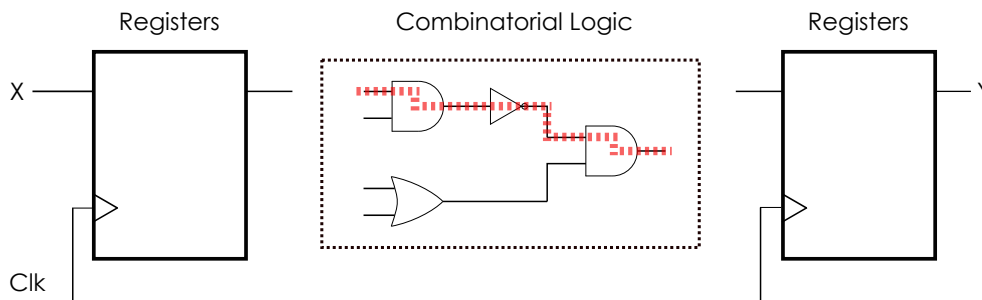


Figure 2.5: Critical path between two registers (red dashed line).

In this way, when scaling the circuit's operating frequency, it is only possible to increase it until it matches the inverse of the critical path propagation delay. When raising the frequency ahead of this value, the critical path is violated, meaning that when the next clock cycle starts, the output of the combinatorial logic may not be the correct one.

2.2.2 Voltage guardband, PVT Variation and Aging

After the release of the digital circuit, silicon vendors often deal with variations of their devices' specifications. On the design stage of the circuit, these variations need to be correctly predicted and accounted for, guaranteeing the correct operation of the circuit throughout time and range of conditions that the circuit will be exposed to.

Process, voltage and temperature (PVT) variation and aging impact the circuit in different manners and the solution for all is to put in place a voltage guardband, as defined in Figure 2.6(a). This voltage guardband increases the circuit nominal voltage from the best-case operating voltage selected from the transistors and gates' intrinsic characteristics. In Equation 2.3, the propagation delay is now obtained by considering the ratio between the best-case operating voltage⁵. Hence, when the silicon vendor opts to put in place a voltage guardband, increasing the supplied voltage (overvoltage), the propagation delay will follow Equation 2.4, where $V_{Threshold}$ depends on the transistors (and so, it does not depend on the supply voltage value) and $V_{Supply Voltage}$ is the supplier controlled parameter. Figure 2.6(b) illustrates the logic gate step response for over and undervoltage. Increasing the supply voltage will make the transistors switch faster, while decreasing it, reduces the switching pace.

$$t_p = -\ln\left(1 - \frac{V_{Threshold}}{V_{Supply Voltage}}\right) \cdot \tau = \ln\left(\frac{V_{Supply Voltage}}{V_{Supply Voltage} - V_{Threshold}}\right) \cdot \tau \quad (2.4)$$

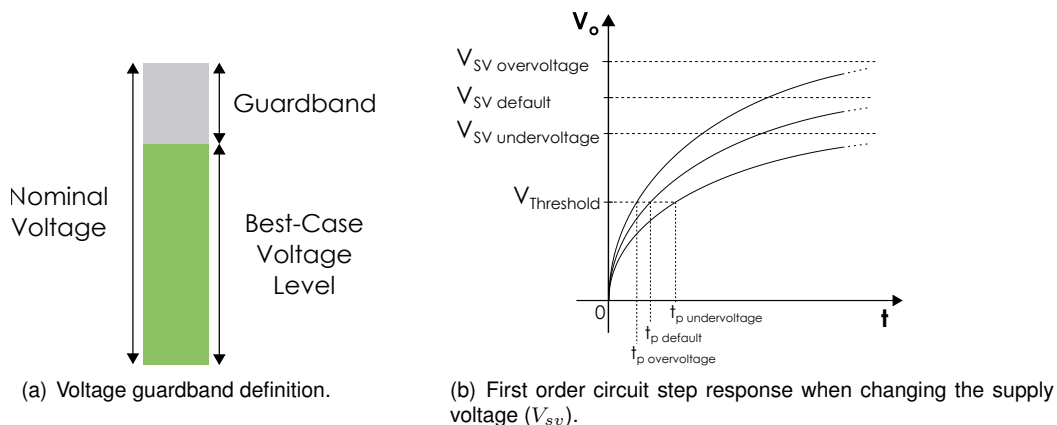


Figure 2.6: Voltage guardband ensures reliability by making the transistors switching faster.

Process variation and aging

Process variation and aging impact the circuit in different ways. Process variation is related to the modern fabrication process of silicon. It results from imperfections in the lithography and dopant diffusion, affecting the transistors' dimensions (for example, length and oxide thickness). This dimensionality change between the transistors can occur either intra-die, when devices from the same die present different features depending on their locations, inter-die, meaning that devices from one die can present different traits from devices from another batch of dies. The process variation affects the speed of transistors and the overall circuit characteristics, by varying the device voltage threshold and speed [15, 16]. Such variations are inherent to the fabrication process. Even though silicon manufacturers try to reduce

⁵(Threshold voltage) and the considered supply voltage

this impact, such variation will always occur as a *static* variation after the release of the chip to the market.

Aging of the CMOS circuits is a result of ongoing chip temperature variation and supply voltage change. The continuous variation of these two factors induces *Bias Temperature Instability* (BTI) and *Hot Carrier Injection* (HCI). BTI causes threshold voltage shifts over long periods due to the presence of voltage stress at the transistors' gate. On the other side, HCI is caused by the acceleration of carriers (electrons/holes) under lateral electric fields in the channel of MOS devices. The acceleration can get up to the point where the carriers gain enough energy and momentum to cause damage, degrading mobilities and again, changing the threshold voltages [17].

The occurrence of these phenomenons raises the possibility of the circuit not meet its original specifications. Hence, a circuit that is dimensioned without any headroom (not sufficiently large voltage guardband) may not be able to cope with such a process variation - producing a not working circuit; or stopping to work overtime due to aging.

Voltage variation

As it was previously observed, running the circuit at an increased voltage compared to the required voltage at the target frequency for typical workloads results in a faster circuit. This increment in performance allows for inserting an extra timing margin in each clock cycle, as described in Figure 2.7.

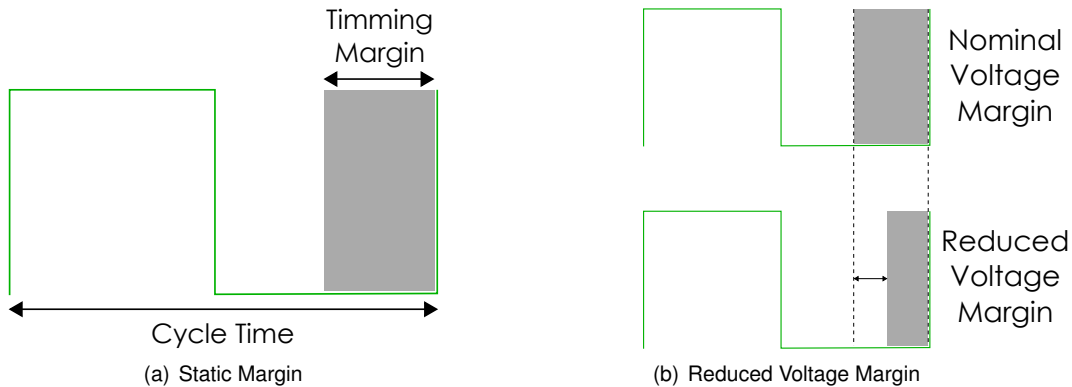


Figure 2.7: Voltage guardband ensures operation reliability by effectively inserting some extra timing margin.

This timing margin is of extreme importance to cope with voltage noise, the leading cause of voltage variation during the circuit execution. Voltage noise is mainly induced by di/dt droop. This phenomenon makes the actual measured voltage that is applied to the circuit components (as formulated in Equation 2.5), depend on the rate of change of the current being drawn.

$$V_{actual} = V_{DD} - L * \frac{di}{dt} \quad (2.5)$$

The runtime workload intensity variation induces the di/dt droop due to the rapid and significant change in current demand from the various circuit functional blocks [16]. Thus, in the case of processors, the workload type can directly impact voltage noise. As a result, a program that induces a bigger di/dt droop will need to have a bigger voltage guardband.

Temperature variation

Temperature variation occurs both due to changes on the environmental temperature, and due to changes in the temperature that is dissipated as heat by the transistors on the circuit. The temperature affects the transistors by changing the carriers' mobility ($\mu[\frac{cm^2}{V.s}]$), and threshold voltage (V_t)[18].

The carriers' mobility μ describes the drift velocity of a particle in an applied electric field, thus the transistor's capability to drive electric current. Usually, the carrier mobility of MOS transistors presents a very complex temperature dependence. However, in general, the mobility is said to decrease with temperature increase.

In the case of threshold voltage, its rate of change also follows the same principle of the carriers mobility. However, it usually has a more straightforward dependency of decreasing linearly with temperature increase.

The work of Freijado *et al.* [19] presents a model that tries to encompass all these variations and test the impact that temperature has on the propagation delay. The designed model predicts that the propagation delay increases linearly with the temperature increase, which would imply that the size of the voltage guardband reduces when the temperature increases.

2.2.3 Power Consumption

On a CMOS circuit, the total consumed power is decomposed into the dynamic and static parts

$$P_{CMOS} = P_{dynamic} + P_{static}. \quad (2.6)$$

The dynamic power relates to the power that is consumed by the transistors flipping stages (inverting the logic values), and corresponds to the power of charging and discharging the internal net capacitances. This value is proportional to the frequency that this change occurs. Equation 2.7 represents the general formulation of the dynamic power, where a represents the device utilization factor, C the total capacitance of the circuit, V the circuit supply voltage, and f the frequency of operation [20].

$$P_{dynamic} = aCV^2f \quad (2.7)$$

On the other hand, the static part of the power consumption comprehends three components: $P_{leakage}$, $P_{short-circuit}$ and P_{DC} [21]. The leakage power is independent of the transistors flip, and it represents the flow of electrons between the transistors' source, drain, and gate, known as leakage current. The short-circuit power comes from the instantaneous short-circuit connection between the supply voltage and the ground when the transistor flips. Finally, the Direct Current (DC) power corresponds to the power needed for powering the circuit. Equation 2.8 represents the expression with all static power consumption components.

$$P_{static} = P_{leakage} + P_{short-circuit} + P_{DC} \quad (2.8)$$

Usually, the dynamic power dominates the total power consumption of a circuit. However, with the current tendency to reduce the manufacturing size of transistors, the static power is becoming a more significant part [22, 23]. Nevertheless, as a common reference, and due to the usually dominant

weight of the dynamic power on the total power consumption, the power used by a CMOS circuit usually changes linearly with the clock frequency and quadratically with the supplied voltage.

2.3 Dynamic Voltage and Frequency Scaling

The widespread use of GPUs in both supercomputers and personal computing machines comes at the cost of a significant increase in power consumption. While a typical modern CPU consumes about 50 to 100W, it is common to see GPUs consuming between 200 and 300W of power. With these figures, the use of energy efficiency techniques to try to reduce power consumption becomes a vital issue.

As stated in Section 2.2.3, the power consumption of a CMOS circuit increases linearly with the operating frequency and quadratically with the supplied voltage. Thus, a direct manner of reducing this figure is by directly acting on these two parameters. A common approach of achieving this is the application of Dynamic Voltage and Frequency Scaling (DVFS), consisting on a power management technique which performs "on the fly" control of frequency and voltage. In particular, DVFS allows for an energy efficiency improvement by matching voltage and frequency settings to the GPU utilization. When the GPU is idle, the frequency is lowered, and when it is active, the frequency is increased. As presented in the previous section, the frequency scaling also implies a consequent change in voltage, to accommodate the fulfillment of the critical path timing constraints.

In general, the applied voltage level V is a function of the current operating frequency f , in the form of $V(f)$. However, by properly controlling the clock frequency, the required voltage level for stable operation of the circuit can also be maintained or even reduced, leading to further power savings.

In general, modern GPU boards offer an independent control over two pairs of frequency and voltage. Each pair (or domain) acts on a distinct part of the GPU, intending to maximize the performance or reduce the power consumption. The first domain concerns the GPU core, acting on all SM/CUs, the cache, and the interconnection fabric. The second affects the DRAM chips that compose the video memory.

The clock frequency is an independently controlled (within the physical possibilities of the CMOS manufacturing process) variable, and its change directly reflects on the performance that is achieved by the GPU. An increase in the clock frequency of the core results in an improvement of the SM/CU execution speed, while the same change in the memory frequency will increase the DRAM I/O throughput [21]. The voltage level of each domain is dependent on the clock frequency being used and it is usually defined based on tests performed by the manufacturer to ensure the correct operation of the circuit, independently of the workload.

The two major GPU silicon vendors, AMD and NVIDIA, adopted the concept of performance levels on their products. Each performance level is a pair of frequency and voltage that can be applied to each GPU DVFS domains. These vary from low power and performance levels to high power and performance ones. The idea of having multiple performance levels is to allow the application to have the best point of operation. In the case of the first GPU device (AMD Vega 10 Frontier Edition) that is going to be used in the experimental phase of this dissertation, the GPU core has eight performance levels, while the memory has only four. Table 2.1 shows the reference values (for the pairs of frequency and voltage) for each of the core and memory performance levels. The frequency and voltage of higher performance levels reflect the expected behaviour, with an increase of the supply voltage to accommodate

the frequency boost.

Core			Memory		
Level	Frequency [MHz]	Voltage [mV]	Level	Frequency [MHz]	Voltage [mV]
0	852	800	0	167	800
1	991	900	1	500	900
2	1138	950	2	800	950
3	1269	1000	3	945	1000
4	1348	1050			
5	1440	1100			
6	1528	1150			
7	1600	1200			

Table 2.1: GPU core and memory performance levels for the AMD Vega 10 Frontier Edition GPU.

Newer GPU DVFS systems, like the one used on the second GPU under test (AMD Radeon 5700 XT) improve the number of performance levels by allowing for a continuous use of all frequency values within the valid range (versus discretizing the domain into a set number of performance levels). In this case, there are three user-defined frequency-voltage pairs on which a quadratic regression is computed (see Figure 2.8), creating a $f(V)$ function that for every frequency, gives the correspondent voltage value.

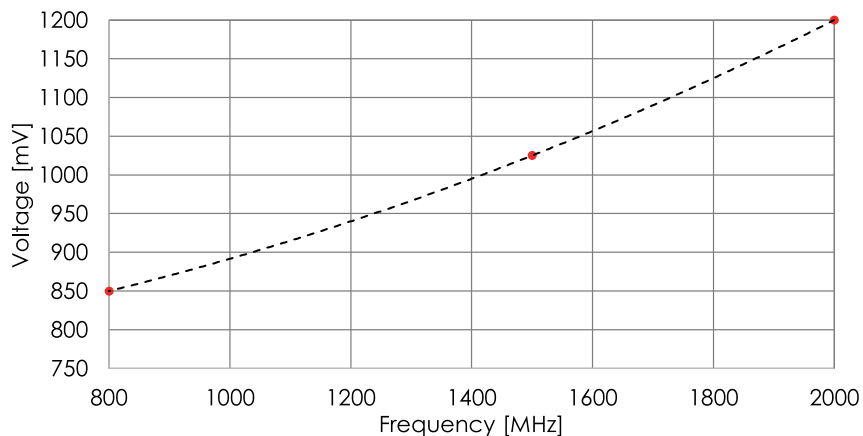


Figure 2.8: GPU Core voltage frequency curve, red dots indicate the default user-defined voltage frequency pair - AMD Radeon 5700 XT.

2.3.1 Control Mechanism

The correct choice of the most appropriate performance level for each GPU DVFS domain when executing a given application is one of the topics that has deserved a special attention from both manufacturers and researchers, since the design of the DVFS controller has a significant impact on the GPU's performance and energy efficiency.

The first implementations of GPU DVFS controllers took a direct inspiration from the CPU DVFS controllers and can be largely classified into interval-based, inter-task, and intra-task DVFS schemes [24].

Interval-based

Interval-based algorithms rely on the periodical measurement of the device's utilization, setting the next frequency and voltage levels based on the average measurement of utilization. The utilization U_i reflects the percentage of working time, w_i , that was spent by the GPU over the last time frame TF_i and can be formulated using equation 2.9.

$$U_i = \frac{w_i}{TF_i} \quad (2.9)$$

By applying either a arithmetic, a geometric, a weighted average, or even a more complex metric over the last n measurements, the next utilization U_{i+1} is predicted. If the predicted value surpasses pre-determined upper or lower thresholds, the frequency is adjusted up or down accordingly [25]. A *governor* is a set of parameters (such as frequency and voltage tables), thresholds and a utilization prediction algorithm that controls how the interval-based DVFS works. By choosing a different *governor*, the DVFS system can react differently to the same workload. Figure 2.9 schematizes the periodic procedure executed by the DVFS system [25].

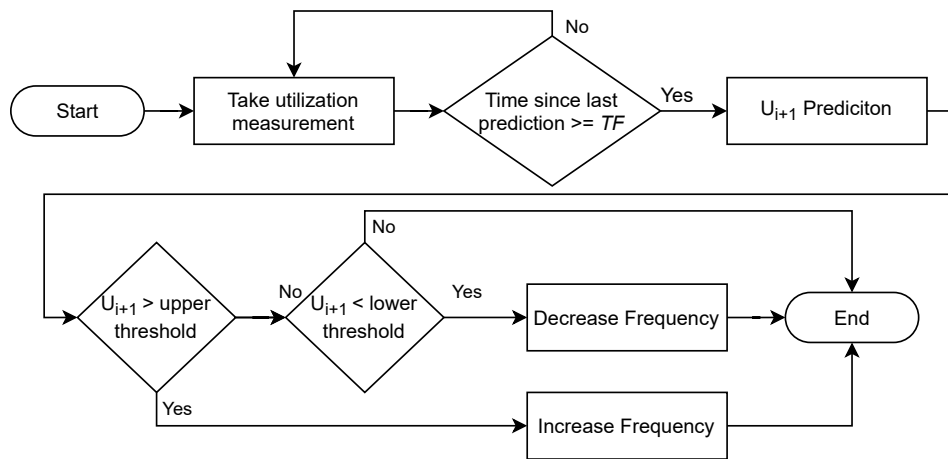


Figure 2.9: Interval-based DVFS procedure.

Inter/Intra task-based

The Inter and Intra task-based DVFS algorithms analyze the program source code, as well as some past runs profiling results to determine the optimal frequency/voltage for each task. This type of algorithm is composed of an intra-task and an inter-task analysis [26]. The intra-task part decomposes the process execution into the on-chip computation and off-chip access latencies. Based on the results, the optimal GPU core and memory frequencies are determined accordingly to the ratio between the two types of execution. Inter-task mechanisms utilize the intra-task results, assigning a signature to each type of task/process. By analyzing, in run-time, the GPU components' utilization, and using a lookup table that stores the signatures, the optimal frequency/voltage is selected for the sequence of tasks to be executed.

In general, the challenges of creating a better GPU DVFS mechanism relates to three factors: GPU power management, DVFS performance and power estimation tools and continuous changing architec-

ture. The current GPU devices deliver minimal and simplified power management mechanism, solely relying on controlling the device power cap. The continuous architectural changes introduced on each GPU generation does not allow for the creation and maturation of accurate quantitative GPU DVFS performance and power estimation tools. And again, the complete redo of the architecture on each generation also makes the energy efficiency strategies applied to one architecture design have different outcomes on the next one [21]. The observed results show that strategies like scaling up the processor frequency, race-to-idle [27] or "racing" [28], when a task is launched in the pursuit of finishing it as fast as possible and return to an idle state, prof to increase the energy efficiency of CPUs. However, that assumption not always results in the same outcome for GPUs [28].

AMD/NVIDIA DVFS Mechanism Example

The GPU DVFS control mechanism that is used nowadays by both AMD and NVIDIA, schematized in figure 2.10, is primarily an interval-based one. In particular, the most recent AMD GPU DVFS mechanism is called Adaptive Frequency and Voltage Scaling (AFVS) [29]. It takes into account the voltage levels across the different parts of the GPU, the die temperature, the desired frequency and the total power consumption. In order to maintain the total power consumption within the required power and temperature envelope. Upon launching a new task and within a set power target, the GPU tries to achieve the highest possible frequency (or highest performance level). For that frequency configuration, it also adjusts the voltage level to the one required to ensure the correct operation. Naturally, with the highest performance level, power and temperature will increase. When one of these parameters achieves the limit, the GPU decreases the frequency (or selecting a more energy-saving performance level) to maintain itself within the power and temperature target.

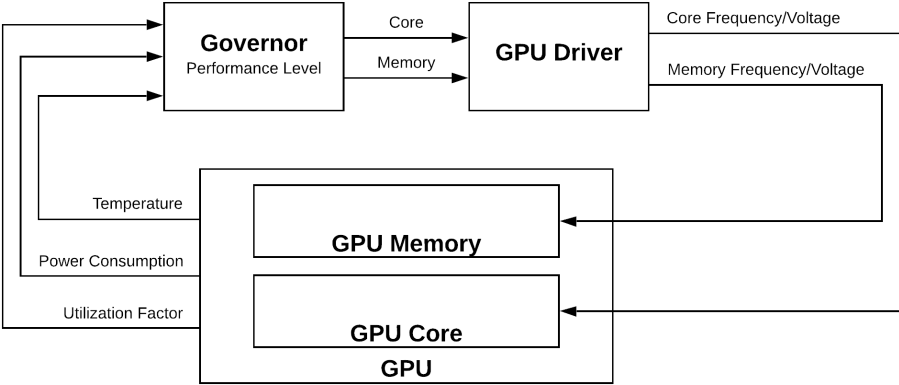


Figure 2.10: AMD/NVIDIA DVFS control mechanism.

The major drawback of current GPU DVFS implementations is not taking into account the type of tasks that the GPU is executing. The dominant control mechanism still considers the GPU as a black box and controls its DVFS settings by only looking at outside parameters. Even though such a black-box approach enables significant improvements in current hardware, it lacks the optimization of the frequency/voltage to the type of workload. For instance, a given application can be either compute-bound or memory-bound, depending on if the time that it takes to perform the task is limited by the processor performance or by the memory bandwidth and latency. This binary classification of applications also depends on the frequency of the core and memory. The same application can be compute-bounded at

a low core frequency but be memory-bounded at a higher one [30] (the bottleneck switches from the processing elements to the memory). In the case of a compute-bounded application, the limitation can be imposed by different components of the processor architecture, depend on the type of data (integer or floats) and on the intended precision (size of the operands). If one compares the computation with the same type of operands, but with different precision (for instance, 16 vs. 32 bits), even though the GPU uses the same arithmetic unit, the critical path for 32 bits operands is of increased length. Hence, the maximum delay between the input and the operations output increases with the operands' precision. Since no information about the application is provided to the DVFS controller, no adaptation of the V-F settings can be employed. Considering that the manufacturers do not usually tune their devices to the minimum voltage level (to accommodate for manufacturing imprecisions and to leave a safe guardband), there is some space to reduce the operating voltage of the circuits, leading to significant power savings.

Overall, the minimum operating voltage of each performance level is set at a high enough level to ensure the correct operation of the GPU, independently of the type of computations/workload. Nevertheless, it is possible to further fine-tune the voltage and frequency configuration if the type of task being executed is known at run-time. Furthermore, a significant limitation of the voltage-frequency control system is the time it takes to change and stabilize the intended V-F pair. On the case on the AMD Vega 10 Frontier Edition, the mechanism takes approximately 200-500 ms to change between performance levels [29].

2.3.2 GPU DVFS Characterization

To improve the performance and energy efficiency of a GPU, it is important to characterize and analyze the effects of applying DVFS techniques on its operation, with a special emphasis in what concerns the impact of the different parameters on different workload scenarios.

A complete GPU DVFS characterization is explored in the literature using two methodologies. The first refers to experimental studies, where researchers use real GPUs to perform voltage and frequency scaling. However, due to past dominance of NVIDIA over AMD [5, 6] and the fact that this manufacturer only offers limited support for independent voltage scaling tools, the majority of the work act solely on frequency scaling. The second approach uses simulators, like GPGPU-Sim⁶ and GPUWatch⁷ [31], to simulate various scaling approaches, like GPU core number scaling and per-core DVFS [21]. The benefit of using simulators over real hardware comes on the increased flexibility, enabling the experimentation of scenarios not supported by the frequency/voltage scaling tools provided by the manufacturers. In both methodologies, the studies act on the impact on performance, energy consumption and overall energy efficiency.

The following subsections present a brief overview of some of these works.

Experimental Approach

Jiao *et al.* [32] scaled the GPU core and memory frequencies of an NVIDIA Tesla GTX 280 GPU using three types of workload: a compute-bounded dense matrix multiplication application; a memory bounded application that performs a dense matrix transpose; and a mixed workload, executing a Fast

⁶<http://www.gpgpu-sim.org/>

⁷<http://www.gpgpu-sim.org/gpuwatch/>

Fourier Transform (FFT) computation. The experimental study showed that for the same core-memory frequency settings, the three applications showed different performance and energy efficiency curves. While a compute bounded application showed to be insensitive to memory frequency scaling, the memory bounded application takes advantage of high memory frequency and low core frequency. At last, the mixed workload FFT application profits from both high core and memory frequency. In general, it was also shown that energy efficiency could be determined by the instructions per cycle (IPC) metric and by the global ratio of memory transactions by computation transactions.

Ge *et al.* [33] explored dense matrix multiplication kernel executions in more detail using an NVIDIA Kepler K20c GPU. The conducted work revealed that for this type of kernel (compute-bounded), the GPU's power and achievable performance is linear to the GPU core frequency and that the total energy consumption had no relation to frequency scaling. In all the used application tests, the energy efficiency has a linear relation with the GPU frequency, with the highest energy efficiency being achieved when the highest clock frequency was employed.

Abe *et al.* [34] introduced a global scaling procedure that combines the GPU core and memory frequency with the host CPU frequency, in order to minimize the energy consumption. The first experiment tried to optimize the computation of dense matrix multiplication with different matrix sizes. By using this global scheme on a small matrix leads to a 28% energy saving, when using low GPU memory frequency and high GPU core frequency. The same procedure was then enlarged to a more diverse set of 33 benchmarks, where all the possible combinations of a low, medium, and high GPU core and memory frequencies were tested to find the optimal working settings. It was found that energy consumption can be reduced as much as 75% with a performance loss of 30% when the best settings were used.

Mei *et al.* [35] conducted a more general experimentation using 37 GPU benchmarks. In this work, it was possible to observe that the effect of GPU DVFS depends on the application characteristics. In all situations, the fine-tuning of DVFS per application (finding the lowest voltage level for the desired running frequency) conveyed an energy-saving of 20%, on average, with only a 4% performance loss. More recently, Mei *et al.*, expanded the previous work to analyze the relation between energy consumption and dynamic frequency scaling settings [21]. For the Rodinia benchmark [36], it was found that some benchmarks increase the energy consumption linearly with frequency scaling while others are insensitive to the change of this parameter. In the particular case of GPU memory frequency scaling, the work revealed that underclocking this component can result in a 30% energy decrease if the running application is not memory bounded. For the case of applications whose performance depends on the memory, decreasing its frequency can lead up to 54% energy increase, mostly due to the increased execution time. The tested set of applications showed that overclocking the memory results in diminishing the execution time with reduced overall energy consumption (the memory energy consumption increase overshadows the small reduction in computation time).

Overall, the relation between the DVFS settings and the resulting energy consumption depends heavily on the application type. One can conclude that a simple linear model (normally used by the GPU manufacturers) for DVFS settings is often inadequate to achieve the best performance or energy consumption/efficiency.

Simulation Approach

As it was previously referred, the characterization of the application of DVFS techniques on GPUs with simulation approaches is usually done by using software tools like GPGPU-Sim⁸ and GPUWattch⁹ [31]. GPGPU-Sim is a simulator of a GPU architecture running CUDA and OpenCL workloads. GPUWattch is an energy model that predicts energy consumption based on the number of computations and memory access that GPGPU-Sim simulates. Together, both programs can be used to accurately model current and novel GPU architectures and DVFS controllers.

Leng *et al.* [31], the developers of GPUWattch, simulated the execution of compute and memory-bounded kernels in three scenarios: no DVFS and using a custom off-chip and on-chip DVFS. The custom DVFS algorithms monitor the average number of stall cycles caused by memory operations. When the number increases, the controller switches to a slower performance state of the core. Contrarily, when the number of stall cycles reduces, the controller places the GPU in a higher performance level. The difference between the on-chip and off-chip DVFS techniques is the time that it takes to respond to the variation of the number of stall cycles. While the on-chip can switch the performance level in 500 clock cycles, the off-chip takes 10000 cycles. The experiments show that using the off-chip DVFS versus no DVFS results in 13.2% of energy savings (on average) while using the on-chip DVFS yield a 14.4% energy saving.

Cha *et al.* [37] used GPGPU-Sim to create a GPU core space-multitasking simulator, where per-kernel dynamic frequency scaling (acting on the computing unit level) settings can be applied in concurrent kernel execution. They used Rodinia suite [36], Parboil suite [38], and Polybench suite [39] of benchmarks and combine the execution of different kernels, creating pairs of two compute-bounded (Com + Com) kernels, one compute-bounded plus one memory-bounded kernel (Com + Mem) and two memory-bounded kernels (Mem + Mem). The work evaluated the performance of the GPU by measuring the number of executed instructions per second. It was shown that for Com + Com concurrent kernel execution, the performance is lcore frequency of both kernels results in a 20.4% performance increase, while a 20% decrease results in a 19.3% decrease in performance. For (Mem + Mem) concurrent kernel execution, it was observed that the performance did not change significantly with the changes on the core frequency. The more interesting case is where mixed (Com + Mem) type kernels are concurrently executing. In this case, a per-kernel DVFS can overclock the CU of the Com kernel while underclocking the ones running the Mem kernel. In this setup, the highest performance is achieved for the Com kernel, and the energy (even though it was not the objective of the work) is minimized.

2.3.3 DVFS Optimization

As induced by the works presented in the previous section, by correlating the DVFS parameters with the application characteristics, it is possible to improve the performance and reduce the energy consumption of GPU accelerated programs. Under this assumption, the investigation on new DVFS mechanisms currently acts on two fronts: enabling a finer-grained DVFS control, with the creation of more clock/voltage domains within each GPU component; and creating novel DVFS control mechanisms, searching for more sophisticated and aware DVFS systems to better control the voltage and frequency

⁸<http://www.gpgpu-sim.org/>

⁹<http://www.gpgpu-sim.org/gpuwattch/>

parameters depending on the workload type. This section provides an overview of the state of the art on both development fronts. However, the second is of increased relevance to the context of the present dissertation.

Increased number of DVFS domains

Sethia *et al.* [40] designed *Equalizer*, a low overhead hardware runtime system, able to dynamically perform the monitorization and management of GPU resources and kernel requirements. This mechanism is placed in the instruction decoder pipeline attached to the kernel scheduler (scoreboard control mechanism that indicates which kernel should be run and where). By controlling the on-chip concurrency, together with the core and memory frequency, they can create two running modes (energy and performance modes) based on four counter utilization values (number of active and waiting threads, and number of ALU and memory instructions). In energy mode, it achieves 15% savings of energy consumption, while in performance mode, it can increase the performance by 22%.

In the work of Cha *et al.* [37], presented earlier, it is discussed how the application of DVFS at the CU level improves the performance and the energy consumption of the GPU. By using the GPGPUSim GPU simulator, they created a multi clock generator able to provide either a fast, a regular and a slow clock setup to each CU at demand. By reserving a register on each CU, the compiler will write the information regarding the most appropriate clock frequency. Each CU will inform the multi clock generator of which of the three clocks should be active. This approach showed the benefits of creating further specialized clock zones, enabling a finer grain of control over frequency and voltage. In the experimental work conducted by Cha, the finer-grained DVFS system was able to accelerate the compute-bounded kernels while still providing the more energy efficient frequency for the memory-bounded ones.

Optimal frequency search and optimization

Thomas *et al.* [41] proposed an Application-aware Scalable Architecture (ApSA) for GPGPU applications. ApSA is a three-stage runtime hardware profiling and scheduler mechanism able to adjust the operation of the GPUs core depending on the application's category under execution. In the first and second stages (*profiling* and *decision-making*), ApSA classifies applications as of type-I or type-II. An application is classified as type-I if it requires more processing cores to increase the performance and type-II if it needs more bandwidth from the memory system to run faster. According to this classification, on stage three (*action*) of this mechanism, the application is run on all available CU if it is type-I. If it is of type-II, the proposed mechanism only indicates that half of the available threads should run the application. At the same time, the frequency controller scales down the core and scales up the memory frequency, increasing the GPU's energy efficiency. By running the ApSA mechanism, a profiling overhead of 1.6% for type-I applications and 1.15% for type-II is introduced. Nevertheless, a reduction of 20.08% of power is achieved by using the ApSA mechanism.

Akiki *et al.* [42] proposes a run-time gradient descent (GD) optimal frequency search algorithm. This mechanism relies on multiple executions of the target application. In each iteration, an exploration of the optimal frequency is made. By indicating a given target metric, such as performance or Energy-Delay Product

$$EDP = energy * computation\ time, \quad (2.10)$$

it is possible to validate if the new frequency is better than the earlier one. By running this procedure alongside a set of benchmarks, with EDP selected as the target metric, it was achieved a reduction of 15% on energy consumption.

Huang *et al.* [43] introduced an novel proportional-integral-derivative neural network (PIDNN) frequency controller. This controller uses gradient descent to find the most appropriate frequency to be applied to the GPU core, memory and interconnect network to reduce energy consumption. The designed neural network has as input the current frequency of each GPU component, the number of kernels to be dispatched, the interconnect message queue size, and the number of caches misses. The hidden layers of the neural network represent the proportional-integral-derivative controllers, and the output layer corresponds to the new frequency to be set on the core, memory and interconnect. After the model is trained (and depending on the GPU model), the novel DVFS controller can reduce energy consumption between 4.39% and 18.67% on the tested benchmarks.

2.3.4 Decoupled V-F - Non-conventional DVFS optimization

The presented state of the art on DVFS exploration and characterization methodologies explored the benefits of using (mainly) frequency scaling to optimize the performance and reduce the power consumption on GPUs. However, the significant frequency and voltage guardbands that are usually adopted by manufacturers to ensure the correct operation of the devices across all possible working conditions, appear to be an extra optimization space waiting to be explored. More recent studies have been exploring this possibility, by working outside of the default V-F pairs. In this case, instead of just optimizing the frequency and relying on the default voltage values to perform DVFS, the studies also explore voltage scaling beyond the conventional DVFS, limits by trying to optimize both parameters for the running application.

This approach can enable a more significant energy efficiency degree of optimization, by allowing the GPU to be run at higher frequencies with reduced energy consumption. The studies presented in this section show that working in this unexplored space can be profitable when certain conditions are present. Due to the voltage reduction, the voltage guardband will be decreased. However, this voltage reduction makes the transistors slower, causing the GPU to be more prone to PVT variations.

Voltage guardband size estimation

To beneficially use voltage scaling outside conventional DVFS limits, it is necessary to understand how the size of the voltage guards and the minimum operating voltage (V_{min}) relate to the different types of applications. The work of Leng *et al.* [1] analyzes the voltage guardband of different applications and creates a statistical analysis procedure to predict the V_{min} (the best case voltage level, as defined in Figure 2.6(a)) depending on the collected performance counters. For the presented voltage guardband analysis, Leng *et al.* used 57 representative programs run on four different GPUs of two different architectures. The testing procedure consists of running each program 1000 times. After which a 12mV undervoltage is performed on the GPU core if every run is successful, repeating the procedure until a fault occurs. The faults can be of two types: runtime error, such as segmentation fault, silent data corruption or OS crash, and incorrect output (the results of the undervolt run being different from the run with default voltage).

Leng also tested the influence of process variation by running the benchmarks on multiple GPUs of the same model, achieving a maximum variability of 0.07V for the same benchmarks. The measured differences for process variation have a relatively low and uniform impact across all tested programs. This result indicates that this factor does not have a sufficiently high impact to be the leading root cause determining the V_{min} and guardband size.

By running the benchmarks mentioned above at two distinct temperatures (40°C and 70°C), Leng also tested the influence of temperature on V_{min} . The temperature change only caused a voltage guardband size variation of 0.02V among the two temperatures, allowing to conclude that there is no practical effect of temperature variation between these two temperatures.

Hence, the measured differences for process and temperature variation seem to have a relatively low and uniform impact across all tested programs. This result seems to indicate that these two factors do not have a sufficiently high impact to be the leading root cause determining the V_{min} and guardband size. The variability of aging was not possible to measure directly. However, it is plausible to assume that insignificant this effect should produce a similar contribution to process and temperature variation [1].

Overall, since the process and temperature variation (and device aging) do not have a substantial impact on the variability of V_{min} , by themselves, voltage noise appears to be the leading cause of this variation.

Following the work of Leng, Papadimitriou *et. al* [44] modeled the GPU voltage guardband in more detail using *GPUVolt* [45]. This modeling framework simulates the voltage noise behavior by calculating the time domain response of the power (voltage) delivery. Using *GPUVolt* in combination with *GPUWattch* to compute the power consumption, the author was able to determine that voltage droops can be originated due to inner and inter-core interference. In the single-core analysis, each component was evaluated to check their contribution to the voltage droop. The obtained results point out that the register file within each SM/CU is responsible for up to 70% of the occurrence of droops due to their high power consumption and high access rate. In fact, the inter-core interference exists due to the increased silicon area to accommodate the high core count of modern GPUs. The high core count can induce multiple high power consumption zones separated by inactive cores, inducing both fast-occurring first-order droops localized at small clusters of neighboring cores and slow-occurring chip-wide second-order droops. Going a step further, Papadimitriou relates the occurrence of these two phenomena with the type of running code. Inner-core voltage droops are mostly related to implicit synchronization mechanisms associated with SIMT, such as cache miss and thread block launch; while inter-core are mostly related to the launch of entirely new kernels.

Leng *et al.* [1] and Nakhaeaa *et al.* [46] propose methods to determine the V_{min} parameter with different objectives. While Leng aspires to reduce the energy consumption, Nakhaeaa expects to improve the lifetime of processors. These two examples are only a few of the possible benefits that the exploration of the voltage scaling brings. To model V_{min} , Leng used performance counter measurements to train an Artificial Neural Network () that predicts the amount of possible undervolt. The Root Means Square Error (RMSE) between the model prediction and the measured result is 0.5%, with the maximum over and underprediction error of 3% and 2%, respectively. Moreover, the trained ANN can correctly model the variation of V_{min} with the type of workload.

In general, a good model of the minimum operating voltage should be able to not only correctly predict

V_{min} across a significant large set of applications, but also minimize the under and overprediction of this value. Overpredicting the V_{min} range will most likely lead to the occurrence of program faults while underpredicting this voltage range may result in not adequately exploring the possible voltage values.

GPU undervoltage behaviour

The study of Tan *et al.* [47] explored the effects of using low supply voltage on the GPU register file. They supported this research by demonstrating that this component is one of the most affected and one of the first provoking program faults when not enough voltage is provided. Under this premise, Tan tested the minimum required voltage that guarantees that a write and a subsequent read of a register produces the same result. Due to process variation, this value can significantly differ from register to register. With this information, the author created an architectural solution that can analyze, for the runtime voltage level, which registers can be used and, by taking advantage of *dead-registers* (registers that contain useless data) the solution can forward the data from a faulty register to a working one.

The considered problem on the register file when decreasing the supply voltage is similar to the increased probability of a clock cycle ending without the critical path's fulfillment. In such situations, the processing elements of the GPU can misbehave and produce faulty computational results.

To conclude, it is generally necessary to estimate V_{min} to predict the size of the voltage guardband for the running application, and to understand the degree of computational errors that can emerge when taking advantage of non-conventional V-F pairs.

2.4 Undervoltage and Imprecision tolerant applications

Another interesting perspective is observed in certain applications that are herein referred to as *Imprecision Tolerant applications*. For these applications, computation errors can be propagated to subsequent logic stages with the expectation that the algorithm itself can recover from it [46]. Hardware designers can exploit these capabilities to reduce power consumption in two ways: designing hardware with lower precision encoding (reduced number of bits, like Google's TPU); minimizing the amount of combinatorial logic; or running traditional hardware (GPUs) in setups where the effort of reducing power consumption, may introduce a degree of imprecision.

Nowadays, a typical application that is taking, significant advantages of GPU acceleration is Deep Learning (DL). One of the most important characteristics of this type of algorithms is their imprecision tolerance. The computation of a deep learning application is an iterative and convergence process, and by the natural adaption of the neural networks learning parameters in runtime, the occurrence of small computation errors do not affect the end prediction of the algorithm [48].

Tang *et al.* [4] studied the impact of DVFS parameters on the energy and performance of DL applications. Even though the considered DVFS exploration only used the default range of frequency and voltage scaling, it was possible to either improve the performance by up to 33%, or reduce the energy consumption by 23.1%. However, this work applied the same pair of frequency and voltage throughout the complete training or inference session. Moreover, the authors point out that the different layers that compose a neural network can have different voltage and frequency optimization points. Furthermore, by using the inherent resilience of neural networks, it is expected that even better working parameters

outside of the conventional DVFS range can be found.

2.5 Summary

The present chapter introduces the fundamental concepts in order to use non-conventional voltage-frequency pairs on regular GPUs. It starts by providing an overview of current GPU architecture and programming model and establishing the relation between the executing kernel and the stress over the different GPU components. It follows by introducing the fundamental notions of CMOS circuits when dealing with frequency and voltage scaling, exacerbating the importance of choosing the most appropriate supply voltage to guarantee circuit stability and protection to variations while trying to achieve the best energy-efficiency possible.

Following the bottom-up approach, the chapter builds on the CMOS circuit characterization to introduce the power management technique and mechanism DVFS. This mechanism has the objective of selecting the most appropriate V-F pairs according to the current GPU state. However, the literature considers that current approaches have two problems. First, they are conservative when scaling the voltage according to the frequency, leaving a substantial guardband that reduces the energy-efficiency of the devices and second, it does not take into account the executing application, neither their current state nor the GPU circumstance.

Grounded with these problems, the following chapters will try to improve the current state of the art by proposing a methodology to explore the voltage guardband and dynamically tune the voltage and frequency to the application being executed.

Chapter 3

GPU architectural characterization to decoupled V-F

The first objective of this thesis is to provide a methodology to uncover the use of non-conventional voltage-frequency pairs on regularly deployed GPUs to improve their energy efficiency. This chapter introduces the developed methodology used to assess the usability and benefits of non-default voltage values on top of the traditional device frequency scaling.

The developed methodology allows for:

- The determination of which GPU components establish the voltage guardband size;
- The identification of the architectural component responsible for the wrong application output (computational error or memory corruption);
- The determination of the relation between the application of non-conventional V-F scaling on both DVFS domains and the resulting performance, power and energy consumption, as well as energy efficiency.

By following the workflow depicted in Figure 3.1, the chapter introduces a set of benchmarks that stress each individual GPU DVFS domain and component to find V_{min} (see Section 3.1), the frequency-dependent minimum operating voltage that (still) leads to correct GPU operation. It continues by presenting the experimental setup and testing procedure on two different GPU architectures, the AMD Vega 10 Frontier Edition and AMD Radeon 5700 XT (see Section 3.2). The establishment of a usable voltage range across the frequency spectrum (presented in Section 3.3) allows for the creation a voltage-frequency exploration and usable space (top right chart of Figure 3.1). The chapter finishes with Section 3.4, with an evaluation of the performance, energy consumption and Energy Delay Product (EDP), for the given benchmarks when subject to non-conventional V-F scaling (bottom right chart of Figure 3.1). Even though the delay of a CMOS circuit is inversely proportional to the supply voltage V_{DD} , the dynamic power is proportional to the square of V_{DD} (and so, energy consumption and EDP will be directly proportional to V_{DD}), computing the EDP brings the additional benefit of directly analyzing the effect of voltage change in the energy efficiency of the device.

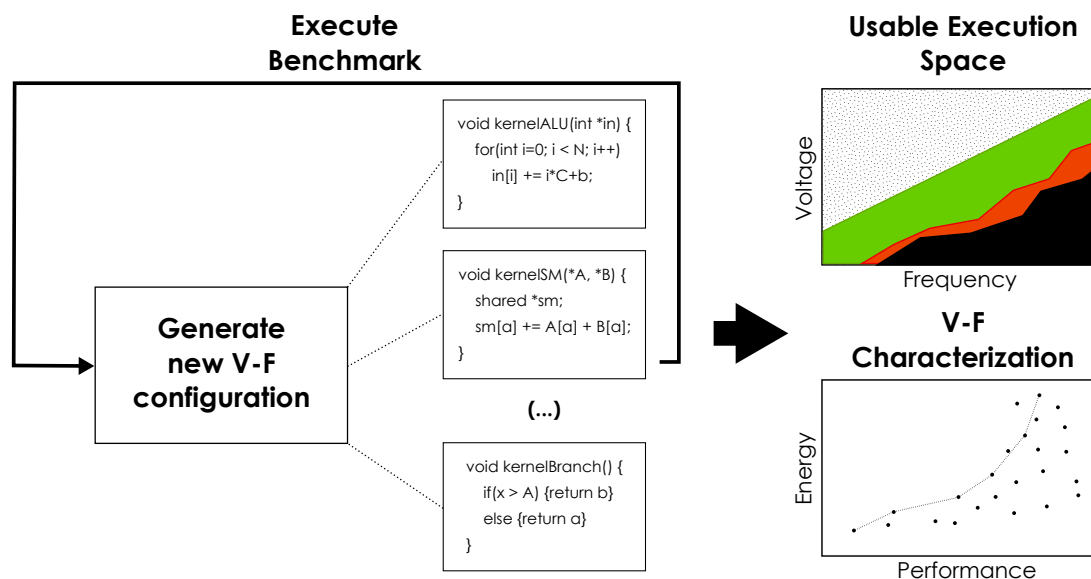


Figure 3.1: Overview of the methodology and its objective outputs.

3.1 Characterization Benchmarks

The devised set of benchmarks, presented in Table 3.1 and made publicly available as open-source¹, individually characterize the different components of the GPU architecture when subjected to the two DVFS domains: *core* and *global memory*. The tested architectural components are DRAM, Shared Memory, Cache L2 and ALU. In more detail, the memory experiments cover the reading and writing operations, as well as the prolonged effects of undervoltage on memory retention, while the ALU tests include the Multiply and Accumulate (MAC) and non-linear operations, as well as the impact of branches. Overall, the developed benchmarks were crafted not only with the intention of stressing the individual components, but doing so in a way that helps to answer the questions presented at the beginning of this chapter. Additionally, a coarser and more representative kernel that stresses multiple architecture elements in many GPGPU applications - the reduction - is also evaluated.

Table 3.1: Devised set of kernels to characterize GPU to Non-Conventional DVFS

Micro-kernels	Data Type	Objective
DRAM	FP32, INT32	Minimum Read & Write voltage, bit-flip, data-corruption Effect of memory to compute bounded kernel on Core DVFS
Cache L2	INT32	Minimum Read & Write voltage, data-corruption
Shared Memory	INT32	Minimum Read & Write voltage, data-corruption
ALU	FP64/32/16, INT64/32/16/8	Computation errors due to timing violations
SFU	FP64/32/16	Computation errors due to timing violations
Branch		Minimum voltage for correct scheduling operation
Mix (reduction)	FP64/32/16	Simultaneous impact of stressing multiple GPU components

Of the listed benchmarks, the placeholder `DATA_TYPE` is used to represent the different tested data types. By following Table 3.1, this generic keyword is replaced by a standard integer and half, single

¹github.com/hpc-ulisboa/nonconventional-dvfs

and double precision data types, depending on the benchmark. The objective is to assess the effect of changing the type of operands and resulting effect on the computation precision on the critical path.

To guarantee that no compiler optimizations are performed on the tested variables, the keyword `volatile` is used. This keyword informs the compiler to not optimize the current variable, guaranteeing that the way that the code is executed, does not change because of the compiler.

DRAM

The benchmark on Listing 3.1 was designed (and validated through GPU counters) to determine the impact of V-F scaling on the GPU DRAM when executing a compute-bounded kernel.

In the presented kernel, each thread is responsible for accessing the global memory and retrieving two values. The accesses are sequential between threads to guarantee the maximum memory throughput and no accessing hazards. These two values are summed and placed on an output vector. A constant value, C determines the distance between accesses, and its value is sufficiently large to guarantee that the new data to be fetched is not present on the local caches. For each data fetch, the defined `OPS` value controls the number of arithmetic operations to be performed before the data is written back on the DRAM. A lower `OPS` value decreases the time between memory access, resulting in a more memory intensive kernel, that depending on the global memory bandwidth and throughput, can become memory-bounded. On the other hand, a higher `OPS` value results in the memory accesses to be more spaced in time, leading to a less memory intensive kernel and, eventually, to a compute bounded kernel.

The DRAM DVFS domain is responsible for controlling the voltage and frequency of the GPU global memory. This DVFS domain affects the memory throughput both on the reading and writing operations. With the characterization of this DVFS domain, it is possible to judge the weight of the global memory on the overall power and energy consumption and the impact of DRAM performance when executing a more memory-bounded kernel. On the other hand, by assessing the GPU performance and energy consumption, with the same kernel, while acting on the core DVFS domain, it provides an understanding of how to best tune the core when the performance bottleneck is not within it.

```
void DRAMcode(DATA_TYPE *IN0, DATA_TYPE *IN1, DATA_TYPE *OUT) {
    const int ite = (blockIdx.x * THREADS + threadIdx.x) % MEM_BLOCK;
    volatile register DATA_TYPE r0;

    #pragma unroll
    for (int i = 0; i < N; i++) {
        r0= IN0[i * C + ite] + IN1[i * C + ite];
        #pragma unroll
        for(int j = 0; j < OPS; j++)
            asm volatile ("");
        OUT[threadId] = r0;
    }
}
```

Listing 3.1: DRAM Benchmark Code

Listing 3.2 renders a benchmark that was specifically designed to evaluate the occurrence of the phenomenon called *bit-flip* and the preservation of the data in memory when exposed to undervoltage. A *bit-flip* is an unintentional state switch (from 0 to 1, or vice versa) of any individual bit stored on a DRAM or other kinds of volatile memories. The work of Kim [49] exposed the existence of *bit-flipping* on CPU DRAM, induced by the continuous activation of a DRAM row that corrupts the data in near-by rows.

The work was of such significant importance that the benchmark called *rowhammer*² to test this exact problem started to be of severe importance to guarantee data integrity in novel systems. The benchmark on Listing 3.2 is a GPU implementation of *rowhammer* that is going to be used to assess if undervolting the GPU can increase the possibility of such problem.

```

void DRAMstresser(DATA_TYPE *IN, DATA_TYPE *OUT) {
    const int ite = threadIdx.x;
    volatile register DATA_TYPE r0;

    // Initiate output memory
    OUT[ite] = IN[ite];
    OUT[ite + THREADS * BLOCKS] = IN[ite + THREADS * BLOCKS];

    for (int i = 0; i < N; i++) {
        r0 = IN[ite];
        #pragma unroll
        for(int j = 0; j < OPS; j++)
            asm volatile ("");
        OUT[ite] = r0;
    }
}

```

Listing 3.2: DRAM Bit-Flip Stress Test Code - *rowhammer* inspired benchmark

For the memory tests (DRAM, Cache and Shared Memory), only the integer data type was considered, since the effects on memory are the same for floating-point operations and it is easier to identify data corruption with integer data types.

Cache

As previously stated, even though the caches are part of the memory subsystem, they are under the *core* DVFS domain. The devised benchmark, presented on Listing 3.3, follows a similar stressing pattern to Listing 3.1. However, with the addition of the external *k* loop, the access pattern is repeated, and so, after the first execution, the data will be available on one of the two levels of cache. Hence, this kernel is then able to test both the state machine responsible for establishing the communication between the cache and the DRAM and the cache itself (results verified with GPU counters).

For this benchmark, the number of issued requests to the cache and to the DRAM-cache controller stays the same independently of the OPS value. However, the frequency of those requests is inversely proportional to OPS.

Shared Memory

The proposed benchmark to characterize the Shared Memory is presented in Listing 3.4. This component is shared between threads in the same CU/SM and is used to ensure the communication between the different executing threads. As so, the developed benchmark uses this component to move data around. Similarly to the DRAM and cache kernels, the OPS variable controls the time distance between memory requests, allowing to control the level of stress over this component and while assessing its behaviour. To guarantee a correct and repeatable execution, the synchronization directive `__syncthreads()` is used to synchronize all the threads that use the same shared memory.

²github.com/google/rowhammer-test

```

void CacheL2code(DATA_TYPE *IN, *OUT) {
    const int ite = blockIdx * THREADS + threadIdx;
    volatile DATA_TYPE r0;

    for (k=0; k<N; k++) {
        #pragma unroll
        for(j=0; j<COMP_ITE; j++) {
            r0= IN[ite];
            #pragma unroll
            for(m=0; m<OPS; m++)
                asm volatile ("");
            OUT[ite] = r0;
        }
    }
}

```

Listing 3.3: CacheL2 Benchmark Code

```

void SharedMemorycode(DATA_TYPE *IN, DATA_TYPE *OUT) {
    __shared__ DATA_TYPE shared[THREADS];
    const int ite = blockIdx * THREADS + threadIdx;
    const int t = threadIdx.x;
    const int tr = THREADS - t - 1;

    volatile register DATA_TYPE r0 = IN[ite];

    for (int i = 0; i < N; i += UNROLL_ITE) {
        #pragma unroll
        for(int j = 0; j < UNROLL_ITE; j++)
            shared[t] = r0;
        __syncthreads();
        for(int k = 0; k < OPS; k++)
            asm volatile ("");
        r0 = shared[tr];
        __syncthreads();
    }
    OUT[ite] = r0;
}

```

Listing 3.4: Shared Memory Benchmark code

Arithmetic and Logic Unit (ALU)

The arithmetic and logic unit (ALU) is responsible for performing all the GPU computations. Being the component that performs such a different number of operations, a set of benchmarks was devised to stress this component in different ways. Overall, the focus of testing this component was on the understanding the degree of computational errors that may occur when overly undervoltage is applied, these resulting from timing violations across the critical path. Of significant importance when investigating the timing faults on the critical path is to test the influence of data dependencies in the code, as these may influence how the warps scheduler orders the threads for execution on the CU/SMs. The devised benchmarks that test the ALU were designed to use all the available threads on the GPU.

In Listing 3.5, a greater emphasis was devoted to the MAC operation due to its prevalence in the Deep Learning (DL) domain. To test the influence of data dependencies on the application, and so the way the scheduler handles the execution of the different threads, a value between 0 and 5 is assigned to variable d . When $d = 0$, no dependencies exist in the code. The setup with $d = 1$ represents the worst-case scenario, since it introduces Read-after-Write (RaW) dependencies between all operations. This particular dependency setup was emphasized in the presented study, due to the prevalence of such type of data dependencies in kernels executed by DL workloads. On the other hand, the setup with $d = 3$ was considered a general case, where some dependencies still exist in the code, but the scheduler can

mask some of them.

```
void MACcode(DATA_TYPE *IN, DATA_TYPE *OUT) {
    const int ite = (blockIdx * THREADS + threadIdx) * 4;

    volatile DATA_TYPE r0, r1, r2, r3, r4, r5;

    r0=IN[ite]; r1=IN[ite+1]; r2=IN[ite+2];
    r3=IN[ite+3]; r4=IN[ite]; r5=IN[ite+1];

    for(j=0; j<COMP_ITE; j++) {
        r0 += r0 * r{0-d};
        r1 += r1 * r{1-d};
        r2 += r2 * r{2-d};
        r3 += r3 * r{3-d};
        r4 += r4 * r{4-d};
        r5 += r5 * r{5-d};
    }
    OUT[ite/4] = r0;
}
```

Listing 3.5: MAC Benchmark Code

Non-Linear Operations

Besides the MAC operation, the ALU also computes a set of non-linear functions, including exponential, logarithmic and trigonometric operations. For such purpose, it uses the special function unit (SFU). The devised benchmark presented in Listing 3.6 tests those operations to find if the undervoltage mechanism, when in use, introduces some modifications in the critical path and so, negatively influences the guardband size.

```
void NonLinearcode(DATA_TYPE *IN, DATA_TYPE *OUT) {
    const int ite = (blockIdx * THREADS + threadIdx) * 4;

    volatile DATA_TYPE r0, r1, r2, r3;

    r0=IN[ite]; r1=IN[ite+1]; r2=IN[ite+2]; r3=IN[ite+3];

    for(j=0; j < N; j+= UNROLL_ITE) {
        #pragma_unroll
        for(j=0; j < UNROLL_ITE; j++) {
            r0 = exp(r2);
            r1 = cos(r3);
            r2 = log(r0);
            r3 = sin(r1);
        }
    }
    OUT[ite/4] = r0;
}
```

Listing 3.6: Non-linear Operations Benchmark Code

Branches

Listing 3.7 renders the devised benchmark to tests the influence on the execution of branches on a kernel. SIMT processors do not favor the existence of branches on code, since it prevents the simultaneous execution of all threads. The scheduler's job is to organize the running threads in wavefronts to be simultaneously executed depending on the execution path dictated by the branches on the kernel. An

increased number of branches makes the job of the scheduler more difficult. So, the developed benchmark analyses the influence of reducing the supply voltage on the scheduler operation. On Listing 3.7, the `#define BRANCHES` directive sets the desired number of branches to test between 1, 2, 4 and 8.

```
#define BRANCHES VALUE
void Branchescode(DATA_TYPE *IN, *OUT) {
    const int ite = (blockIdx * THREADS + threadIdx) % MEM_BLOCK;
    const int branch = ite % BRANCHES;

    volatile register DATA_TYPE r0, r1, r2, r3;

    for (int i = 0; i < N; i++) {
        if(branch == 0)    r0 = IN[ite];
        #if BRANCHES >= 4
            else if(branch == 1)    r1 = IN[ite];
            else if(branch == 2)    r2 = IN[ite];
        #elif BRANCHES == 8
            else if(branch == 3)    r3 = IN[ite];
            else if(branch == 4)    r0 = IN[ite];
            else if(branch == 5)    r1 = IN[ite];
            else if(branch == 6)    r2 = IN[ite];
        #elif BRANCHES >= 2
            else {r3 = IN[ite];}
        #endif
        OUT[ite] = r0;
    }
}
```

Listing 3.7: Branches Benchmark Code

Reduction

The reduction benchmark, presented in Listing 3.8 performs the reduction of a N -sized vector to $N/blockDim$ elements, by performing an element-wise sum. The tested implementation of this operation is considered the one that achieves the highest performance, and so, it is the most widely used. It makes use of the shared memory to enable inter-thread communication and improve performance. Hence, this benchmark stresses all elements of the architecture (DRAM, Cache, shared memory and ALU) and allows to assess a more complex use-case, where a single kernel stresses multiple architectural units.

This kernel's objective is to test if this component is responsible for the observed behavior when applying the undervoltage mechanism, or if higher-order interdependencies exist between the stressed units.

```

void Reduction(DATA_TYPE * idata, DATA_TYPE * odata){
    __shared__ DATA_TYPE s[THREADS];
    unsigned int i, k, t = threadIdx;
    unsigned int index = blockIdx * blockDim * N + threadIdx;

    // cooperative load from global to shared memory
    s[t] = 0;
    for (i=0; i< 4; i++, index += blockDim.x)
        s[t] += idata[index];
    __syncthreads();

    // do reduction in shared memory
    if(t < 64) {
        s[t] += s[t+64];
        __syncthreads();
    }

    if(tid <32){
        s[t] += s[t+32];    s[t] += s[t+16];
        s[t] += s[t+8];    s[t] += s[t+4];
        s[t] += s[t+2];    s[t] += s[t+1];
    }

    // write result for this block to global mem
    if(t == 0) odata[blockIdx.x] = s[0];
}

```

Listing 3.8: Reduction Kernel Code

3.2 Experimental Setup and Methodology

The devised benchmarks were applied to characterize two GPUs with different architectures: the AMD Vega 10 Frontier Edition and the AMD Radeon 5700 XT, whose specifications are presented in Table 3.2.

Table 3.2: Considered GPUs in the conducted experimental characterization.

Model	Unit	Vega 10	Radeon 5700 XT
Architecture		GNC5	RDNA
CUs		64	40
DRAM size	[GB]	16	8
Default Power Cap	[W]	220	190
Core frequency range	[MHz]	[852 - 1980]	[800 - 2050]
Core voltage range	[mV]	[900 - 1200]	[750 - 1200]
DRAM frequency range	[MHz]	[500 - 1200]	Fixed at 1000
DRAM voltage range	[mV]	[800 - 1200]	Fixed at 1000
Default Frequency-Voltage (F-V) setups			
Core F-V	[MHz ; mV]	[995;900, 1140;950, 1350;1050, 1440;1100, 1530;1150, 1600;1200]	[1200;950, 1400;1000, 1600;1050, 1800;1100, 2000;1200]
DRAM F-V	[MHz ; mV]	[500;900, 800;950, 950;1000]	[1000;1000]

As it was described in Section 2.3, the DVFS system automatically selects, according to the GPU workload, power and temperature, a voltage-frequency pair from the ones presented in Table 2.1. However, through the use of the GPU vendor ROCm System Management Interface (rocm-smi³) tool, it is possible to independently control the frequency and voltage values of the performance levels. The following section presents the capabilities of the software tool and testing methodology. Annex A provides a more comprehensive description of how rocm-smi is used to set the target voltage frequency pair.

³github.com/RadeonOpenCompute/ROC-smi

Finally, to ensure that the GPU power cap does not block any of the intended configurations, its value is changed to match each GPU thermal design cap (220W to 300W for the Vega 10 and 190W to 285W on the Radeon 5700 XT). The GPUs under test are installed on a machine equipped with an Intel i7 4770K CPU, with 32 GB of main memory.

3.2.1 Voltage and Frequency control API

The ROCm System Management Interface (rocm-smi)⁴ is a user-friendly command-line application for manipulating the Radeon Open Compute Kernel (ROCK). This tool makes it possible to know and control the state of the GPU devices present in the system.

- **GPU utilization:** Retrieves the current utilization rates corresponding to the device's major sub-systems, one value for the processing core and other for the main device memory. The rate is computed over a specific time interval set on the device driver. The processing core utilization reflects the percentage of time that the GPU core was being used to perform computations. In contrast, the main device memory utilization reflects the percentage of time the memory was being read or written.
- **GPU power:** Retrieves the average power used by the device. Similarly to the utilization rate, the average power is computed over a defined time interval, during which a number of power samples are taken.
- **Clock rate and voltage level:** Retrieves both the current clock frequency and device voltage level. Of significant importance is the fact that the retrieved voltage corresponds to the maximum measured value between the GPU core and the main device memory voltage. Current versions of AMD GPUs do not allow for querying the specific voltage value of the different DVFS domains.

The tool also displays the performance level tables for the Core and DRAM DVFS domains.

The rocm-smi interface also allows querying the device temperature, the current fan speed, and the selected performance level.

rocm-smi also provides a mechanism to control and change some of the device parameters:

- **Set performance profile:** Allows the user enable/disable the automatic DVFS system. When disabled, the GPU adopts the user-defined voltage and frequency pair.
- **Set clock rate and voltage level:** Set the clock frequency and the voltage level of any of the performance levels of both the GPU core and memory.
- **Reset clock rate and voltage level:** Resets the clock rates and voltage levels to the default values.

Additionally, the interface allows the user to manually set the fan speed that is required to guarantee the same temperature level for all executed tests.

The versatility and ability to independently control the clock rate and voltage level of the device, enabled by rocm-smi, was the defining factor for choosing an AMD GPU over the more popular NVIDIA options.

⁴github.com/RadeonOpenCompute/ROC-smi

3.2.2 Testing procedure

The default frequencies of the GPU *Core* and *DRAM* domains, presented in Table 3.2, were selected as the starting point for the proposed non-conventional DVFS. For each frequency, the devised experiment started at the maximum voltage ($1200mV$) and a gradual undervoltage of the GPU V-F domain under test was applied with $50mV$ steps. For each step, the benchmarks were executed ten times to obtain the median value of the execution time and energy consumption.

All the tests were performed using randomly generated inputs to avoid any bias incurred by the considered data values. Integer values were obtained from a normal distribution across their complete 32-bits range, while floating-point operands were generated using a uniform distribution in the interval $[0.1 ; 1]$. The choice for limiting the floating-point range to an interval with a maximum value inferior to one ensures that operations are never applied to numbers with a significantly different exponent value, thus avoiding rounding errors that would conduct to the discard of the operator with the lowest absolute value.

While performing the undervoltage, the GPU goes through three distinct stages. At the first stage (*working*), the GPU works regularly, and no changes are detected in the application output. Then, by continuing the GPU voltage reduction, some *computational errors* are introduced and some application outputs change when compared with the default voltage setup (the output comparison between conventional and non-conventional execution is performed on the CPU). For the integer experimentation, a computational error is asserted if the output value differs from the default run; for the floating-point operation, the relative error between the default and non-conventional run is computed for each individual result. When the relative difference between the above is greater than 10^{-4} , it is said that the result has *computational errors*. By reducing the GPU voltage beyond this stage, the GPU enters into the *crash* state, becoming unusable.

To accurately determine the areas of interest (i.e., when only infrequent computation errors occur) and to determine the crash point, the undervoltage step was reduced to $10mV$. Furthermore, when dealing with the *DRAM* V-F domain, the *Core* V-F domain was set to its default values; during the characterization of the *Core* V-F domain, the highest frequency and default voltage of the *DRAM* domain was selected.

The GPU power consumption was measured using `gpowerSAMPLER`⁵ [50], at every millisecond. At the end of the execution, the energy is computed as the integral of all the taken measurements.

3.3 Limiting components to the voltage exploration space

The execution of the different benchmarks while controlling the V-F values provided usable information about the voltage guardband each component has. More specifically, it allowed to explore and quantify how far it is possible to decrease the operating voltage from each default frequency level while measuring the impact on the application output and working capabilities of the GPU device.

This section presents the voltage margin results for every architectural component of the Vega 10 and Radeon 5700 XT GPUs. The Radeon 5700 XT only has the *Core* DVFS domain, so the results of the *DRAM* DVFS domain were only obtained on the Vega 10 GPU.

⁵github.com/hpc-ulisboa/gpowerSAMPLER

For the Core domain, only the frequencies above and equal to 1440MHz (for the Vega 10) and 1600MHz (for the Radeon 5700 XT) are shown in order to reduce the charts size. For all frequencies below, it was found that the GPU could be run at any voltage from the default to the minimum (900mV for the Vega 10 and 750mV for the Radeon 5700 XT).

3.3.1 DRAM

Figure 3.2 illustrates the usable voltage range of the DRAM domain on the Vega 10 GPU. Across the complete and tested frequency range, it is possible to undervolt the memory to 800mV, independently of the value of OPS parameter (varying between 0 and 50 operations - see Listings 3.1). The conducted experiment shows that no computation error or crashes happen for the default frequencies within the complete voltage range. The kernel runs successfully, with no perceptible change in the output.

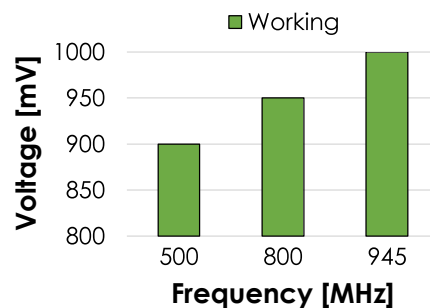


Figure 3.2: Vega 10 - DRAM domain - Usable voltage for each frequency configuration.

The result and the conclusion derived from the previous experiment was further verified with the execution of the kernel of Listing 3.2. It is of extreme importance to validate if the use of lower voltage values induces data corruption on two edge cases: extreme use of part of the DRAM and prolonged data storage. That said, the kernel was executed, and after time intervals of 1, 2, 5, 10, 30 minutes and 1, 2, 4 and 8 hours, the output data was retrieved and compared to the original set. Again, no perceptible change in the output was detected, further validating the previous result and assessing that the DRAM can be successfully used with lower voltage levels.

3.3.2 Cache

Figure 3.3 presents the usable voltage interval at different frequency setups for both GPUs. For the Vega 10 GPU, no computation errors or crashes were observed at frequencies below 1530 MHz. The GPU operates correctly works at the lowest voltage across all the tested frequencies. Only for frequencies higher than 1530MHz the undervoltage resulted in the program crash. A critical observation is that no computation errors occur, meaning that this architectural component either works normally or makes the GPU crash. This phenomenon is of significant importance to identify the root cause of failures: whenever it is observed a GPU crash without prior computation errors, this may be the preeminent component causing it. Furthermore, an increase of the OPS parameter allows for a higher amount of undervolt. Since this change only affects the stress over the DRAM-Cache controller (the number of cache accesses and hit-rate maintains the same), it can be concluded that it is the Cache-DRAM controller that limits the undervoltage range and not the memory elements of the cache.

For the Radeon 5700 XT a similar behaviour is observed. For frequencies below 1600 MHz, it is possible to use the GPU at the lowest voltage without any computation errors or crashes being observed. For higher frequencies, when the voltage is reduced the most, the GPU can crash. Increasing the value of the OPS parameter (decreasing Cache stress) increases the undervoltage capabilities.

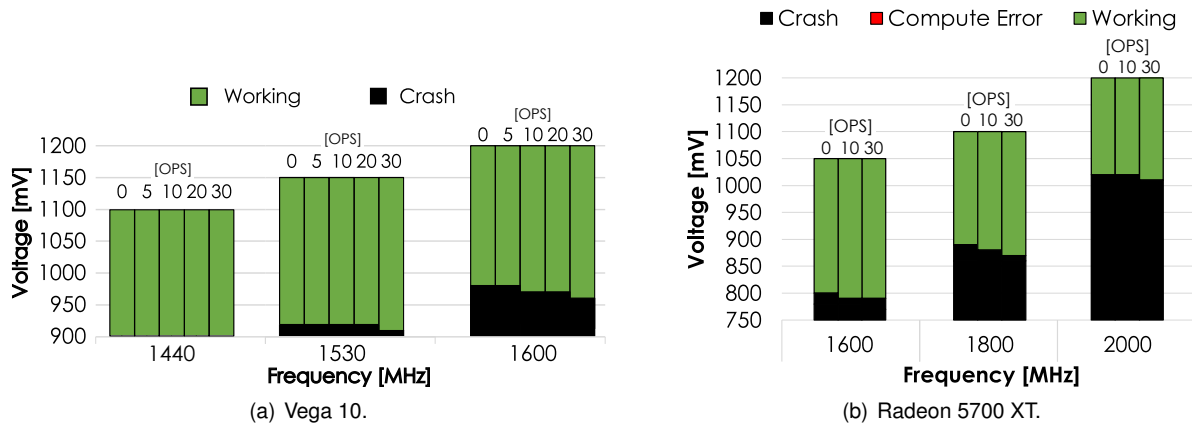


Figure 3.3: Core domain - Cache L2 - Usable voltage for each frequency configuration with varying cache stress (*OPS* value).

3.3.3 Shared Memory

The obtained voltage guardband results for the Shared Memory are presented in Figure 3.4. Similarly with the Vega 10 Cache results, the benchmark's output was not affected by the applied undervoltage, for frequencies below 1530MHz, with the GPU operating correctly in the complete voltage range. For the frequencies above and equal to 1530MHz, both computation errors and crashes occur. With the increase of the OPS parameter, from 0 to 10, it is observable that the GPU starts allowing an increase of 10mV of undervoltage. This phenomenon indicates (and was also confirmed with GPU counters) that with the reduction of the shared memory stress, the limiting factor to V_{min} switches from the shared memory to the ALU (result confirmed ahead). The conclusion was further validated for the test at 1600MHz, where increasing OPS to 40 makes the GPU start crashing, meaning that the ALU is now the limiting factor.

For the Radeon 5700 XT, a similar behavior is found. However, even at the test with OPS=40 the limiting factor is still the Shared Memory. This is mainly due to a reduced memory interface and to a lower maximum memory bandwidth of the Radeon 5700 XT (256-bit and 448GB/s), contrasting to 2048-bit and 484GB/s of the Vega 10 GPU.

3.3.4 Arithmetic and Logic Unit

Figure 3.5 represents the usable undervoltage range for the ALU benchmark. This benchmark was executed either for integer and for the several different floating-point precisions available on each GPU.

On the Vega 10 GPU, and in all considered scenarios using an operating frequency below 1440 MHz, the benchmark was successfully run for all voltage values. For higher frequencies, it is observed that some computation errors start appearing after a certain amount of undervoltage. In particular, the GPU crashes if a further undervoltage level is applied. It is also observable that the voltage margin



Figure 3.4: Core domain - Shared Memory - Usable voltage range for each frequency configuration with varying shared memory stress (*OPS* parameter).

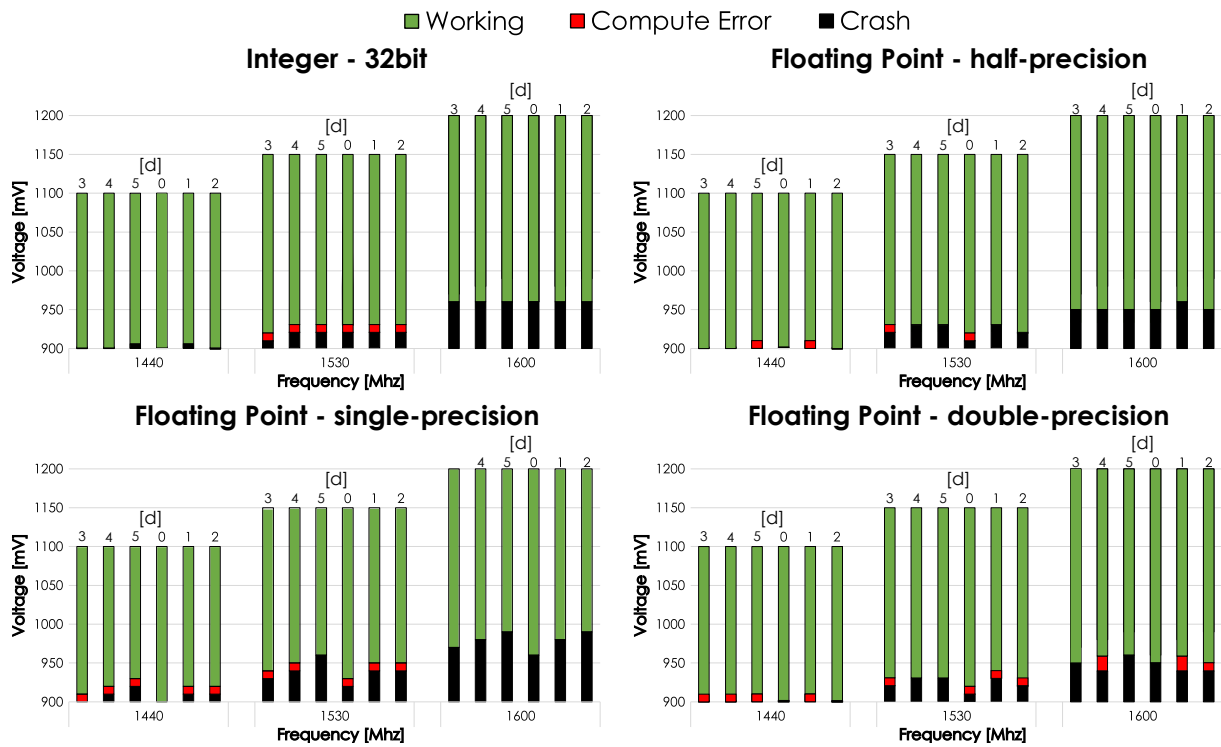
increases with the operating frequency, from around 170mV for 1440 MHz to around 210mV for 1600 MHz (values for single-precision floating-point). On the Radeon 5700 XT GPU, the ALU benchmark works correctly independently of the applied voltage, for frequencies below 1600 MHz. At higher frequencies, and similarly to what happens with the Vega 10, a computation error margin exists when undervolting, before the occurrence of crashes. Overall, the Radeon 5700 XT GPU allows more than 200mV of safe undervoltage across the complete frequency spectrum.

In general, it was observed that the operands that are "easier" to compute (namely the integer and half-precision) allow the highest amount of undervoltage, not being the computation of those the limiting factor of the ALU. In fact, it was expected that when comparing floating-point single precision with double precision, the second would have a worse undervolting capabilities, due to having the double amount of bits to compute. However, the opposite relation was observed, with double-precision allowing for an extra 10 to 20mV of undervoltage (when comparing the crash point). This leads us to conclude that single and double floating-point computation is not performed in the same way. In fact, although most RISC microprocessors are pipelined, the way the pipeline stages are divided is not the same. It is reasonable to expect that the double-precision computation is split among more clock cycles than the single precision, allowing for an extra relaxation of the timing constraints on that data path.

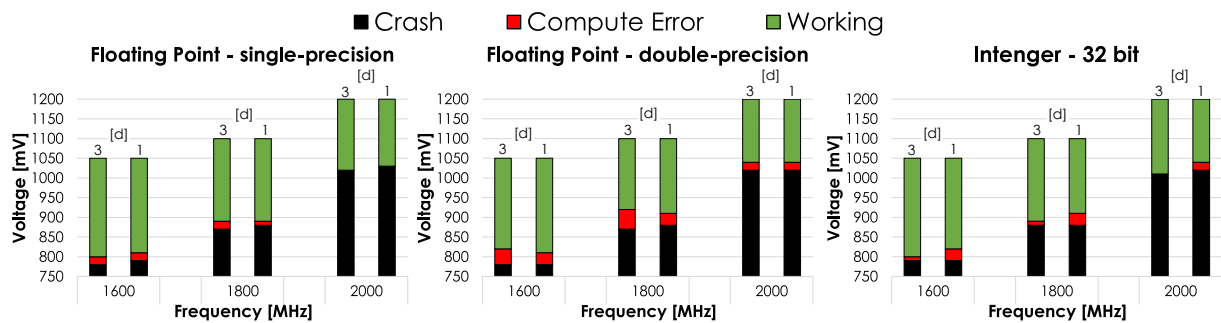
The dependencies effect also varied from integer to floating-point operation. In the first case, the code's dependencies do not affect the size of the voltage margin. In the second case, the existence of dependencies in the code reduces the voltage margin's size. In general, when compared with the setup with no dependencies ($\alpha=0$), the undervoltage range of the benchmark configuration that represent the general case ($\alpha=3$ - see Listings 3.5) is reduced by 10mV.

3.3.5 Non-linear Operations

The special function unit (SFU) was also tested for floating-point operation of single and double-precision, with the results being presented in Figure 3.6. Similarly with the results of the MAC operation, double precision allows a further 10 to 20mV of valid undervoltage, when compared to single precision. Overall, this unit's voltage guardband is similar to the one of the ALU on the MAC operation.

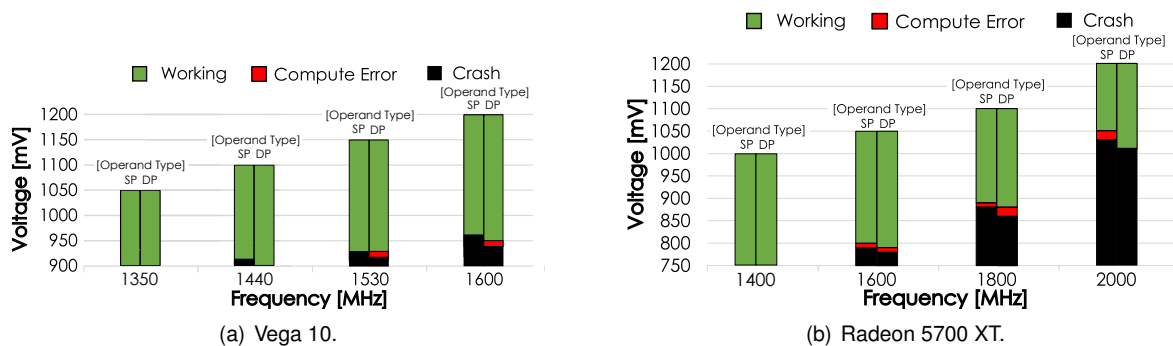


(a) Vega 10.



(b) Radeon 5700 XT.

Figure 3.5: Core domain - ALU-MAC - Usable voltage margins for the different data types and operand's precision.



(a) Vega 10.

(b) Radeon 5700 XT.

Figure 3.6: Core domain - Special Function Unit - Usable voltage for each frequency configuration with varying operand type: SP - Single Precision Floating-Point, DP - Double Precision Floating-Point.

3.3.6 Branches

Figure 3.7 presents the results of the branches test. As it can be observed, the number of branches on the kernel did not affect the voltage guardband's size, with all configurations (for each operating frequency) allowing the same degree of undervoltage. Hence, the obtained result allowed the conclusion that branch miss-prediction does not negatively impact V_{min} . It is expected that, independently of the number of branches that a kernel has, the limiting factor to V_{min} is the type of operations being executed, leading to computational errors and eventually to GPU crash as determined by the other experiments.

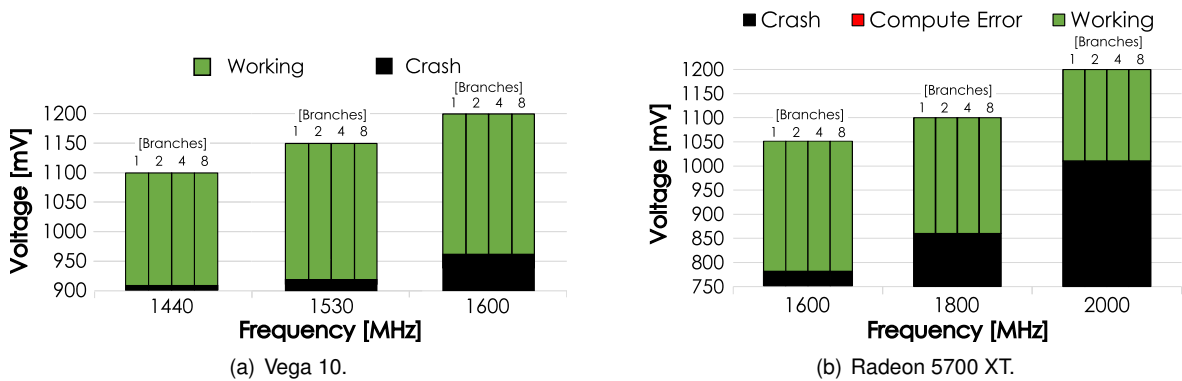


Figure 3.7: Core domain - Branches - Usable voltage for each frequency configuration with varying number of branches per iteration.

3.3.7 Reduction

Figure 3.8 presents the usable voltage range for the `reduction` benchmark. As can be seen, due to the high pressure exercised on the cache by this benchmark, the minimum usable voltage coincides with the one already measured for the Cache benchmark. The ALU benchmark can explain the several data types range of compute errors and V_{min} value differences, with double-precision floating-point having a bigger margin of computational errors (on the Radeon 5700 XT) and overall reduced voltage guardband at the higher frequencies on both GPUs.

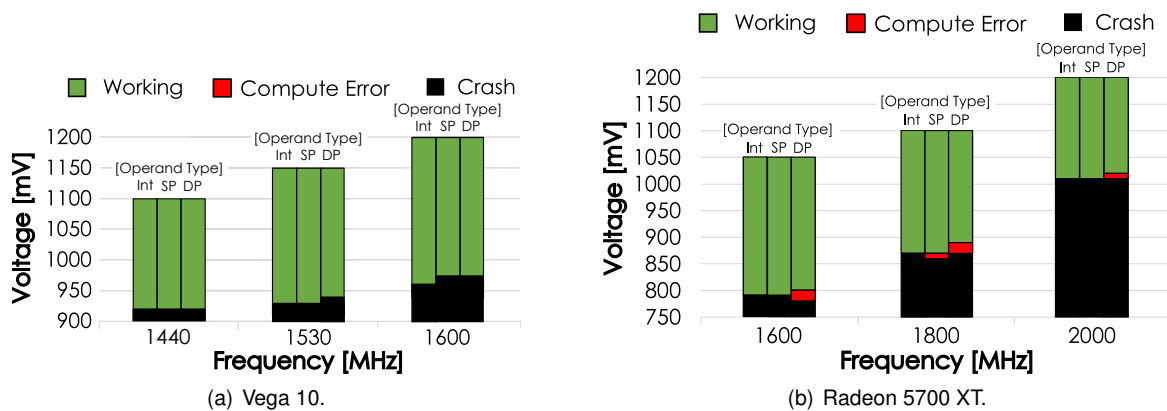


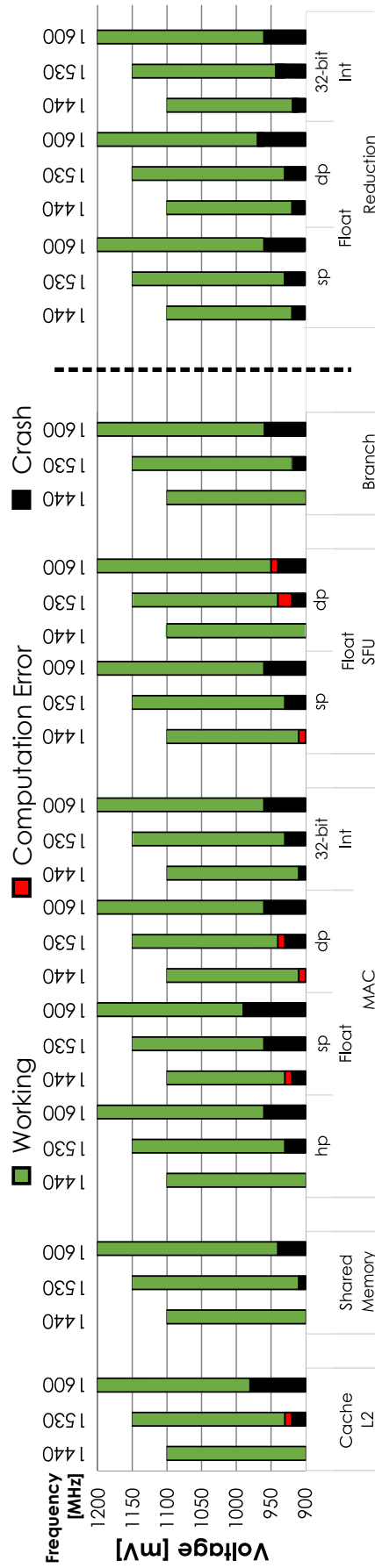
Figure 3.8: Core domain - Reduction benchmark - Usable voltage for each frequency configuration with varying operand type: Int - 32-bit Integer, SP - Single Precision Floating-Point, DP - Double Precision Floating-Point.

3.3.8 General Comments and Remarks

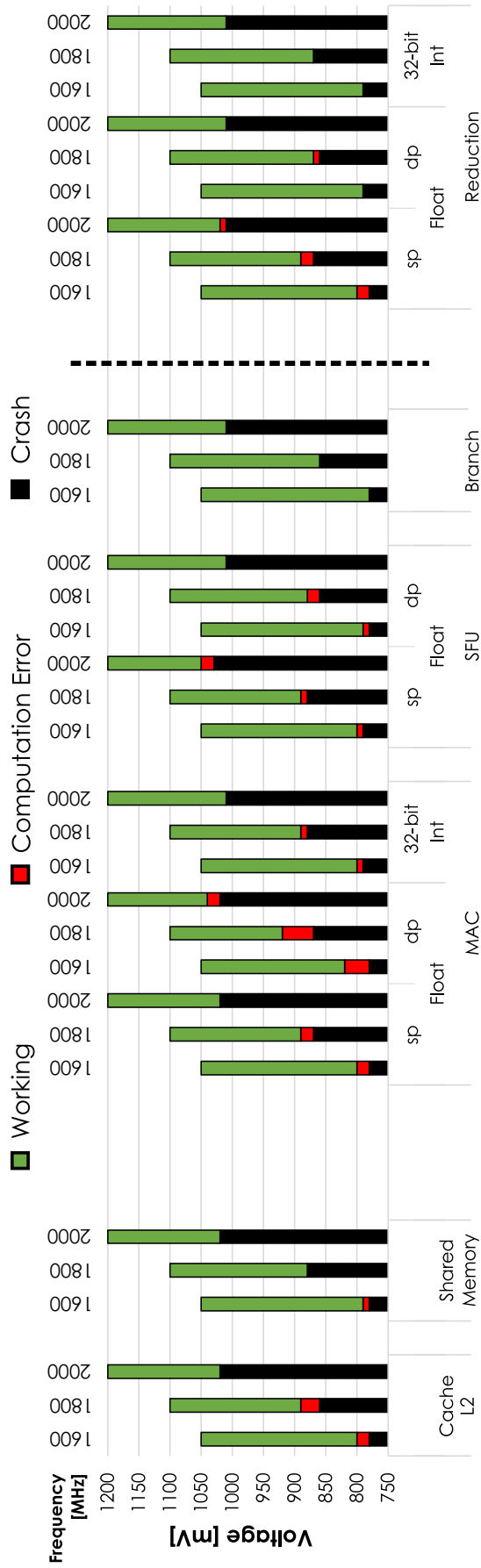
Figure 3.9 presents a comprehensive comparison of the valid voltage ranges for all the considered architectural components of both GPUs. The general observation is that the CacheL2 and the ALU are the two components that tend to compromise the undervoltage capabilities. CacheL2 affects the kernels that are more memory intensive, while the ALU limits those that are more compute-intensive, either with linear or special non-linear operations.

In more detail, the results of both benchmarks that test the ALU are similar. This allows concluding one of two things, either there are two similar critical paths (in terms of timing constraints) on both the linear and non-linear data paths, or the critical path is at the beginning of the ALU where the operands are forward to the appropriate computational unit. Although it is not possible to assess which preposition is the correct one, the second explanation tends to be more credible.

In general, even though two completely different GPU architectures are under test, the general behavior of both is similar. At the lower frequencies available, the minimum voltage can be safely applied without the existence of computation errors or crashes. The utilization of undervoltage for higher frequencies will depend on which components the application being executed stresses the most. Across all frequencies, both GPUs allow for more than 20% of safe undervoltage, being a considerable amount to be explored in the following section.



(a) Vega 10.



(b) Radeon 5700 XT.

Figure 3.9: Comparison of usable GPU core voltage ranges for all the considered architectural components of the GPU.

3.4 Effect of a decoupled V-F scaling on performance and energy consumption

As it was observed in the previous section, the GPU behavior to V-F scaling depends on the stressed components. Having explored and established the usable V-F pairs, this section presents the energy-performance charts and the EDP heat-maps for each of the tested benchmarks. In particular, this section's results indicate the maximum attainable improvement on each component and acts as a target optimization for subsequent analysis using complete applications.

For an easier understanding of the obtained results, the following charts only represent the data-points that correspond to voltages equal-to and lower-than the default voltage of each frequency level (no interesting data is found at higher voltage levels), with each data point representing a $50mV$ undervoltage in relation to the previous one. Furthermore, the presented performance, energy consumption, and energy-delay product charts are normalized to the results achieved with the highest core frequency and default voltage (Vega 10 - {1600 MHz; 1.2V} and Radeon 5700 XT - {2000 MHz; 1.2V}). As such, a smaller number indicates an improvement in relation to that configuration.

3.4.1 DRAM

Figure 3.10 illustrates the normalized performance and energy consumption of the conceived DRAM benchmark (see Listings 3.1) when varying the OPS parameter between 0 and 50 operations for the DRAM DVFS domain, normalized to the highest DRAM DVFS V-F configuration - {950MHz, 1V}.

This experiment shows that, for all OPS values (0 to 50), the highest DRAM frequency delivers not only the best performance but also the lowest energy consumption. Moreover, undervolting the DRAM at that frequency did not result in a relevant reduction in the total GPU energy consumption, leading to the conclusion that the weight of DRAM in the GPU energy consumption is not significant in relation to the *Core* energy consumption. In accordance, the highest DRAM frequency will be hence-forwardly considered, guaranteeing the maximum performance and leaving the voltage control for the automatic DVFS system.

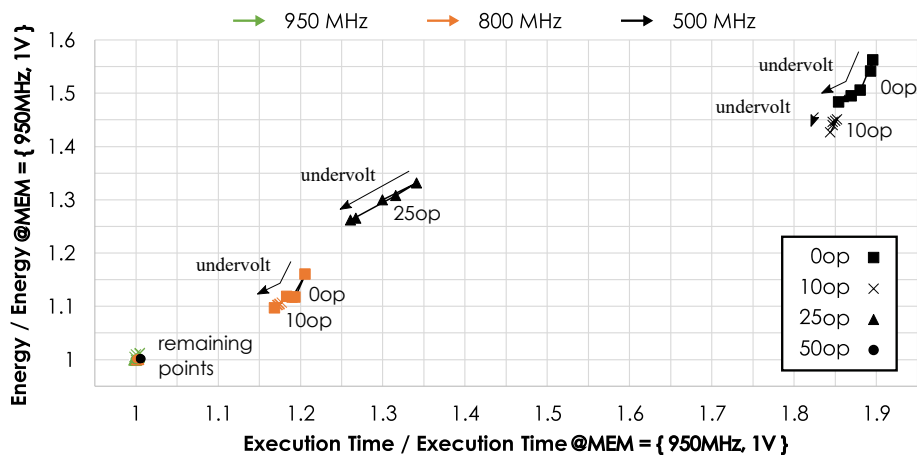


Figure 3.10: Vega 10 - DRAM domain - DRAM benchmark - Normalized energy consumption and execution time. Each connected data point represents a $50mV$ undervoltage in relation to the previous one.

A collateral experiment was also conducted to evaluate any possible cross-relation between the applied V-F setup at the Core domain when executing the DRAM benchmark with a variable amount of stress in memory bounded kernels. Figure 3.11 illustrates the normalized performance and energy consumption, with varying OPS parameter between 0 and 50 operations (see Listings 3.1). For this benchmark, it is not relevant to determine the minimum usable voltage, since that value will be defined by the type of operations being performed on the Core. The objective is to determine the configuration that offers the best energy efficiency for kernels that are memory bounded.

The experiment shows that for memory bounded kernels (OPS values 0 and 10), performing frequency scaling at the core domain does not change the computation time. However, it significantly impacts the energy consumption (going from the highest frequency to the lowest yields a 54.2% reduction on energy consumption). For the case of a kernel that is not memory bounded (OPS = 25), the benefit of exploring non-conventional V-F pairs becomes clear. However, as stated, the optimization of the V-F pairs for compute bounded kernels depends on the type of operations being performed in the ALU, analyzed in the following sections.

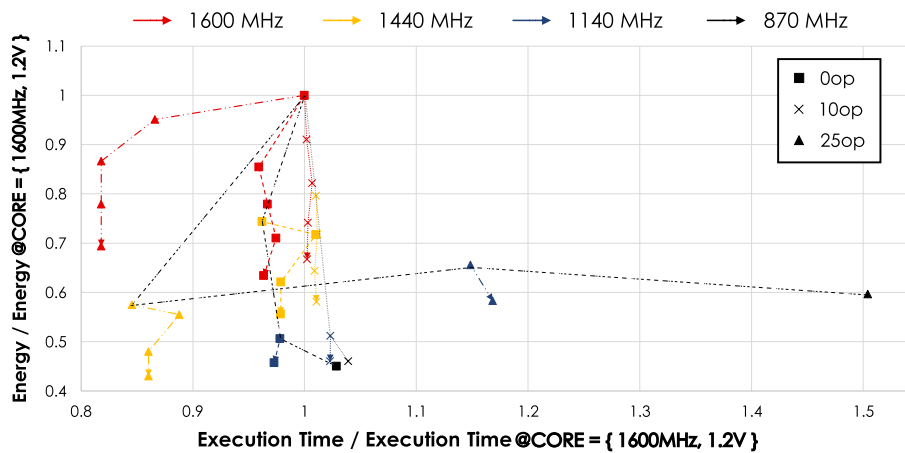


Figure 3.11: Vega 10 - Core domain - DRAM - Normalized energy consumption and execution time. The dashed line connects the default V-F configurations and each connected data point represents a 50mV undervoltage in relation to the previous one.

3.4.2 Cache and Shared Memory

Figure 3.12 and 3.14 presents the normalized energy consumption and execution time for different V-F setups for the cache and shared memory benchmarks, respectively. Acting on core domain, the normalization of the present and all subsequent benchmark is done to the highest V-F configuration available: Vega 10 - {1600 MHz; 1.2V} and Radeon 5700 XT - {2000 MHz; 1.2V}.

In what concerns the energy and performance variations, it was observed that performing the conventional voltage-frequency scaling (using the default voltage by for each frequency value - points connected by the dashed line) provides an energy consumption as high as 46% for Vega 10 GPU and 30% for Radeon 5700 XT GPU. However, this also introduces a performance degradation of 61% on both GPUs.

As it was referred before, the advantage of performing non-conventional DVFS becomes apparent by allowing for energy reduction without any performance degradation. For example, the Vega 10 running at

{1600MHz ; 1V} allows an energy reduction of 36% while still being a peak performance. In the Radeon 5700 XT case, the use of non-conventional V-F pairs yields a minimum energy consumption that is even lower than the most energy-saving default V-F configuration, with the configuration of {1800MHz ; 0.8V} achieving an energy consumption reduction of 41% with only a 10% performance downgrade. This energy consumption reduction is twice as big as the one achieved at the most energy-saving default V-F configuration. The results of the shared memory are similar.

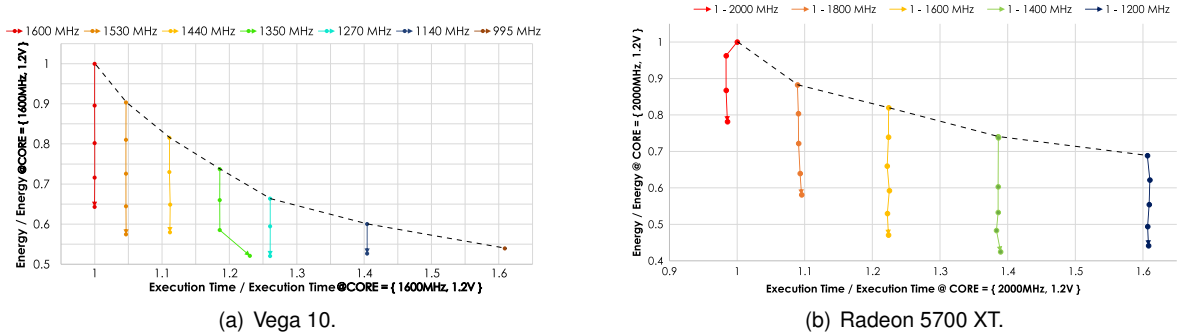


Figure 3.12: Core domain - Cache L2 - Normalized energy and performance variations with OPS=0. The dashed line connects the default F-V configurations.

Figure 3.13 and 3.15 presents the obtained EDP for the Cache and shared memory benchmarks. These components favor the utilization of minimum voltages to achieve an EDP improvement of over 40%. Comparing the non-conventional V-F results to the default ones, and taking into account the results of the previous section, using the proposed configurations improves performance and energy-consumption, and so energy efficiency without any inconvenient.

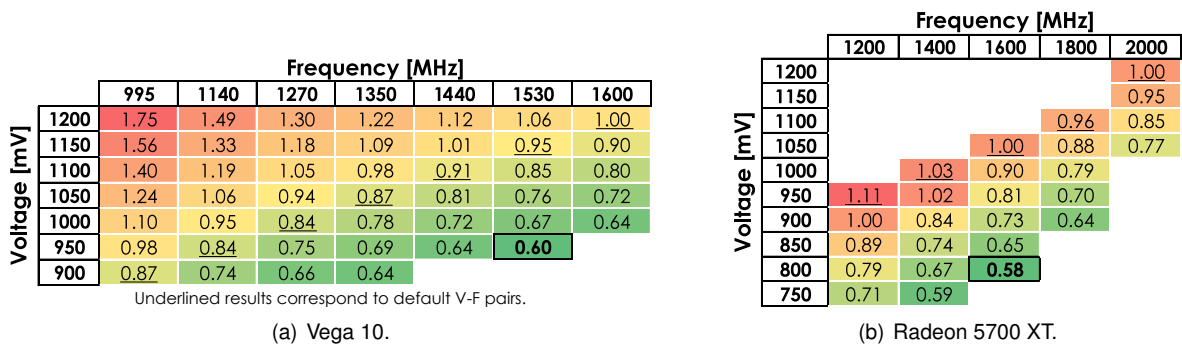
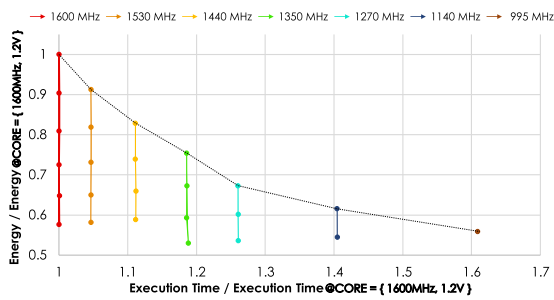


Figure 3.13: Core domain - Cache L2 - Obtained normalized Energy-Delay Product (EDP) with OPS=0.

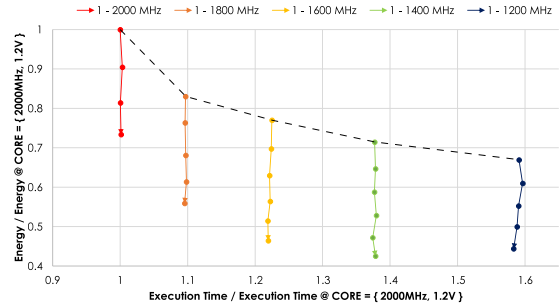
3.4.3 Arithmetic and Logic Unit

Figure 3.16 represents the normalized (to the highest core domain V-F configuration) energy consumption and execution time of the MAC benchmark for different V-F pairs. In the figures that follow, it was chosen to represent the result of the single-precision floating-point data-type, since it is the most used. However, the results for the remaining data types are similar.

An interesting phenomenon is observed in the energy-execution time plot for the highest frequencies of both GPUs. Performing undervoltage not only reduces energy consumption (as expected), but it also allows for faster execution time. An explanation can be found by analyzing the power consumption



(a) Vega 10.



(b) Radeon 5700 XT.

Figure 3.14: Core domain - Shared Memory - Normalized energy consumption and execution time. The dashed line connects the default F-V configurations and each connected data point represents a $50mV$ undervoltage in relation to the previous one.

Voltage [mV]	Frequency [MHz]						
	995	1140	1270	1350	1140	1530	1600
1200	1.81	1.53	1.34	1.24	1.14	1.07	1.00
1150	1.60	1.37	1.21	1.12	1.03	0.95	0.90
1100	1.44	1.23	1.08	1.00	0.92	0.86	0.81
1050	1.28	1.09	0.96	0.89	0.82	0.77	0.73
1000	1.14	0.97	0.85	0.80	0.73	0.68	0.65
950	1.02	0.86	0.76	0.70	0.65	0.61	0.58
900	0.90	0.77	0.68	0.63			

Underlined results correspond to default V-F pairs.

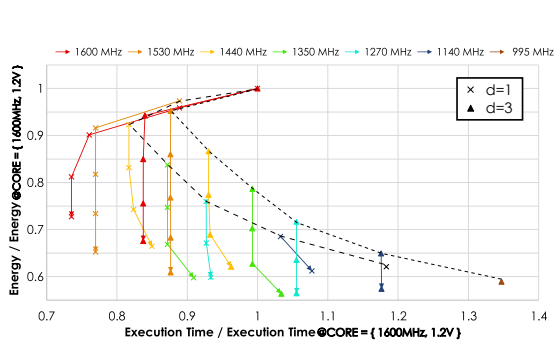
(a) Vega 10.

Voltage [mV]	Frequency [MHz]				
	1200	1400	1600	1800	2000
1200					1.00
1150					0.91
1100				0.91	0.81
1050			0.94	0.84	0.73
1000		0.98	0.85	0.75	
950	1.07	0.89	0.77	0.67	
900	0.97	0.81	0.69	0.61	
850	0.88	0.73	0.63		
800	0.79	0.65	0.57		
750	0.70	0.59			

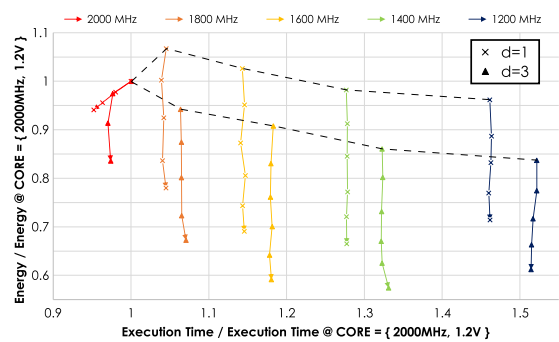
(b) Radeon 5700 XT.

Figure 3.15: Core domain - Shared Memory - Obtained normalized Energy-Delay Product (EDP) with OPS=0.

during the benchmark execution. For the default voltage, the power surpasses the power cap, which activates the GPU protection mechanisms, halting the execution until the power is reduced. By applying an undervoltage, the power significantly decreases (as $P_{Static} \propto V$ and $P_{Dynamic} \propto V^2$, see [50]) and allows a sustained maintenance of the desire DVFS configuration.



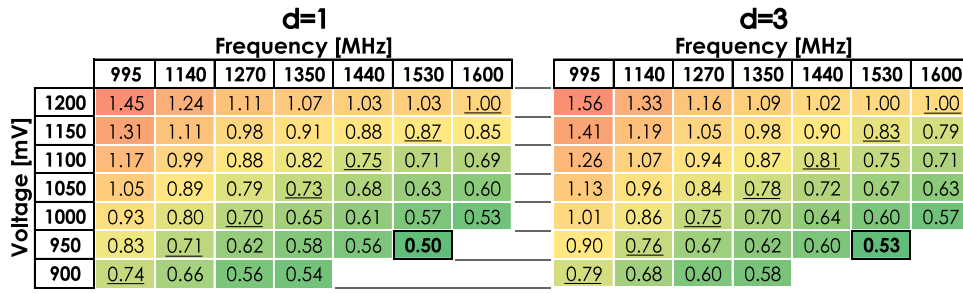
(a) Vega 10.



(b) Radeon 5700 XT.

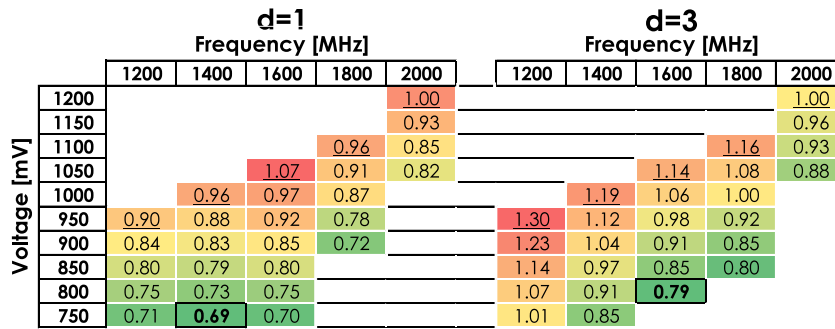
Figure 3.16: Core domain - ALU-MAC - Normalized energy and performance chart for the benchmark setup with different values of d for single precision floating-point (see Listing 3.5). The dashed lines connect the results for default F-V configurations.

Finally, Figure 3.17 presents the obtained EDP chart for single-precision floating-point data-type. The Vega 10 GPU favors higher frequencies to achieve the best energy efficiency, while the Radeon 5700 XT achieves better results at the frequencies where it is possible to undervolt the most.



Underlined results correspond to default V-F pairs.

(a) Vega 10.



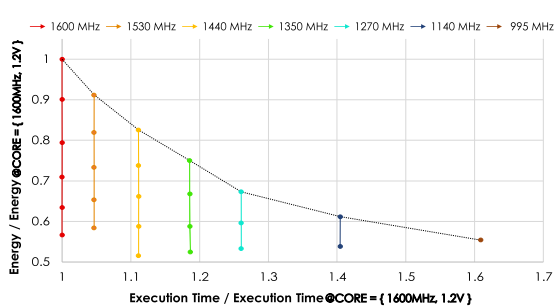
(b) Radeon 5700 XT.

Figure 3.17: Core domain - ALU-MAC - Obtained normalized Energy-Delay Product (EDP) for d=1, 3.

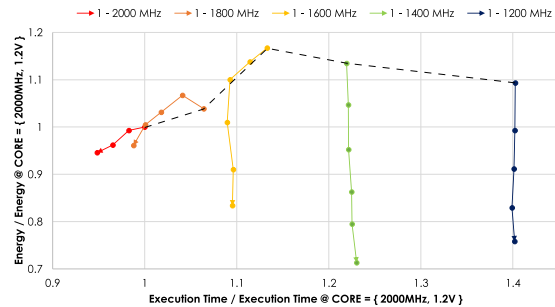
3.4.4 Non-linear Operations

Figure 3.18 showcases the normalized energy consumption and execution time for different V-F pairs for the single-precision floating-point SFU benchmark. In this case, the GPU behavior differs on the two architectures. Vega 10 displays a similar behavior, to Cache and Shared Memory benchmarks, with the applied undervoltage not affecting the performance. On the other hand, the Radeon 5700 XT behavior to non-conventional V-F is more similar to the MAC benchmark. At the two highest frequencies, performing undervoltage has a more significant benefit on performance than on energy.

Overall, the degree of energy savings that is achieved goes in line with the previous results.



(a) Vega 10.



(b) Radeon 5700 XT.

Figure 3.18: Core domain - Special Function Unit - Normalized energy and performance for single-precision floating-point data type. The dashed lines connect the results for default F-V configurations.

Figure 3.19 represents the obtained EDP heat-map, where it can be seen that the best energy efficiency is achieved with the highest amount of undervoltage, with this unit favoring lower frequencies on both GPUs.

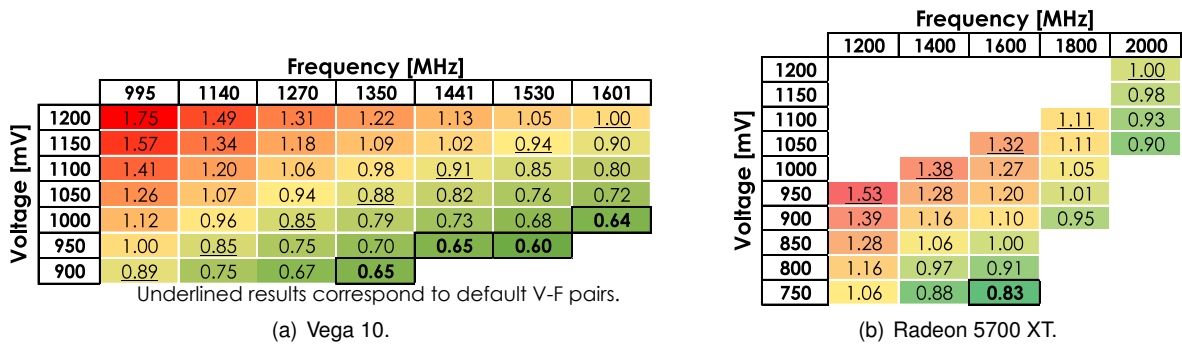


Figure 3.19: Core domain - Special Function Unit - Obtained Energy-Delay Product (EDP) for single-precision floating-point data type.

3.5 Temperature Model

The ALU and Cache L2 benchmarks were tested at a set temperature of 45°C, by fixing the GPU fan to 100% and waiting for the temperature to stabilize between runs. This temperature was chosen by observing that the GPU's temperature, while executing short benchmarks, stabilized around that temperature. However, during the execution of longer applications or when changing environmental conditions, the device can become hotter. The described work of Leng *et al.* [1] pointed out that only a small variation on V_{min} is observed due to temperature variations. However, the performed experiments only cover temperatures up to 70°C, easily surpassed by the GPU underuse.

To access the undervoltage capabilities in a broader range of temperatures, the benchmarks were continuously executed on both GPUs, by varying the GPU fan speed, and the output of the execution was analyzed. Varying the frequency did not change the temperature behavior, so the results of all executions were combined in Figure 3.20, where only the results corresponding to voltage variations were represented.

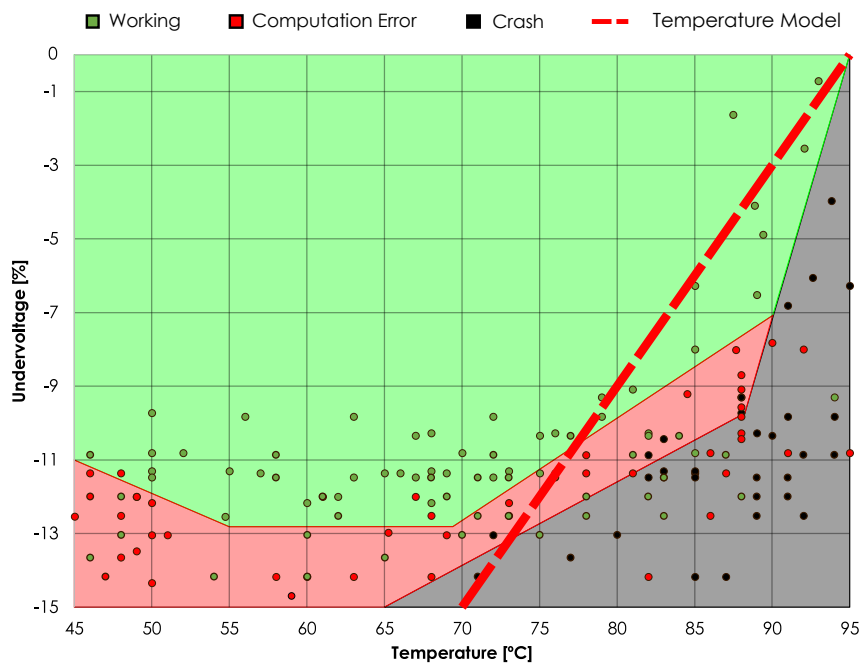


Figure 3.20: Undervoltage capabilities with changing temperature condition and defined temperature model.

Hence, changing the amount of undervolt or fan speed resulted in the three different output scenarios. *Working* - the benchmark's output was correct and was the same as when running with conventional V-F pairs. *Computation Errors* - the benchmark's output was not correct. However, the GPU was still working and responding to the kernel commands. *Crash* - the GPU stop working and responding to the commands of the application.

Overall, the undervoltage capabilities stay relatively the same until the 70°C to 75°C temperature, with the highest undervoltage capabilities being achieved at the 55°C mark. After the 75°C mark, and following Freijado's work [19], the carriers' mobility decreases and starts limiting the undervolting capabilities of the CMOS circuit. This result leads to the creation of a simple temperature model that limits the undervoltage potential for temperatures above 70°C, as indicated in Figure 3.20. This model acts as a fail-safe that guarantees that the non-conventional V-F exploration performed at 45°C (as described until now) can be safely used across the complete temperature spectrum. The final user will have to limit the percentage of undervoltage according to this model, depending on the current GPU temperature. By doing so, it guarantees that setting a safe non-conventional V-F pair will not cause a GPU crash with temperature rise.

3.6 Summary

The work presented in this chapter has three distinct objectives: 1) it acts as a feasibility assessment, allowing us to understand, measure and evaluate the degree of undervoltage that current GPU architectures allow; 2) it characterizes the optimization space that one should consider to achieve the best energy-efficiency out of the devices; and 3), it works as a target energy-efficiency optimization benchmark to measure against non-conventional V-F pairs in complete applications.

After analyzing both DVFS domains, the characterization of the Vega 10 GPU showed that the DRAM domain does not benefit from either non-conventional V-F pairs or even solely frequency scaling. AMD should have made the same conclusion since the next generation card (Radeon 5700 XT) omitted this DVFS domain.

The following chapters will focus on improving the device energy efficiency by controlling the Core DVFS domain. On this domain, it was found that both GPUs significantly benefit from non-conventional V-F pairs, by allowing an increase of the energy efficiency at lower operating frequencies while having (and even surpassing) the performance corresponding to the traditional highest frequency and voltage configurations.

Chapter 4

Decoupled V-F Optimization

Mechanism

The preliminary results that were presented in the previous chapter denote the idea that energy-efficiency (in particular, the EDP metric) of an out-of-the-shelf GPU can be significantly improved if a specific non-conventional V-F pair is applied on the Core DVFS domain.

However, the preliminary experiments that were conducted in the previous chapter, did not include any adaptation of the frequency and voltage scaling in a dynamic way, being one of this dissertation's objectives. To address this dynamic tuning of frequency and voltage, while using the increased exploration space of the operation conditions (and taking into account that it is necessary to guarantee a safe GPU operation), two approaches are now envisioned: creating a forecasting model, or creating an online optimization mechanism.

The forecasting model would predict the most appropriate V-F pair based on the application executing code (static analysis of the Assembly code) and performance counters (run-time trace of the application being executed). This option would have the benefit of allowing the complete execution of the target application under the best possible configuration. However, such a forecasting model would also have to take into consideration the GPU temperature, utilization (the target application being executed by itself or concurrently with others) and more importantly, it would be very tied to a specific GPU model. Consequently, this option would become rather complex and not easily scalable between different GPUs.

On the other hand, an online iterative optimization mechanism could target the native code repetition patterns usually observed in GPGPU applications. For these applications, the best overall configuration is the best V-F pair for each algorithm's step, so by intelligently exploring a V-F configuration in each iteration of the user application, it is possible to find the best V-F pair for the running algorithm. This approach brings the added advantage of optimizing not only the GPU pre-execution state, accounting for the aging and all PVT variations, but it also reacts to the on-execution state, changing the V-F configuration in accordance with device temperature and utilization.

Due to the added benefits of targeting the current GPU state, the second approach, consisting on the creation of an online V-F optimization mechanism, was followed. Accordingly, this chapter is divided into three sections with the following outline: Section 4.1 presents a description of the developed optimization mechanism; Section 4.2 describes the interfaces and other programs that were used to develop the

devised mechanism. Finally, Section 4.3 presents a description of the implemented functions and how they should be integrated in the user application.

4.1 Decoupled V-F Optimization Mechanism description

The envisioned V-F optimization mechanism aims to search and find the optimal V-F configuration to the running GPGPU application and current GPU state, optimizing it for *performance*, *energy consumption* or *energy-efficiency* (EDP). As it was described in the previous chapter, this optimal V-F configuration depends on the type of computations being performed, the GPU temperature, utilization, PVT variations and aging, and it is obtained by searching over an exploration space based on the set of observations that were obtained from the set of experiments covered on chapter 3.

An essential consideration that must be taken into account when designing the optimization mechanism is the time that the regular GPUs voltage-frequency controllers take to change these parameters. As described in Chapter 2, the controllers equipping out-of-the-shelf devices take between 200 to 500 ms to change and set the new V-F pair, making them unsuitable for continuously adapting to each operation executed on the device. Instead, it is necessary to group a collection of operations (as described, a complete iteration of the user application) and find the most suitable V-F configuration for that set. In particular, it is necessary to allow a portion of time between the V-F changes in order to guarantee that the control mechanism has time to perform the change and stabilize both the frequency and the voltage before continuing to execute the computation.

As it was previously referred, at the time of this dissertation, only AMD provides the necessary tools to independently control voltage and frequency, being that a requirement that is out of our control when designing this tool. However, if other manufacturers develop command-line applications that provide the same degree of control, this optimization mechanism can easily be ported to accommodate the same.

4.1.1 Architecture and Execution Overview

The devised V-F optimization mechanism follows the block diagram of Figure 4.1, consisting of a two-phase process. In the first phase, data about the GPU DVFS system is gathered and application baseline metrics are taken. In the second phase, the user application algorithm is executed while searching for the best V-F pair. When the best V-F configuration is found, the devised optimization mechanism continues to monitor the user application to guarantee a safe operation, reacting to eventual GPU state changes (for example, changing temperature).

On this context, and depending on the considered optimization metric, the best V-F configuration is the one that achieves the highest *performance*, lowest *energy consumption*, or higher *energy efficiency* for the running application. Taking into account the main objective of this dissertation, the chosen and exemplified metric from now on is *energy efficiency*, evaluated using the EDP metric as described in Equation 2.10.

The following sections describe the operation of each stage of the devised optimization algorithm.

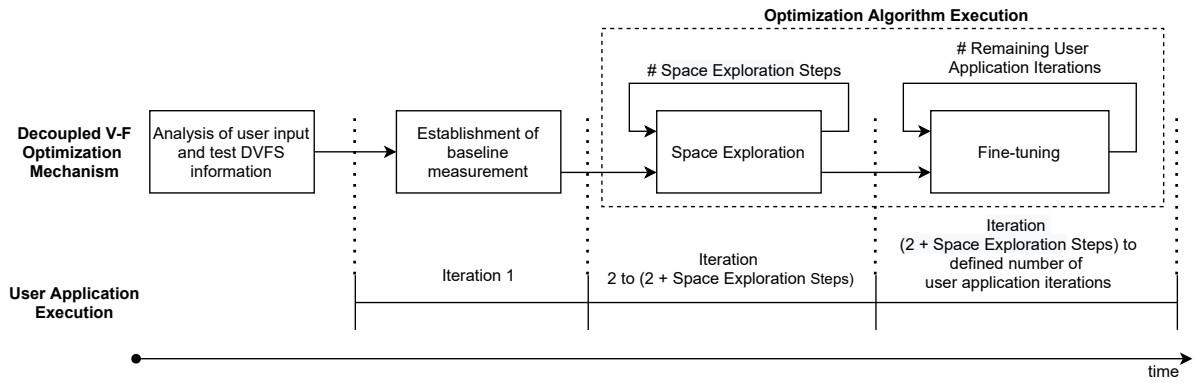


Figure 4.1: V-F Optimization Mechanism Block Diagram.

Analysis of user input and test DVFS information

The proposed procedure starts by receiving the user input that summarizes the characterization results obtained in Chapter 3. The summary includes the tested Core DVFS frequencies and allowed voltage range for each frequency that guaranteed a correct operation in all tested benchmarks. Figure 4.2 shows an example input to the mechanism and their corresponding usable execution space. There, the user should indicate the tested frequencies and corresponding V_{max} and V_{min} . After that, information about the current GPU device DVFS system is gathered and compared to the user input, guaranteeing the validity of the proposed usable execution space, which will be explored by the optimization mechanism.

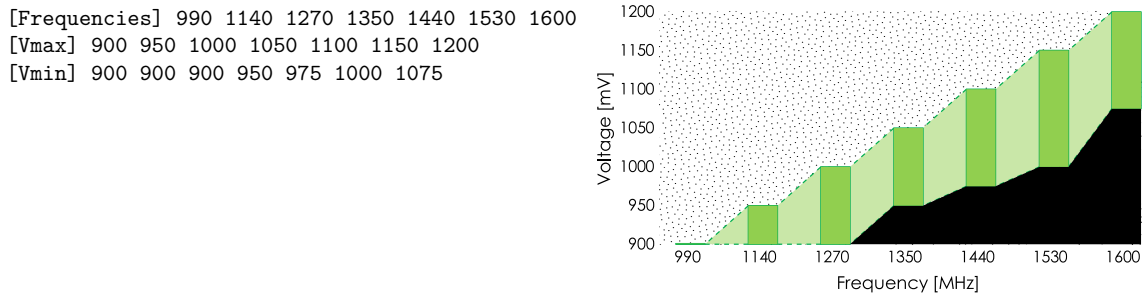


Figure 4.2: User provided input example for the optimization mechanism (left) and correspondent usable execution space chart (right). Black-shaded regions represent unusable operation points (GPU crash), while green shaded regions represent tested DVFS operating points.

Establishment of baseline measurement

In this stage, a single step of the user application is executed with the highest frequency-voltage pair that was identified on the previous step, while the execution time and energy consumption are measured to compute the baseline EDP value. All the following measurements are normalized to this first result, in order to compare the results at different V-F configurations.

Optimization Algorithm Execution

In general, each stage of the Optimization Algorithm Execution phase (*Space Exploration* and *Fine-Tuning*) follows the flowchart presented in Figure 4.3. This has three distinct steps, denoted as *Application execution and metrics*, *V-F control* and *Online monitorization*. The first and last steps are the same for the two stages of the Optimization Algorithm Execution. However, the second *V-F control* step, varies between the *Space Exploration* and *Fine-Tuning* stages, as described ahead in V-F control during Space Exploration stage and V-F control during Fine-Tuning stage.

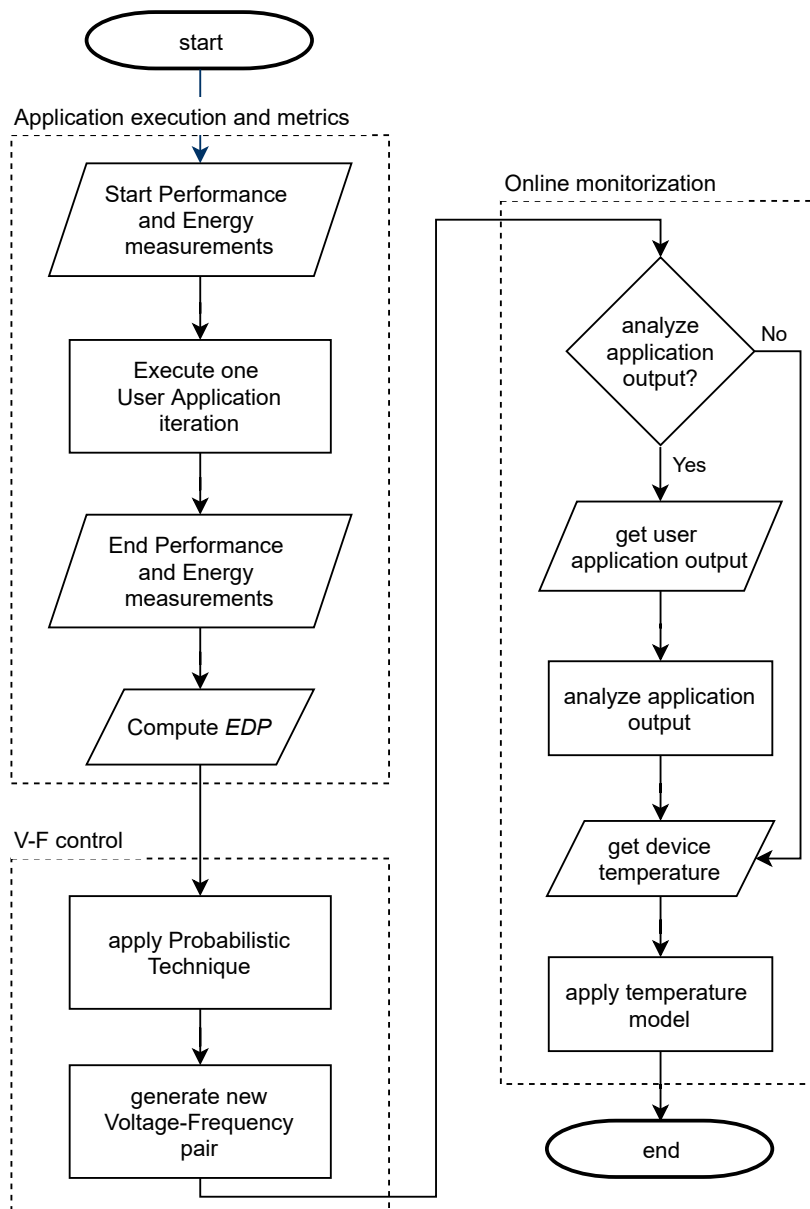


Figure 4.3: Flowchart of each iteration of the Optimization Algorithm Execution.

The following subsections describe the operation of each step of the optimization algorithm execution stage.

Application execution and metrics

On the *Application execution and metrics* step, an iteration of the user application is executed, and

the performance and energy consumption is evaluated to compute the optimization metric EDP.

V-F control

This step aims to analyze the considered optimization metric, EDP value, and, according to a probabilistic technique, accept or reject the tested V-F configuration to generate a new V-F pair.

As previously stated, the devised V-F Optimization Mechanism works by iteratively exploring the usable execution space V-F configurations, while measuring the resulting EDP value for each of them. Hence, the challenge here is to find the most suitable configuration (corresponding to the V-F pair that minimizes the target metric) without testing every possible configuration, as performed on Chapter 3. A panoply of algorithms could be employed to decide which configurations to test, while searching for the global minimum of the $EDP(f, V)$ function. The selection of such probabilistic technique considered the use of the mathematical optimization technique of Hill Climbing [51] or the metaheuristic algorithm of Simulated Annealing [52] (depending on the Optimization Algorithm Execution stage), supported on the first by improving its ability to escape local minimums, and so improving the chances of finding an approximate global optimum in a fixed amount of iterations.

In more detail, if the achieved EDP value is smaller than the current baseline after the *Application execution and metrics* stage, the current V-F configuration is stored alongside the corresponding EDP value as the current *V-F pair baseline* and it is inserted on a list of best configurations. Then, based on the current *V-F pair baseline*, the new V-F configuration is generated using the algorithms depicted ahead.

V-F control during Space Exploration stage

As the name implies, this step objective is to explore the usable execution space, finding a V-F configuration that achieves a good approximation of $\min(EDP(f, V))$ in a reduced number of iterations. To tackle this problem, the *Simulated Annealing* algorithm was applied as the adopted Probabilistic Technique (see Figure 4.3) of this stage. Since it is impossible to guarantee that $EDP(f, V)$ does not contain any local minimums (without any prior knowledge), it is preferable to use an algorithm that reduces the chance of those impacting the final chosen V-F configuration. In practice, this is reflected by the algorithm allowing for some tested V-F pairs that did not achieve the best EDP value, to be accepted and stored as *V-F pair baseline* and so, escaping from a possible local minimum. Based on the current *V-F pair baseline*, the new V-F configuration is randomly generated according to Algorithm 1 followed by Algorithm 2.

If no better configuration is found after N execution steps (or the predefined *Space Exploration Steps* have all been tested) the *Space Exploration* phase is ended, and the list of best configurations is analyzed, with the V-F configuration that achieved the best EDP acting as a baseline for the *Fine-Tuning* phase.

V-F control during Fine-Tuning stage

After the execution of the *Space Exploration* stage, a quasi-optimal configuration is found. Considering that the current *V-F pair baseline* is near the global minimum of $EDP(f, V)$, this second stage is responsible for fine-tuning the V-F pair to achieve the actual global minimum. For such purpose, it performs the *Hill Climbing* algorithm around the baseline configuration, only accepting V-F pairs that achieve

Algorithm 1 Space Exploration - Generate new frequency.

Input: *baseline_frequency*: current baseline frequency
Input: *default_frequencies*: list of available default frequencies
Output: *new_frequency*: new random generated frequency

baseline_frequency_index \leftarrow get *baseline_frequency* index on *default_frequencies* list
frequency_step \leftarrow choose random number from $\{-1, 0, 1\}$
new_frequency_index \leftarrow *baseline_frequency_index* + *frequency_step*
if *new_frequency_index* < 0 **then**
 new_frequency_index \leftarrow 0
else if *new_frequency_index* > length(*default_frequencies*) - 1 **then**
 new_frequency_index \leftarrow length(*default_frequencies*) - 1
end if
new_frequency \leftarrow *default_frequencies*[*new_frequency_index*]

Algorithm 2 Space Exploration - Generate new voltage.

Input: *baseline_voltage*: current baseline voltage
Input: *new_frequency*: new random generated frequency
Input: *UES(f)*: usable exploration space, provides the maximum and minimum voltage for a given frequency

Output: *new_voltage*: new random generated voltage

voltage_step \leftarrow choose random number from $\{-50, -25, 0, 25, 50\}$
new_voltage \leftarrow *baseline_voltage* + *voltage_step*
if *new_voltage* < *UES(new_frequency)*[*minimum*] **then**
 new_voltage \leftarrow minimum voltage of usable execution space for *new_frequency*
else if *new_voltage* > *UES(new_frequency)*[*maximum*] **then**
 new_voltage \leftarrow maximum voltage of usable execution space for *new_frequency*
end if

a better EDP value. The optimal configuration may be found between two default frequencies and at a voltage level that required to discretize the voltage range more finely. For that purpose, Algorithms 3 and 4 are used, discretizing both variables in 10 MHz and 10 mV steps.

Algorithm 3 Fine-tuning - Generate new frequency.

Input: *baseline_frequency*: current baseline frequency
Input: *default_frequency_range*: minimum and maximum available frequencies
Output: *new_frequency*: new random generated frequency

frequency_step \leftarrow choose random number from $\{-10, 0, 10\}$
new_frequency \leftarrow *baseline_frequency* + *frequency_step*
if *new_frequency* < *default_frequency_range*[*minimum*] **then**
 new_frequency \leftarrow *default_frequency_range*[*minimum*]
else if *new_frequency* > *default_frequency_range*[*maximum*] **then**
 new_frequency \leftarrow *default_frequency_range*[*maximum*]
end if

Online monitorization

The *Online monitorization* step introduces two other inputs to the optimization mechanism: the device temperature and (optionally) the application output. As it was referred to in Chapter 3, these two new metrics impact the amount of undervoltage that is allowed, by changing the voltage defined by the *V-F control* phase.

In the event that a representative variable can be obtained at the application output that identifies its validity (for example, a floating-point number which is tested to see if it deviates from the correct

Algorithm 4 Fine-tuning - Generate new voltage.

Input: *baseline_voltage*: current baseline voltage
Input: *new_frequency*: new random generated frequency
Input: *UES(f)*: usable exploration space, provides the maximum and minimum voltage for a given frequency
Output: *new_voltage*: new random generated voltage

```
voltage_step ← choose random number from {−10, 0, 10}  
new_voltage ← baseline_voltage + voltage_step  
pair_default_frequencies ← compute pair of frequencies around new_frequency  
minimum_voltage(f) ← compute a linear interpolation between  
    UES(pair_default_frequencies[inferior])[minimum] and  
    UES(pair_default_frequencies[inferior])[minimum]  
maximum_voltage(f) ← compute a linear interpolation between  
    UES(pair_default_frequencies[inferior])[maximum] and  
    UES(pair_default_frequencies[inferior])[maximum]  
if new_voltage < minimum_voltage(new_frequency) then  
    new_voltage ← minimum_voltage(new_frequency)  
else if new_voltage > maximum_voltage(new_frequency) then  
    new_voltage ← maximum_voltage(new_frequency)  
end if
```

value or becomes Not a Number - NaN), it is possible to include such analysis metric to be executed in each iteration of the user application, or at every x number of iterations. In this situation, where it is possible to validate the application output, the new V-F configuration is provided to that procedure (*Analyze Application Output*), which analyzes the user application iteration output and concludes about its validity. If the output is evaluated as invalid, this procedure increases the voltage by 10mV (decreasing the amount of undervoltage). Finally, the chosen frequency is given to the *Temperature Model*, depicted in Section 3.5, which reduces the amount of undervoltage when the device temperature surpasses the 70°C. At the end of these two procedures, the new V-F pair is generated and applied on the GPU in order to proceed with the algorithm execution.

It should be noted that the analysis of the application output acts as a fail-safe, and it is not mandatory in the execution of the optimization mechanism since the correct use of the methodology introduced in Chapter 3 already guarantees the correct GPU voltage operation limits.

4.2 Optimization Mechanism Implementation

The devised V-F optimization mechanism was developed in Python and acts as a wrapper using a set of functions that allow the user to implement and integrate the mechanism around its application. The decision for this programming language was purely out of convenience for the language used by the deep learning tested and optimized in Chapter 5. Ideally, the same procedures may and should be implemented on the target application's programming language to improve and facilitate the communication of the measurements between the different blocks that implement the optimization mechanism.

Figure 4.4 presents a layer diagram of the developed optimization mechanism, illustrating the main APIs that were used by the tool to communicate with the GPU device. The blocks coloured in blue represent the developed parts of the optimization mechanism developed in the context of this dissertation. These act in conjunction to control the voltage-frequency pair that is applied on the DVFS Core domain, according to the user application output and device target energy consumption and performance profile.

Following the description provided in Annex A, the V-F Controller block controls the *rocm-smi* tool to select the desired V-F pair. The Optimization Algorithm block implements the previous section’s functionality, which, according to the current performance and energy measurements, selects the V-F pair to be applied. The Python Wrapper controls the different blocks and the user application. Section 4.3 provides the definition and functionality of each of the functions provided through the wrapper. The blocks colored in orange represent the two software provided by AMD that allows for user control of the GPU and DVFS system. The first is the ROCK kernel, that is the first layer of software control and talks directly with the GPU hardware, and the second is *rocm-smi* (software previously presented in Chapter 2) that allows control over the DVFS system and retrieve current GPU power consumption and utilization. Finally, the block represented in green, *gpowerSAMPLER* is open-source software that utilizes either AMD and NVIDIA kernel API to compute GPU power and energy consumption.

The user invokes the appropriate developed functions (more details provided on Section 4.3) that communicate with the *gpowerSAMPLER* (energy measuring) tool, *rocm-smi* and *ROCK* (voltage and frequency control interface) APIs to retrieve and control the related parameters to the GPU device.

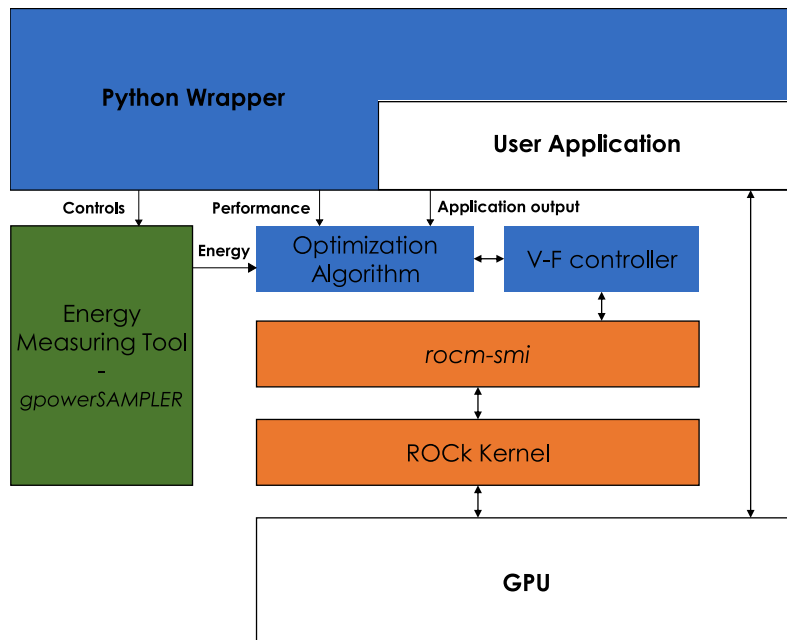


Figure 4.4: Layer diagram of the developed V-F Optimization Mechanism. The blocks colored in blue represent the developed parts of the optimization mechanism developed in the context of this dissertation.

4.3 Library description

This section describes how users should include the V-F optimization mechanism on their GPGPU applications. As indicated, the wrapper that provides all the functions to enable this mechanism was developed in Python. However, the description of the functions provided ahead should allow for porting the same for others programming languages.

def initOptimizationMechanism(pathToCharacterizationResults):

Analyzes the user-provided Usable Execution Space file and queries the device DVFS control inter-

face (rocm-smi on the current implementation) to guarantee that the indicated frequency and voltage ranges are allowed by the target device.

def initMeasurements():

Initiates the execution time and energy consumption measurement (gpowerSAMPLER on the current implementation).

def computeMeasurements(baseline=None):

Indicates to end of the execution time and energy consumption measurement (gpowerSAMPLER on the current implementation) and returns both results. If the argument baseline, containing the baseline performance and energy measurement values, is provided, the output is normalized to each baseline value.

def analyseOutput(output):

A method that should be implemented by the developer that analyzes the user application output to determine its validity. The function returns a boolean indicating the ser application output validity.

def optimizationMechanismSpaceExploration(measurements, optimizationMetric="EDP", outputValid=None):

Analyzes application measurements and apply the new V-F configuration. The function receives the performance and energy measurements, the optimization metric and the results of the application output validity and implements the Simulated Annealing algorithm to accept or reject the tested V-F pair in accordance with the current *V-F pair baseline*. If the configuration is accepted, the current *V-F pair baseline* becomes the tested V-F pair. The method returns if the tested pair was accepted or not. It also implements the Online Monitorization phase.

def optimizationMechanismFinetuning(measurements, optimizationMetric="EDP", outputValid=None):

Similar to the previous function, however it implements the Hill-Climbing algorithm as Probabilistic Technique.

Listing 4.1 provides an example of the order and place where each function call should be performed and where the user should insert its application code.

```
initOptimizationMechanism(pathToCharacterizationResults)
... initiate user application ...
initMeasurements()
... execute baseline execution ...
baselineMeasurements = computeMeasurements()
rejected = 0
executedIterations = 0
for epoch in range(space_exploration_epochs):
    initMeasurements()
    ... code of application step ...
    measurements = computeMeasurements(baselineMeasurements)
    ... other necessary user application iteration operations that do not account
    for the user algorithm execution ...
    outputValid = analyseOutput(output)

    # analyses the measurements, application output, temperature
    # generates and applies new V-F configuration
    accepted = optimizationMechanismSpaceExploration(measurements, outputValid)
    executedIterations += 1
```

```

    if accepted == False:
        rejected += 1
    if rejected == 5:
        break

for epoch in range(total_number_of_epochs - executedIterations):
    initMeasurements()
    ... code of application step ...
    measurements = computeMeasurements(baselineMeasurements)
    ... other necessary user application iteration operations that do not account
    for the user algorithm execution ...
    outputValid = analyseOutput(output)

    # analyses the measurements, application output, temperature
    # generates and applies new V-F configuration
    optimizationMechanismFinetuning(measurements, outputValid)

```

Listing 4.1: Usage example of the V-F Optimization Mechanism Library. Blue statements represents the added programming elements for the mechanism

4.4 Summary

The development of the V-F Optimization Mechanism that was described in this chapter fulfills the objective of having a concrete application to enable a non-conventional DVFS system. The devised mechanism uses the previous chapter's experimental results to provide the user with convenient models that support the uncover V-F pairs to extract better energy-efficiency of their devices, without having any detailed knowledge of the GPU architecture.

The following chapter demonstrates and evaluates the application of both the characterization and optimization mechanisms to improve the energy-efficiency of the Vega 10 GPU when running deep learning applications.

Chapter 5

Application to Deep Learning

Nowadays, Deep Learning algorithms are the most common applications being executed on GPUs. In addition to it, from the set of applications that are known to be imprecision tolerant, deep learning algorithms stand out as one of the most prominent ones. These two characteristics make Deep Learning the perfect candidate to be the target application to apply the current findings of the previous chapter. This chapter starts by providing a brief overview about deep learning applications, presenting the concept of deep neural networks (DNN), how the training of these algorithms is performed and how they are implemented using high-level libraries.

The chapter continues by applying non-conventional V-F scaling to the most energy consumption layers of a Convolutional Neural Network (CNN) [53]. Finally, the devised V-F Optimization Mechanism, described in Chapter 4, is applied to the training process of a CNN, allowing the reduction of the consumed energy and massive a improvement on energy efficiency of GPUs running this kind of algorithm.

5.1 Deep Learning Overview

In the last few years, Deep Learning (DL), and more particularly Deep Neural Networks (DNN), have had a significant impact in industry and society, by allowing for important breakthroughs in many application domains, such as computer vision, speech recognition, natural language processing, drug discovery, genomics, and others [2].

Deep Learning is a subset of the larger family of Machine Learning methods, also known as deep structured learning or hierarchical learning. This type of algorithms can be utilized for supervised, semi-supervised, and unsupervised learning [54, 55]. Different Deep Learning architectures are being developed, such as convolutional neural networks (CNN), recurrent neural networks (RNN), and unsupervised pre-trained networks (UPT), targeting different objectives and being able to analyze and learn from the data in different ways. The general trend over the years is to increase the number of trainable parameters, usually denoted as weights of the network, to achieve better results. Such an increase in the tunable parameters not only demands an improvement in the device's memory - to accommodate the increased size of the model; in the device's performance - to be able to train and use the model in usable time; and more importantly, energy efficiency - to allow a sustained increase of the number of devices in supercomputers and to be able to run the algorithms in portable computing devices.

5.1.1 Deep Neural Networks

The central element of a deep neural network (DNN) is the artificial neuron. This element can be mathematically modeled by a set of multiplications and summations, as shown in Equation 5.1, where W_i represents the weight and b is the bias applied on each artificial neuron.

$$Y = \sum_{i=1}^n W_i X_i + b \quad (5.1)$$

One of the most significant benefits of deep neural networks is their ability to capture non-linear relationships between the input parameters. For such purpose, an activation function is usually attached to each neuron, helping him to handle scenarios where problems are not linearly separated [56]. The most common non-linear activation functions are hyperbolic tangent (*tanh*) [57], Rectified Linear Unit (ReLU) [57] and sigmoid [57]. Hence, the prevailing deep neural network architecture is the combination of the linear transformations performed by the neurons plus the non-linear activation functions.

A neural network is an arrangement of neurons and activation functions in several layers. Each layer is responsible for applying a series of transformations to the data according to the weight and bias stored in each artificial neuron. As represented in Figure 5.1, a DNN is composed of at least three layers, the input layer, with a number of neurons equal to the input size, at least one hidden layer, being this the distinguishing characteristics of a DNN, and an output layer with the number of neurons equal to the output size. The possible different organization of the layers in size, type of operation and number of connections of layers defines the type and architecture of the DNN.

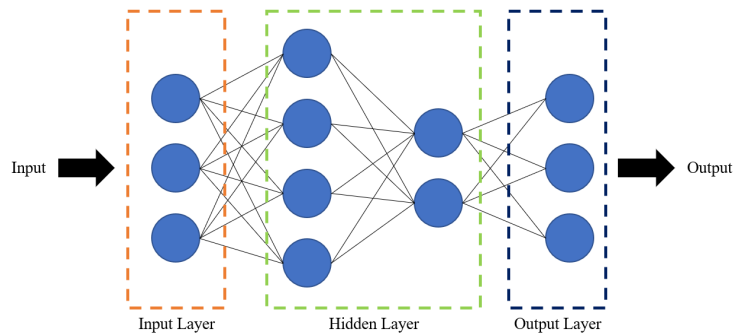


Figure 5.1: Model of fully-connected (feed-forward) DNN

5.1.2 DNN Architectures

Current DNN architectures are grouped into three types of architecture, depending on the fundamental primitive operation being performed. These are Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN) and Unsupervised Pretrained Networks (UPT).

Convolutional Neural Networks (CNN)

Convolutional Neural Networks are usually used to extract features from data via the convolution operation, being mainly used for image and object recognition and sound analysis. This type of architecture excels when there is some structure in the input data, that is, the data contains sets of specific patterns, organized in a spatial manner, that the neural network can learn to recognize.

The CNN architecture generally follows the following pattern: input layer, feature-extraction layers and classification layer. The input layer receives a form of three-dimensional data, usually an image with a specific height and width and a depth value (representing color or intensity). The feature extraction layers perform higher-order features extract from the input, generally by performing patterns of convolution layers and pooling layers. Finally, the classification layer is a vector of size N , where each output represents a score of prediction confidence for the input to be of a given output class.

Three common datasets that are often used to compare and analyse the CNN performance are MNIST [58], CIFAR 10 [59] and ImageNet [60], each with increased input complexity, number of output classes and overall dataset size. MNIST consists of 70 000 images of handwritten digits 0 to 9, CIFAR10 consists of 60 000 organized images of 10 object and animal classes, and ImageNet is a collection of 14 million images of 20 thousand different classes.

Recurrent Neural Networks (RNN)

Recurrent Neural Networks have the added capability of sending information over time-steps. This characteristic allows this type of architecture to have parallel and sequential data modeling, not only recognizing features from each input but also allowing for the extraction of features from the sequences of inputs, modeling the time dimension.

RNNs are often used to model time-series, language, audio, and text since this type of data is inherently ordered and context-sensitive. RNNs contain feedback loops between the layers, allowing each layer to have insights about what happened before. The prediction model follows the general format presented in Equation 5.2, where the current timestamp model output $y^{(t)}$ is a function f of the previous models output $y^{(t-1)}$ and current model input $x^{(t)}$, in addition to a bias factor θ . The equation reflects the influence of previous inputs for each output.

$$y^{(t)} = f(y^{(t-1)}, x^{(t)}, \theta) \quad (5.2)$$

The component responsible for the feature extraction characteristic of RNN models is the LSTM - Long Short Term Memory. This type of layer has three gates - input, output and forget gates. The content of each LSTM is mainly defined by the input and forget gates; any change on these elements reflects on the value stored at the memory cell. If both gates are closed, the memory content remains unmodified between the current time-step and the following. Hence, the LSTM structure allows for information to be retained/forgotten on the memory cell across different time-steps.

Unsupervised Pre-trained Networks (UPN)

Unsupervised Pretrained Networks is a category of DNN architectures that encompasses network structures such as autoencoders and generative adversarial networks (GANs). This class of DNN architecture contrasts with the previous ones by learning in an unsupervised manner, meaning that the input data given to the network is not previously labeled. This introduces an extra degree of learning freedom by allowing the network to identify and recognize patterns that distinguish the output classes the most.

Autoencoders are used to efficiently learn data codings and can be applied for dimensionality reduction or augmentation. Autoencoders provide a reduction of noise present in a signal or to perform a

resolution increase on an image. GANs can be used to perform sound and video synthesis from images or text using two neural networks in parallel - a discriminator and a generative network. First, the generative network creates a synthesized output from the given input. Then, the discriminator network tries to classify the input as real or synthesized, providing the classification to the generative network. With this data loop, the generative network updates their weights to fit best what is described as real data.

5.1.3 Training and Inference

The mathematical description of the DNN training process, represented in Equation 5.3, is equivalent to treating the network as a loss function L , where inputs X , outputs Y and the network's weights W and bias b are function arguments. The training session's objective is to optimize the in-network parameters W (weights) and b (bias) to minimize the overall loss [61].

$$(W, b) = \arg \min_W L(X, Y, W, b) \quad (5.3)$$

The DNN training is an iterative process, where at the end of each iteration, a loss value is computed, and the set of in-network parameters is updated. This loss value represents how well are the input parameters being modeled by the network. As the number of training iterations increases, the loss value reduces and converges to a minimum, at which the model prediction accuracy will be at its maximum. At this point, the training session can be stopped.

The most common method for the parameters update is the Stochastic Gradient Descent (SGD) algorithm (see Equation 5.4), an iterative algorithm that, after processing mini-batches of the training data, computes new weights and bias for each neuron [62].

$$W_{i+1} \leftarrow W_i - \alpha \sum_{n=1}^m \frac{\partial L}{\partial W_i} \quad (5.4)$$

In Equation 5.4, m represents the number of mini-batches to run, W_i is the current parameter, W_{i+1} is the update parameter, α is the learning rate and $\frac{\partial L}{\partial W_i}$ is the partial derivative of the loss function L in order to the parameters. This last equation is obtained by applying the derivative chain rule in a backward-cascade fashion with respect to inputs, outputs, and parameters of each DNN layer.

Each iteration of the training process is composed of a forward and backward data propagation. On the forward propagation, the loss function for the current in-network parameters is evaluated, computing the loss value. By performing the backward propagation, the partial derivative $\frac{\partial L}{\partial W_i}$ of each of the parameters is obtained to apply the DNN algorithm.

Hence, the inference (or prediction) process corresponds to the execution of a forward propagation, with the intended input, on a previously trained neural network. At the output layer, a set of values (or probabilities) is computed, corresponding to the model's prediction to the given inputs.

5.1.4 High-Level Libraries and Software Frameworks

The popularization of GPUs as the defacto DNNs execution device results in the availability of high-level libraries and frameworks from device manufacturers and other software houses. The available GPU libraries implement the underlying mathematical operations performed during the DNNs, while the

frameworks operate on the execution of DNNs by masking the complexity of creating, training and using these models [63].

The most used libraries are cuBLAS¹ and cuDNN², by NVIDIA and rocBLAS³ and MIOpen⁴, by AMD, which implements the most optimized versions of matrix multiplication, convolution, and other mathematical operations to be used on the DNN models. In what concerns the frameworks, TensorFlow⁵ and PyTorch⁶ are open-source frameworks developed by Google and Facebook, respectively, that allow an easy implementation of the models in GPUs.

5.2 DNN Performance and Energy Efficiency Improvement

DNNs are usually characterized by significant computational burdens, particularly when considering the training of very deep and complex networks that deal with high dimensional data, such as images and videos. For such purpose, researchers (and data scientists, in general) often rely on accelerators, such as GPUs, to cope with the associated computational burden and reduce the training time. As a result, GPUs are now commonly deployed on most supercomputers, data centers and other computational infrastructures related to the development of artificial intelligence algorithms.

Additionally, several software frameworks, algorithms and techniques have been proposed to manage and optimize the execution of DNNs on GPUs (e.g., Mittal [64]). However, most optimization techniques neglect the training phase's energy impact, usually resulting in considerable costs.

To overcome this problem, researchers have also explored other solutions that allow mitigating the energy impact of neural network training. One particular and common approach relies on the use of low-precision arithmetic (e.g., Nabavinejad [65]), eventually trading network accuracy with increased processing performance and lower energy consumption.

Researchers have also looked at alternative approaches, such as exploiting DVFS on both the inference and training phases. In fact, by carefully selecting the used voltage-frequency (V-F) levels, significant energy savings can be obtained, although depending on the considered DNN architecture and computing device [4]. This is achieved by a careful balance between the performance and power consumption of the different GPU components (particularly the core and global memory) to minimize the stalls in the compute cores. In fact, not only can DVFS be used to decrease the power consumption, but it can also boost the system performance [4], by increasing the voltage and frequency levels (as long as the GPU total power envelope and thermal limits are not surpassed).

Hence, supported on the performed GPU characterization to non-conventional V-F pairs, this work continues by exploring the impact of these configurations on CNN layers, considering both the training and inference phases.

¹developer.nvidia.com/cublas

²developer.nvidia.com/cudnn

³github.com/ROCmSoftwarePlatform/rocBLAS

⁴github.com/ROCmSoftwarePlatform/MIOpen

⁵tensorflow.org/

⁶pytorch.org/

5.3 Non-conventional V-F on CNNs

As it was referred in Section 2.4, Tang [4] has recently studied the impact of frequency scaling on the performance and energy consumption of DNNs executed in GPUs. By extending this study with the capability to also apply undervoltage techniques, a broader range of DVFS configurations are herein envisaged to provide even greater benefits.

Another important characteristic of DNNs is their tolerance to a certain degree of computation errors [3], without any significant change in the training and inference results. Consequently, it is important to complement the characterization that was performed in Chapter 3 with the voltage scaling effects in DNN training and inference phases and, in particular, with its influence on the computation errors that might occur when exploring the existing voltage margins.

As discussed before, at the particular case of the CNN, its feature extraction ability is mostly supported by the convolution operator. Nonetheless, CNN also includes other types of layers, such as fully connected and pooling layers. However, the convolution and fully connected layers take up to 97% of the GPU energy consumption [53], which makes them particularly suited to exploit energy-saving mechanisms.

Hence, to apply and assess non-conventional V-F pairs on the execution of CNNs, the high-level deep learning framework PyTorch and the default mathematical libraries (rocBLAS and MIOpen) provided by the considered GPU manufacturer (AMD) were extensively evaluated on the Vega 10 GPU. This section reports the main achieved conclusions for the convolution operator and fully connected layers.

5.3.1 Convolution Layer

The AMD MIOpen library provides multiple convolution implementations, being the *Direct*, *GEMM* and *Winograd* [66] the ones that are more often used. At the beginning of the execution, this library performs one convolution operation with each of the algorithms that are able to solve the required operation. Then, the algorithm that takes the shortest execution time is chosen and used to perform the remaining convolutions of the current layer. The convolution layer is defined by the parameters presented in Table 5.1.

Parameter	Description
W, H	Input Width, Height
N	Mini-batch Size
C, K	Number of features, kernels
R, S	Kernel Width, Height
Pad_W, Pad_H	Padding Width, Height
Str_W, Str_H	Stride Width, Height

Table 5.1: Convolution Parameters.

At this respect, a relevant question that may be raised is about the existence of a convolution algorithm that is more energy-efficient than the others. Such question is of particular interest for this work. If such possibility is there, it can be exploited by using one of the input parameters of the convolution API that allows the algorithm selection, disabling the automatic algorithm selection procedure. To answer to such question, the execution time, and energy and power consumption of all algorithms were measured

while executing 100 different convolutions layer configurations from the DeepBench benchmark⁷, with each algorithm being executed ten times with the median results being taken. The obtained results showed that the algorithm that achieved the shortest execution time in all cases also achieved the best energy consumption. By looking at the power consumption across the different algorithms, the result showed that it was similar in all cases. Two other important results were that no algorithm proved to be the fastest for all or for any subset of configurations and that not all convolution layer configurations are able to be solved by all the different algorithms available in the library. As previously explained, the library executes one convolution using each available algorithm to determine which are able to solve it to compare the execution time between the different alternatives, choosing the one with the shorter execution time. Even though this technique appears simplistic and with space to be further optimized, the results show that, in reality, that is not the case. Measuring the execution time at the beginning of the execution for all possible execution alternatives optimizes both the performance as well as energy consumption since power consumption is similar between all the different convolution algorithms. This technique also brings the benefit of optimizing the execution to the current GPU state and utilization, which is one of the premisses of the presented work.

To conduct this convolution analysis, a set of 20 distinct convolution layer configurations was selected from the DeepBench benchmark to understand how each algorithm is affected by the V-F configuration, both in its inference and training phases.

Layer Guardband and Characterization

Figure 5.2 presents the set of valid voltage ranges that were obtained for the inference and training phases of the convolution layer. By comparing these results with those obtained in the individual component characterization, it can be observed that some computation errors (and even some GPU crashes) were detected at lower frequencies. In fact, since this operation is more complex and requires the utilization of multiple architectural components, the undervoltage limit is more likely to be violated by the voltage drops induced by the activation and deactivation of the GPU architectural components [16]. This phenomenon will make certain parts of the GPU not to work properly (even momentarily), producing an increased rate of computation errors and an increased crash threshold voltage.

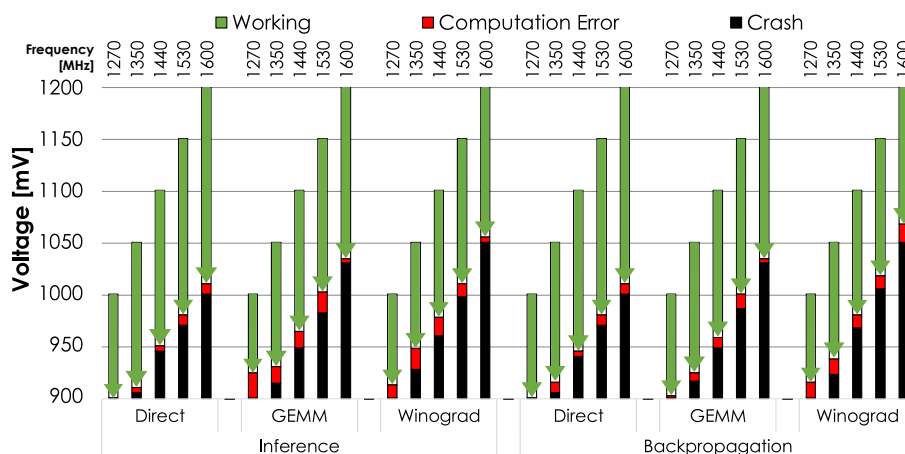


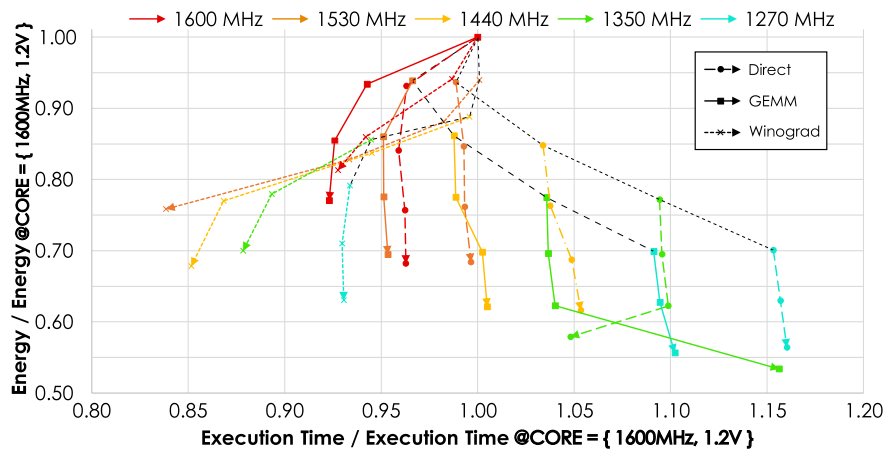
Figure 5.2: Core Domain - Convolution Layer - Usable voltage range.

⁷github.com/baidu-research/DeepBench

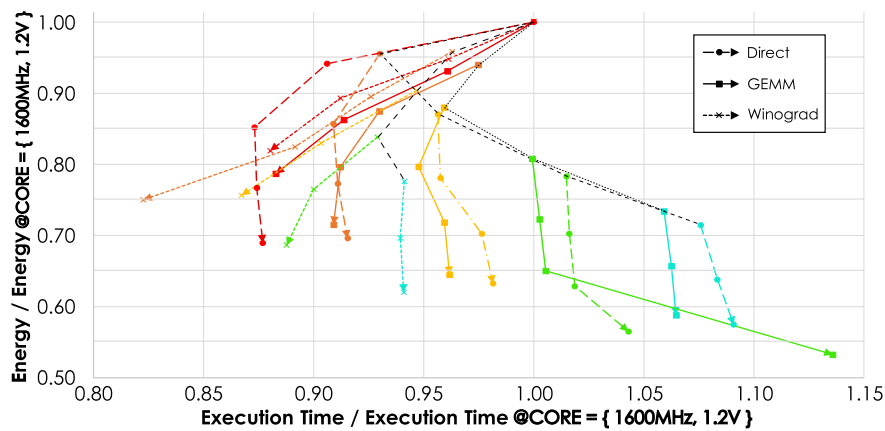
When comparing the three convolution algorithms, it is observed that the *Direct* algorithm allows for the greatest amount of undervoltage, followed by *GEMM* and *Winograd*. The *Direct* algorithm is the simpler of the three, with no data transformation and movement being necessary for its execution. In contrast, *GEMM* and *Winograd* need some data pre-processing before the convolution is performed. This extra step relies on the activation of more GPU components, making these algorithms more prone to induce voltage drops.

When comparing the training and inference phases, it is observed that they present similar undervoltage capabilities (for all algorithms), with the crash point diverging only around 10mV. However, the training algorithm is more prone to the introduction of computation errors, starting to be observed at a lower degree of undervoltage when compared to the inference.

Figures 5.3 and 5.4 illustrates the impact of non-conventional V-F on energy consumption, execution time and EDP. When working with the default voltage level of each frequency (dashed lines), the *Direct* and *GEMM* algorithms exhibit a valley in their performance chart, with the frequency of 1530 MHz providing the best performance. In contrast, the *Winograd* achieves its best performance with the lowest frequencies. Upon the introduction of independent voltage scaling, it is possible to improve the execution time and energy consumption (in comparison with the default voltage) by 8% and 23%, 19% and 8%, and 14% and 24% for the *Direct*, *GEMM* and *Winograd* algorithms, respectively.



(a) Inference



(b) Training

Figure 5.3: Core domain - Convolution Layer normalized energy and performance chart to non-conventional V-F configurations.

The EDP charts, depicted in Figure 5.4, indicate that the most energy-efficient configuration for the three algorithms is at the lowest frequencies and maximum undervoltage possible. At these configurations, although the execution time is reduced by 16% (for the *Direct* and *GEMM* algorithms), it is still possible to achieve a reduction in energy consumption of up to 46%. The use of the most efficient configuration for the *Winograd* algorithm improves both the execution time and the energy by 15% and 32%, respectively.

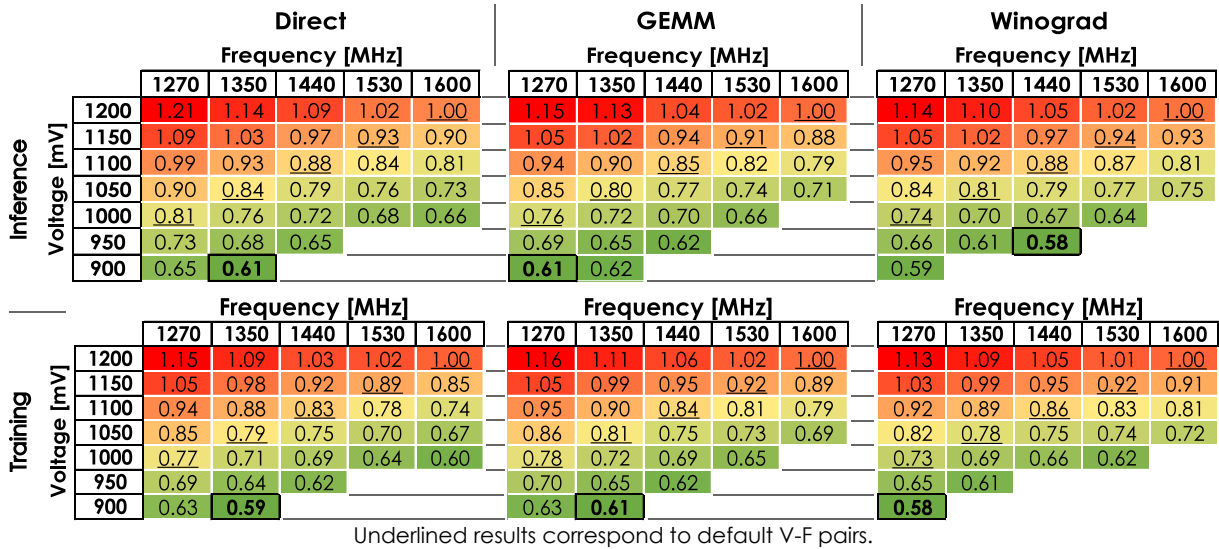


Figure 5.4: Core Domain - Convolution Layer - Obtained normalized Energy-Delay Product heat-map for the three algorithms.

5.3.2 Fully-Connected Layer

The RocBlas library provides a single API for matrix multiplication, the underlying mathematical operation of the fully connected layer. By analyzing the performance counters and the kernels invoked by the library, it is possible to understand that multiplication is performed in one of two ways, depending on the size of the matrices. According to the work of Nabavinejad [67], small matrices are first loaded to Cache, and all the operations are performed with the data in this memory component, making the operation compute bounded. For large matrices, the matrix multiplication is split, performing the multiplication in a tiling fashion. In this way, submatrices are first loaded to the local caches, and the corresponding submatrices of the results are produced. When the submatrix is consumed, another pair is loaded, and the process repeats until the full computation is performed. The threshold size of the submatrices corresponds to the size of the L1 Cache.

Layer Characterization

Non-conventional V-F impact the two implementations of the matrix multiplication in different ways. For the small matrices variant, since all the necessary data is already available on the local caches before the computations start, it is the ALU that will limit the undervoltage. Consequently, it is expected that a valley-like shape is observable in the performance chart after the application of frequency scaling

(see section 3.4.3). On the other hand, for large matrix sizes, the cache will be stressed the most, with constant requests on the DRAM-Cache controller limiting the undervoltage. Consequently, the results will be similar to those that were observed in section 3.3.2 and 3.4.2. Figure 5.5 illustrates the results of the conducted experiment procedure and confirms the prediction: for small matrix sizes, it is possible to perform a higher degree of undervoltage.

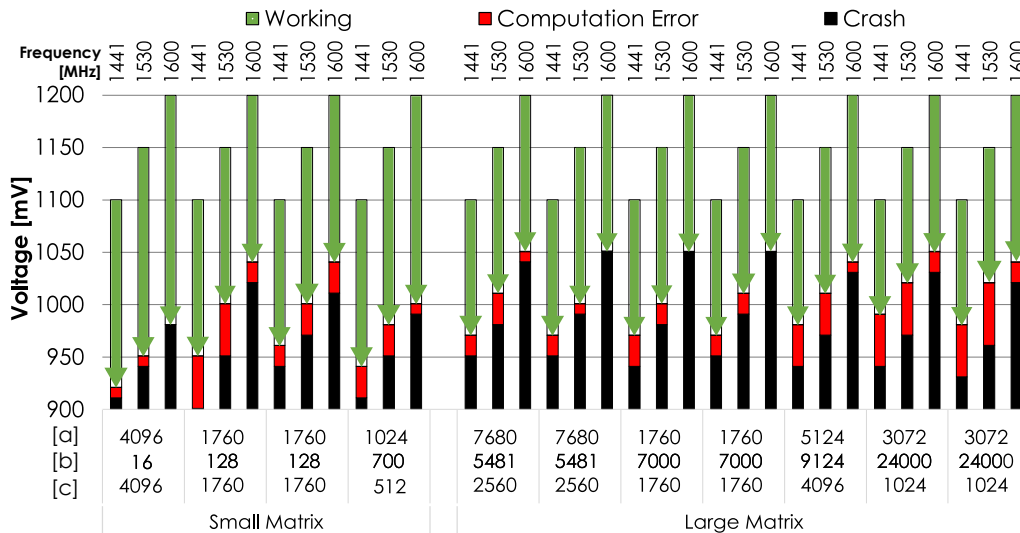


Figure 5.5: Core Domain - Fully-Connected Layer - Usable GPU core voltage range. [a], [b] and [c] values represent matrix sizes (example $A_{a \times b} \cdot B_{b \times c}$).

From the observation of the results presented in Figures 5.6 and 5.7 it is possible to conclude that an improvement in the execution time and energy consumption can be achieved for both types of computations. The EDP chart indicates the same energy efficiency configuration for both cases, which results in an average reduction of 52% in energy consumption and 8% of improvement of the execution time.

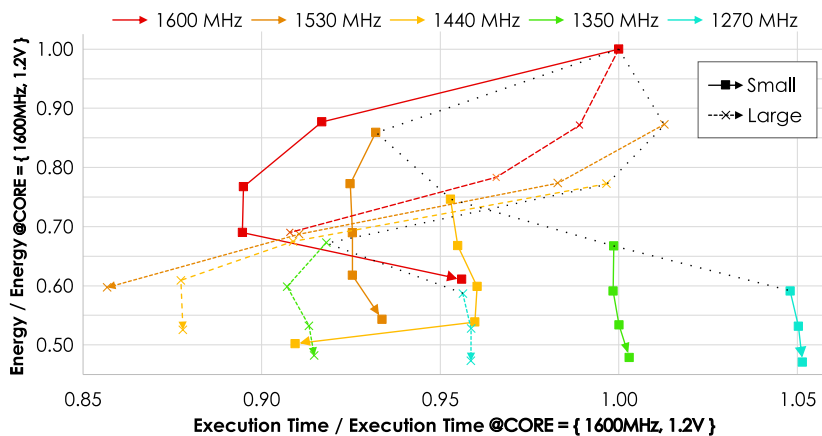


Figure 5.6: Core Domain - Fully-Connected Layer normalized energy and performance chart to non-conventional V-F configurations..

5.3.3 Error Analysis

To evaluate the occurrence of computation errors due to the utilization of non-conventional V-F scaling, each benchmark was executed with the default "automatic" parameterization and with the conven-

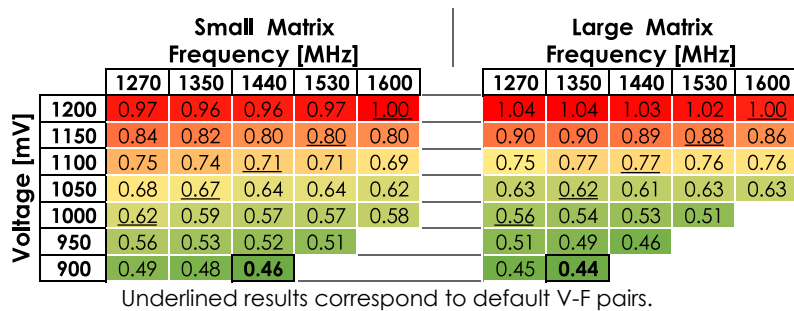


Figure 5.7: Core Domain - Fully-Connected Layer - Obtained normalized Energy-Delay Product heatmap.

tional V-F pair under testing with the same input data. A *warmup_kernel* was also executed before each of these two runs, to fill the cache with random data.

For the error analysis of the CNN layers, a different error metric was adopted due to the utilization of software libraries (versus custom kernels) operating over floating-point numbers (as before, generated from an uniform distribution in the interval $[0.1 ; 1]$ to ensure that operations are never applied to numbers with significantly different exponent values). These libraries can launch the kernels in a different order, changing the order of operations, with a possible impact in the final result. In fact, by conducting experiments on the default voltage, it was observed that the kernel execution order resulted in a relative output difference not greater than 10^{-6} . In accordance, a computation error was asserted whenever the relative difference in each position of the output vectors was greater than or equal to 10^{-5} .

Convolution Layer

Figure 5.8 depicts the distribution of the output results of the convolution layer for the three considered convolution algorithms (at both inference and training phases) for the minimum usable voltage values (i.e., before GPU crash) across all considered core frequency values. The obtained results emphasize the little effect of the applied undervoltage on the computed values. Most of the output results are still almost entirely accurate, and only a small portion of the results present deviations. In fact, it should be emphasized that not only is the fraction of non-accurate results very small, but the normalized relative error of those non-accurate results has a shallow magnitude. A particular observation is worth noting about the results of the inference phase with the *GEMM* algorithm. Although the amount of non-accurate results is greater than in the other configurations, the normalized relative error's magnitude is much smaller.

Fully Connected Layer

Figure 5.9 represents the same error evaluation for the Fully Connected Layer, for both small and large matrices. Even at these extreme configurations, it is observed that most results are still computed with full accuracy (98% of the cases), with a normalized relative error as low as 1.37×10^{-3} (on the remaining 2% of the cases).

Hence, the obtained results demonstrate that the amount of error introduced by these lowest voltages before crash conditions, on the two most important layers, are cooped with both DNN training and inference, demonstrating the ample viability of this approach.

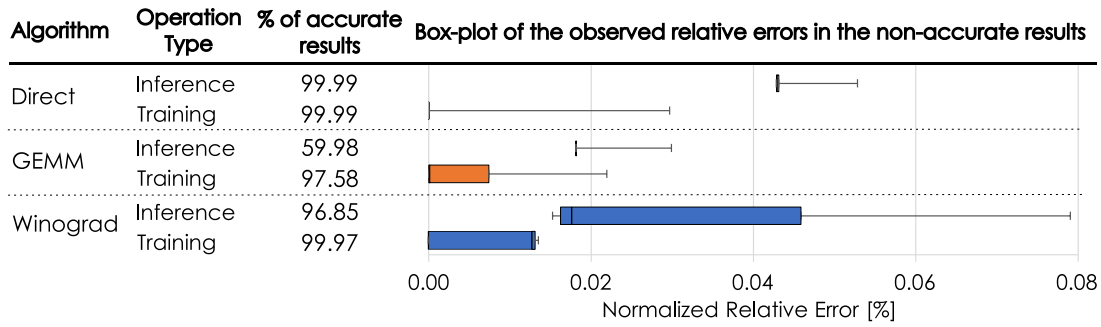


Figure 5.8: Convolution Layer - Average percentage of accurate results and relative error distribution of non-accurate outputs for the minimum usable core voltage across all considered core frequency values.

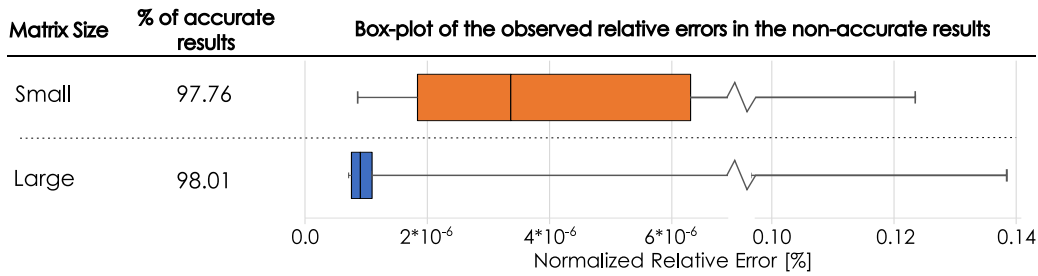


Figure 5.9: Fully Connected Layer- Average percentage of accurate results and relative error distribution of non-accurate outputs for the minimum usable core voltage across all considered core frequency values.

5.4 CNN training and inference with non-conventional V-F

The preliminary assessment of the main CNN primitives, presented in the previous sections, exhibits the same behavior of the primitive operations analysis, performed in Chapter 3. From the obtained results, it can be concluded that it is possible to safely undervolt the GPU of the two main CNN primitives without compromising the accuracy and with relevant energy and performance gains. In accordance, the logical next step is to adapt the optimization mechanism described in Chapter 4 to the training procedure of complete state-of-the-art CNNs.

This section starts by presenting the usability of non-conventional V-F on the complete training of CNNs, followed by a description of how the optimization mechanism was adapted to achieve the same energy efficiency gains automatically.

5.4.1 Feasibility Assessment

The results that were obtained in the previous section evidence that the small number and relative amount of computation errors introduced by these near threshold conditions is well cooped with the operations that are conducted in DNN layers. However, the same evaluation urged to be done for the whole network.

Following the same methodology as before, an exploration of decoupled frequency and voltage variations was performed for the training + inference and inference phase of four complete well-know CNN models. The tested models are LeNet [68], VGG11 [69], AlexNet [70] and WideResNet [71], corresponding to increasing model sizes, complexity, and characteristics. This feasibility assessment will prove that the considered voltage and frequency scaling is allowed by the algorithms under execution, both with

negligent impacts on the models' final training accuracy and with considerable energy-efficiency improvements. To guarantee that the changing V-F is the only source of the observed variation, in all tests, the networks' weights are initialized with the same value.

Training and Inference

Table 5.2 presents the median classification accuracy over ten training runs of the considered networks after applying four different undervoltage levels. The obtained results demonstrate that, when compared with the default setup (i.e., undervolt = 0), the introduced computation errors do not induce any significant change in the network's final training accuracy. In more detail, Figure 5.10, presents the behavior of the loss and model accuracy of each network, measured on the test set over the training session on all the tested V-F pairs. It is possible to observe that the two metrics' progress is, within small variations, the same for all tested V-F configurations.

Amount of undervolt [mV]	AlexNet [%]	LeNet [%]	VGG11 [%]	WideResNet [%]
0	76.59	59.84	86.14	80.32
50	76.48	60.08	86.14	80.39
100	76.60	59.94	86.04	80.23
150	76.61	60.12	86.39	80.08
Number of trained epochs	50	100	30	30

Table 5.2: Comparing CNN training test set accuracy with the application of different undervoltage levels.

Figures 5.11 and 5.12 depict the energy performance charts and EDP results for the conducted V-F exploration. Of significant importance is the comparison of the automatic DVFS system versus the non-conventional V-F configurations, represented as a black dot Figure 5.11. For this configuration, the training procedure was executed, while allowing the DVFS system to vary the current V-F pair and adjust all parameters automatically. In neither of the four tested models, the automatic system can achieve the best performance or energy consumption, demonstrating the analysis provided in Chapter 2, that the automatic DVFS systems of GPUs are not able to take full advantage of the hardware.

In particular, the default V-F pairs are able to produce either the lowest energy consumption or the highest performance. The main benefit of exploring non-conventional V-F is allowing for higher or even the highest frequency (maximizing performance) while having similar energy consumption to using the lower frequency/energy savings performance levels. Overall, these new configurations decrease the EDP of the training sessions the most, as depicted in Figure 5.12.

Table 5.3 emphasizes and summarizes the main achievements when running the CNN training under non-conventional V-F. It compares the CNN execution at the default V-F setup, with the results obtained with frequency scaling using the default voltage values to non-conventional V-F. It considers two configurations: (i) at the highest frequency, and (ii) at minimum EDP. As it can be observed, by exploring non-conventional V-F, it is possible to significantly improve all three metrics (energy, training time, and EDP) without compromising the resulting accuracy of the whole CNN.

Inference

The inference of the same CNN models was also tested independently. This test tries to understand if the same V-F configuration should be used for the algorithm's inference part. Figure 5.13 exhibits

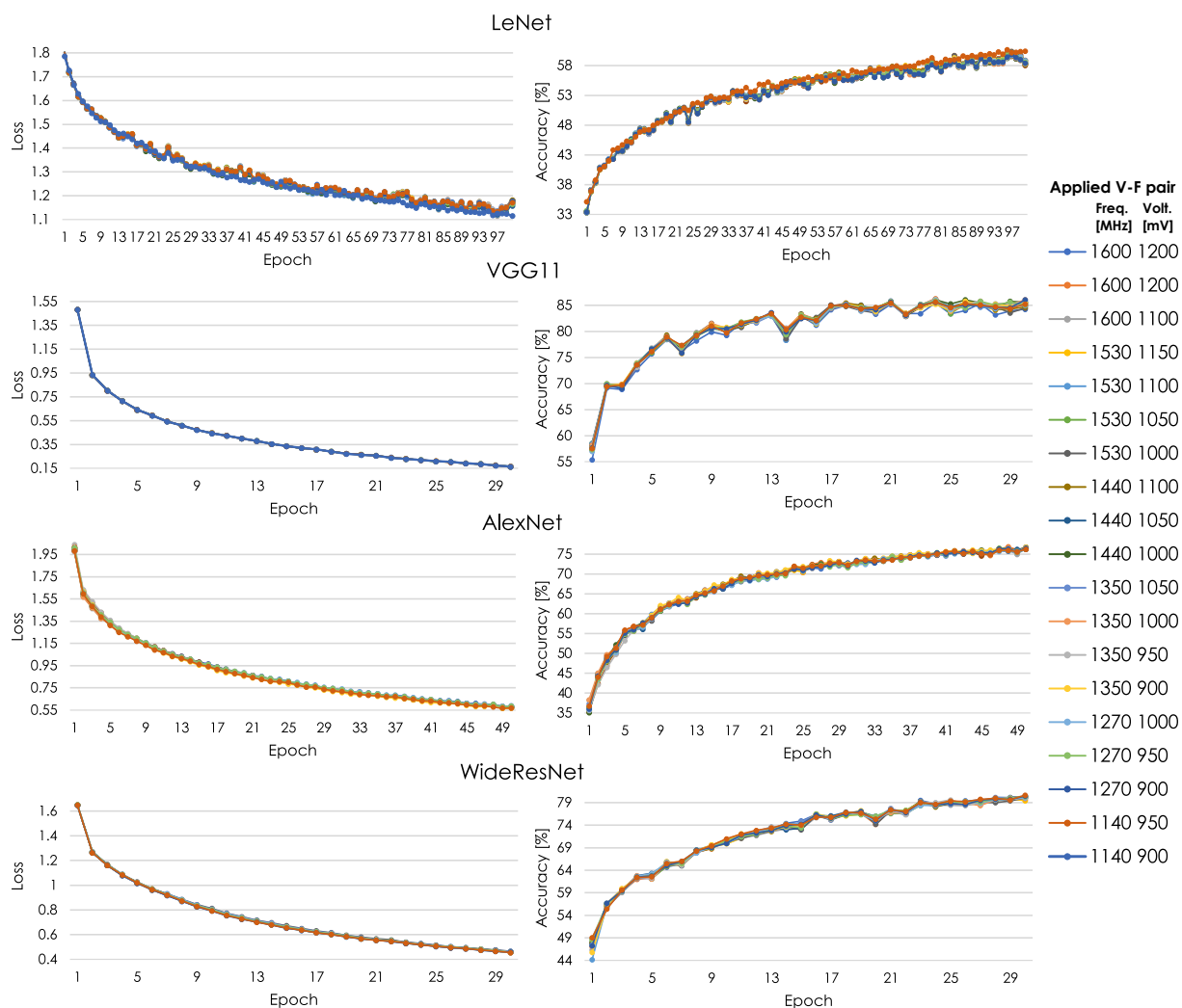


Figure 5.10: Core domain - CNN models - superposition of all V-F configuration test set loss and model accuracy.

the EDP values for the V-F exploration of the four CNN models. By comparing the EDP heat-maps of the training + inference (Figure 5.12) versus the heat-maps of the inference (Figure 5.13), it was observed that the configuration that minimizes the EDP, and so, maximizes the energy-efficiency, only matches on the VGG11 model. However, it should be noted that the same model's heat-maps present an approximate distribution, with the global minimum being around the same V-F configurations for training and inference.

5.4.2 Optimization algorithm adaptation to CNN training

To benefit from automatically adjusting the V-F configuration to CNN training, the developed V-F optimization mechanism, presented in the previous Chapter, was applied to the training procedure.

As the block diagram of Figure 4.1 presents, the V-F optimization mechanism is composed of three stages, *Analysis of user input and test DVFS information*, *Establishment of baseline measurement* and *Optimization Algorithm Execution*. The present subsection indicates how each stage was adapted to the CNN training procedure and, in specific, in which way the *Online Monitorization* is performed on this

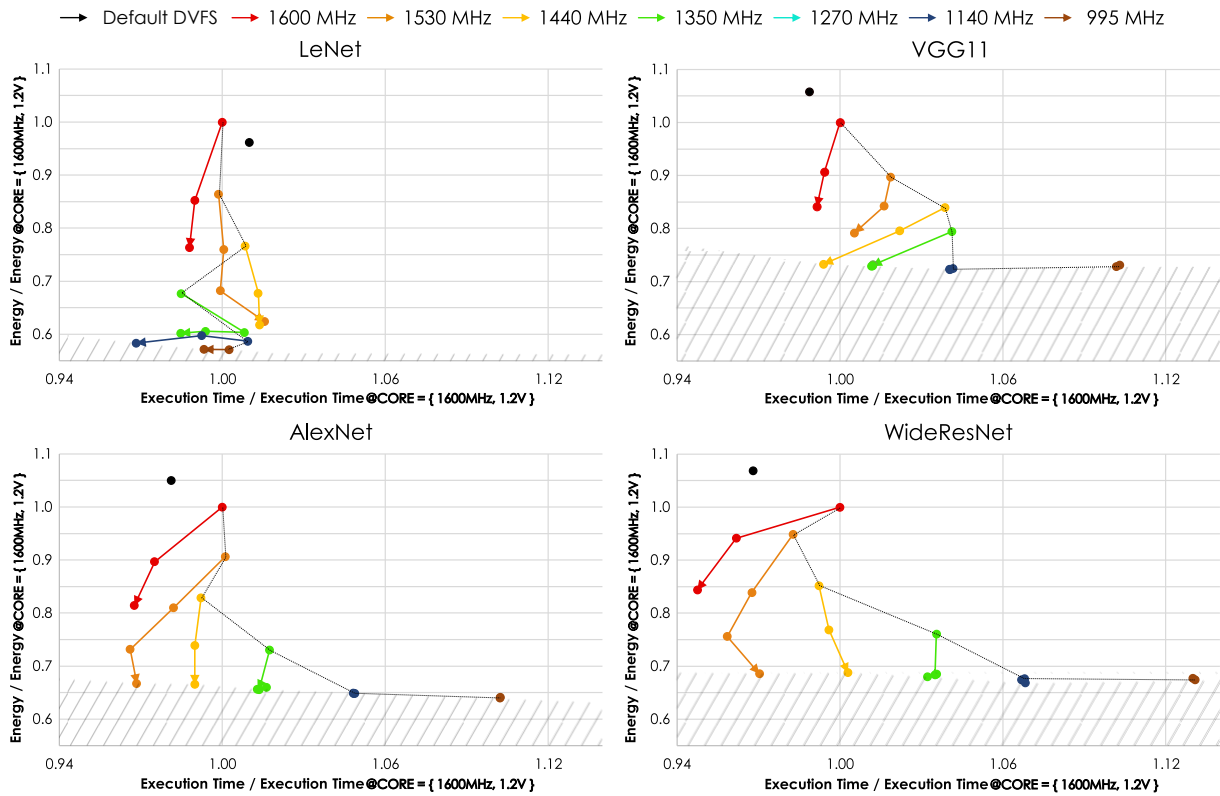
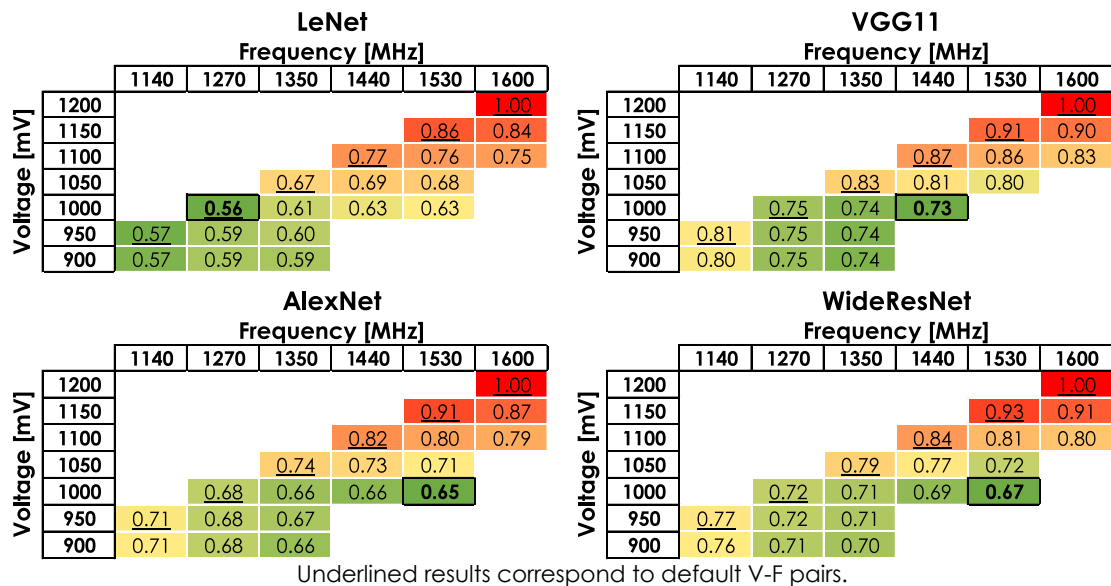


Figure 5.11: Core domain - CNN models - Normalized energy consumption and execution time for training + inference. The dashed line connects the default V-F pairs, and the diagonal striped pattern indicates the plateau of minimum energy consumption.



Underlined results correspond to default V-F pairs.

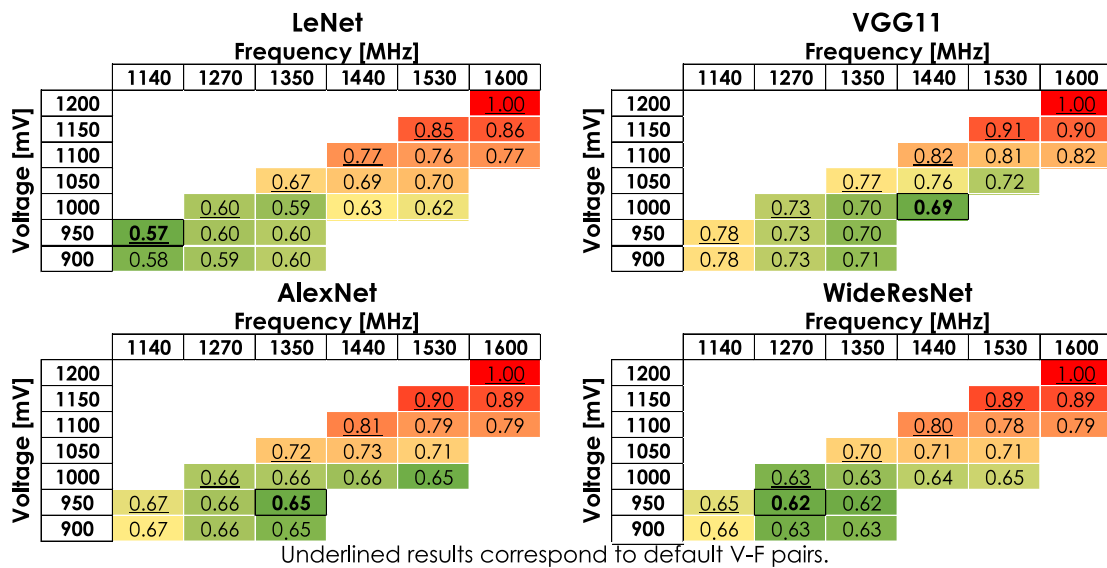
Figure 5.12: Core domain - CNN models - Obtained normalized Energy-Delay Product (EDP) for training + inference.

application. It is first described the last stage of the Optimization Algorithm Execution - *Optimization Algorithm Execution*, because the way it is implemented conditioned how the two other stages were adapted to this application. Similarly to the example given in Chapter 4, the chosen optimization metric was the energy-efficiency computed as the EDP.

Metric	Selected configuration	Improvement vs default V-F			
		AlexNet	LeNet	VGG11	WideResNet
Energy	At highest frequency	24%	20%	22%	23%
	At best EDP	38%	38%	33%	38%
Training time	At highest frequency	1%	2%	0%	6%
	At best EDP	3%	-3%	-2%	0%
EDP	At highest frequency	21%	22%	22%	23%
	At best EDP	38%	41%	32%	36%
Improvement vs F scaling with default V-F pairs					
Energy	At best EDP	-2%	0%	-1%	-2%
Training time	At best EDP	8%	0%	6%	10%
EDP	At best EDP	3%	0%	3%	6%

a positive value indicates an improvement vs the default V-F configuration of the GPU.

Table 5.3: Evaluation of performance, energy, and EDP when applying non-conventional DVFS in the training of neural networks.



Underlined results correspond to default V-F pairs.

Figure 5.13: Core domain - CNN models - Obtained normalized Energy-Delay Product (EDP) for inference phase.

Optimization Algorithm Execution

Chapter 4 described that on the Optimization Algorithm Execution phase of the V-F Optimization Mechanism, the user application is executed on both stages of that phase. In this case, the total number of training epochs would be split by the *Space Exploration* and *Fine-tuning* stages. However, it was observed that the inference EDP heat-map (see Figure 5.13) presents a similar shape to the training + inference EDP heat-map (see Figure 5.12), with both having the $\min(EDP)$ at the same or near the same V-F pair. Nonetheless, executing one inference epoch is significantly faster than running one training + inference epoch. This observation presented the idea of instead of executing the CNN training procedure in the two Optimization Algorithm Execution stages, execute instead just the inference on the *Space Exploration* phase. With this change, at the cost of introducing an overhead to the training procedure since for the first *Space Exploration Steps*, the CNN model is not being trained, it is possible to complete this stage faster than running the training + inference in each epoch. Considering that it is on the *Space Exploration* stage that the less energy-efficient configurations are tested, it is possible to

spend less time testing them and so, reduce the total energy consumption to complete this stage. By doing so, the training procedure of the CNN model starts with a configuration that is better suited for it, with improved energy-efficiency.

Figure 5.14 graphically exemplifies the power consumption overtime of the first ten training epochs of a CNN model to compare the proposed implementation versus the one where the training procedure is executed on both stages of the *Optimization Algorithm Execution*. Rendered in green is the case that was implemented, where ten *Space Exploration Steps* are performed executing just the inference, and after it, on the *Fine Tuning* stage, the CNN training procedure starts. Represented in orange is the case where the CNN training procedure starts at the *Space Exploration* stage, executing in each epoch one training + inference pass. The figure exemplifies that at the end of the *Space Exploration* stage, the same power consumption and execution time (length of the epoch 9 in orange and epoch 0 in green) is achieved to execute a training + inference epoch, so the correspondent EDP value is the same in both cases. Overall, this chart graphically represents the results obtained when adapting the devised V-F Optimization Mechanism to the training procedure of CNN models. The introduced overhead created by not starting the CNN training procedure at the beginning of the Optimization Algorithm Execution phase is offset for more quickly finding a more energy-efficient V-F configuration running just the inference of the CNN model.

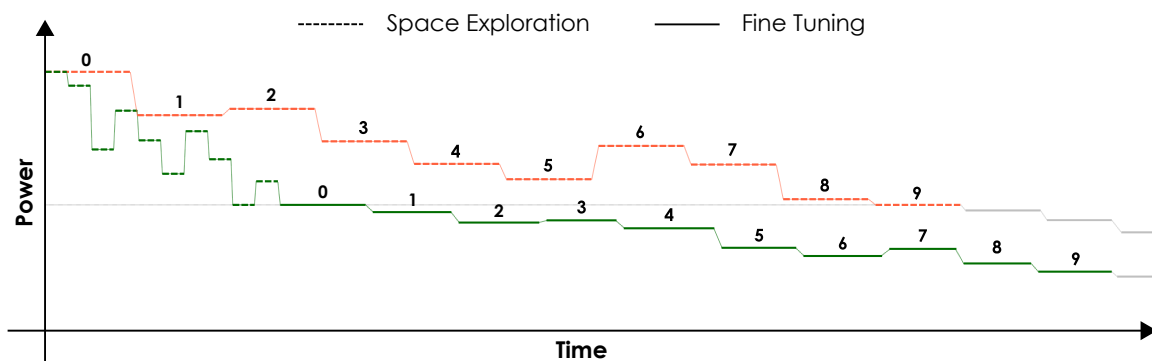


Figure 5.14: Graphical representation of the power consumption overtime of the first ten training epochs of a CNN model (numbers 0 to 9 represent the execution of training epochs). The orange line represents the case where the training procedure is started at the Space Exploration stage, and the green line represents the case where the training procedure is only started on the Fine-Tuning stage.

Analysis of user input and test DVFS information

To set the usable execution space for the CNN training and execution, both the results of the GPU characterization (presented in Chapter 3) and the specific results of the preliminary assessment performed in this chapter (see Section 5.3) were used. Figure 5.15 presents the usable execution space for the Vega 10 GPU, optimized for CNN training and execution.

For the first stage of the Optimization Algorithm Execution (*Space Exploration*), only the green bars that correspond to tested frequencies are considered. However, for the second stage of the Optimization Algorithm Execution (*Fine-Tuning*), the area between the dashed lines, corresponding to the linear regression between the margins of two adjacent frequencies, is also regarded as valid V-F configurations.

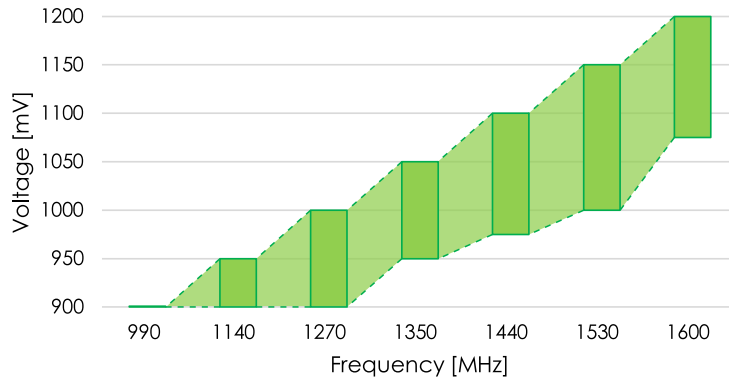


Figure 5.15: Usable Execution Space for the Vega 10 GPU.

Establishment of baseline measurement

In accordance with the adaptation of the V-F Optimization Mechanism to the CNN training procedure described above, two individualized baseline measurements are taken, the first covering one inference epoch and the second containing one training + inference epoch. The inference baseline is used on the *Space Exploration* stage, whereas the training + inference baseline is used for the *Fine Tuning* stage. To execute with the baseline measurements, the GPU was set to the highest V-F configuration {1600MHz; 1.2V}, and the execution time and energy consumption were measured, computing the EDP by multiplying both results.

Online Monitorization

The considered output validity metrics adopted in the optimization mechanism are the loss function value and the network weights' value. If any of these values becomes not a number (NaN), the output is considered invalid. NaN's appearance in these two values is attributed to computational errors that may occur when the used voltage is near V_{min} . To mitigate or at least reduce the computational error, the voltage value is increased by 10 mV, decreasing the amount of applied undervoltage.

5.4.3 Experimental Results

Four different CNN architectures were considered in this evaluation of the proposed V-F Optimization Mechanism, with each model being executed 10 times. The total number of training epochs was the same as the one presented in Table 5.2.

To determine a reasonable number of *Space Exploration Epochs* that guarantees that this stage is able to find a good approximation of the V-F configuration that achieves the global energy efficiency minimum, the *Space Exploration* stage was independently executed on the WideResNet model 50 times. On this evaluation, no limit was set on the number of *Space Exploration Epochs*, since the objective was to analyze how many epochs are necessary to achieve the intended result. It was found that, on average, on the twelfth epoch of *Space Exploration*, the best EDP value is achieved. For such purpose, it was allowed a maximum of 20 *Space Exploration Epochs* when testing the V-F Optimization Mechanism on all the four CNN models, allowing some extra epochs to guarantee that for other models the result is also achieved.

Figure 5.16 presents an example of the V-F configuration and resulting normalized EDP value when applied to the WideResNet model.

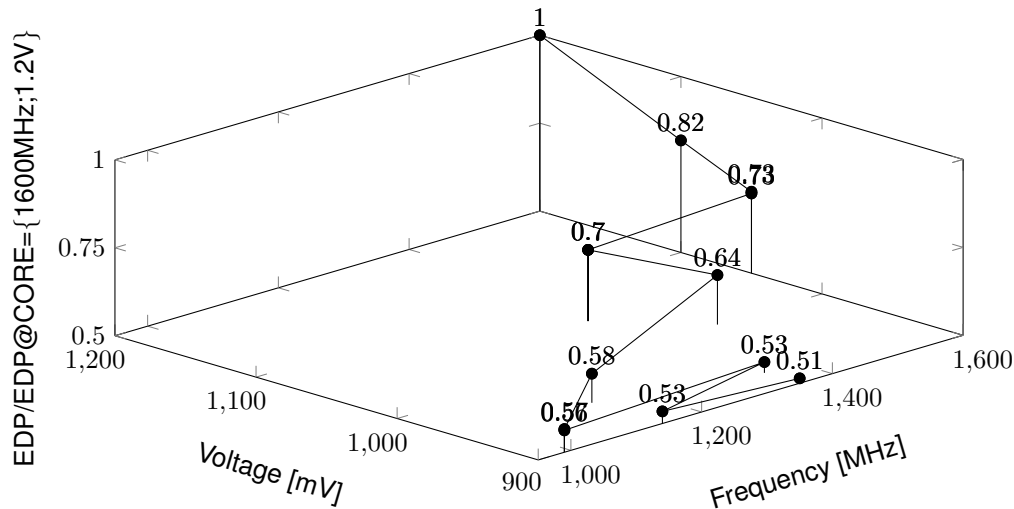


Figure 5.16: WideResNet *Space Exploration* example - normalized EDP values for 20 epochs. Initial value is 1600MHz and 1.2V.

At the end of the *Space Exploration* stage, performed with just an inference epoch, the V-F configuration that achieved the lowest EDP value is passed to the *Fine-Tuning* stage, where the CNN training process is executed.

Overall, the devised V-F Optimization Mechanism fulfilled its purpose and was able to deliver the same degree of energy-efficiency improvement that was achieved in Section 5.4.1 without having to previously test every possible V-F configuration to find the most suitable one.

Figure 5.17 presents and compares the resulting median EDP achieved over the 10 runs of i) the default automatic DVFS system of the Vega 10 GPU - represented in blue; ii) the best V-F configuration found on the feasibility assessment with manual exploration (brute-force) - represented in orange; iii) training with the V-F Optimization Mechanism (that encompasses the overhead of only starting the CNN training procedure on the *Fine Tuning* stage) - represented in green, and iv) the best configuration achievable (attained by performing a new training session with the final configuration chosen by the devised V-F Optimization Mechanism) - represented in grey, all results are normalized to the result achieved by the default automatic DVFS system. As it can be observed, training with the developed mechanism yields a reduction of the EDP of 27% to 42% over the default DVFS system. On the LeNet and AlexNet models, the developed mechanism (with its optimization overhead) is even able to improve on the best configuration found in Section 5.4.1, achieving 42% (against the 41% of the feasibility assessment) for the LeNet and 41% (versus the 37% of the feasibility assessment) of improvement for the AlexNet against the DVFS system included on the Vega 10 GPU. This is due to the finer-tuning of frequency and voltage performed by the optimization mechanism, testing and using V-F pairs between the ones pairs evaluated on Section 5.4.1. For the other two CNN models, the mechanism only falls less than 5% short compared to the results of the feasibility assessment. Nonetheless, the devised system still greatly improves the EDP value on 22% for the VGG11 model and 31% for the WideResNet model, when compared to the default result and is able to achieve this EDP optimization without any prior testing of the CNN model and while performing the final training procedure. Looking at the result indicated

in grey, that establishes itself as the best possible one for the target model on the Vega 10 GPU.

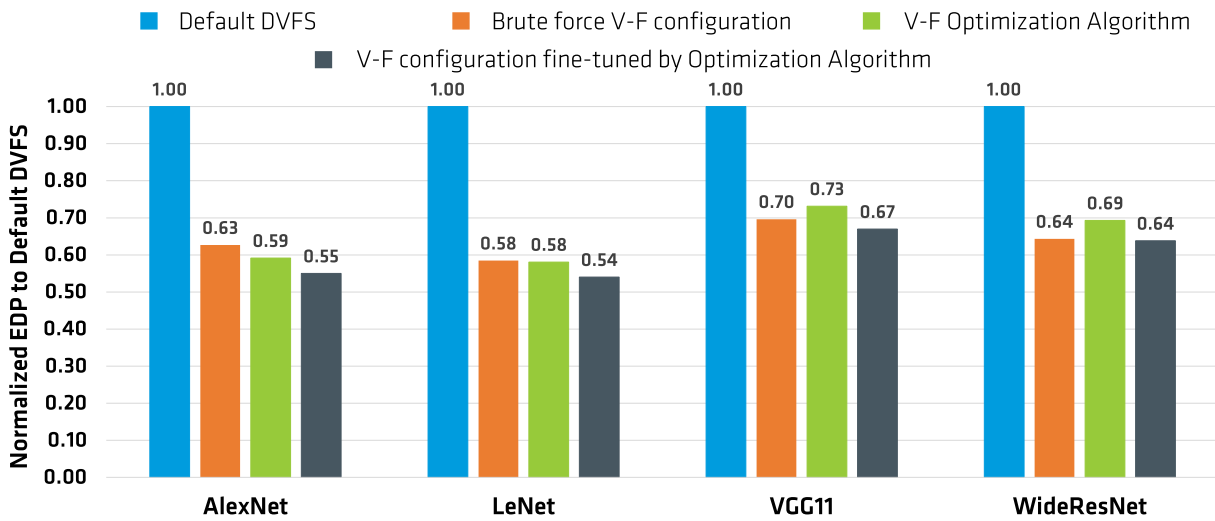


Figure 5.17: Median normalized EDP of default DVFS vs training CNN with V-F optimization model.

Table 5.4 indicates the correspondent V-F configuration at the end of each algorithm execution stage, on the run correspondent to the results of Figure 5.17. The resulting V-F configuration selected by the *Space Exploration* stage corresponds, as expected, to a good approximation to the best possible V-F pair, as it is possible to conclude by observing the EDP heat-map of Figure 5.12. The selected V-F pair at the end of the *Fine Tuning* stage is, in all four cases, in-between the ones that were tested on the feasibility assessment, both in terms of frequency and voltage. This result demonstrates that it is beneficial to finely discretize and tune both these control parameters to the target application, to achieve the optimal energy-efficiency out of the out-of-the-shelf GPUs, as this thesis wanted to demonstrate.

Model	LeNet	VGG11	AlexNet	WideResNet
<i>Space Exploration</i>	[1140MHz; 0.95V]	[1440MHz; 1.0V]	[1350MHz; 0.95V]	[1350MHz; 0.9V]
<i>Fine-Tuning</i>	[1230MHz; 0.95V]	[1370MHz; 0.97V]	[1410MHz; 0.98V]	[1500MHz; 0.99V]

Table 5.4: Mode of selected V-F configurations at the end of each optimization stage.

5.5 Summary

Deep learning and, more specifically, deep neural networks, are a prominent type of algorithm being executed on GPUs nowadays. These types of algorithms use special mathematical operations (like the convolution) to be able to extract features from the input data. The execution (inference) of these algorithms and, more importantly, the training session can significantly benefit if the GPU energy-efficiency is optimized.

This chapter combines all the previous knowledge acquired and presented in the previous chapter of the dissertation. It sets the usable execution space of the convolutional neural networks and it adapts the V-F Optimization Mechanism to the training of this algorithm with excellent results. By characterizing the target GPU to the target application, it was concluded that the energy efficiency of the training session

could be significantly improved, with an EDP reduction of 27% to 42%, depending on the CNN model, over the default DVFS system.

Chapter 6

Conclusions

The main question that steered this master thesis's development was "How can we improve the energy-efficiency of already deployed GPUs?". This question guided all possible solutions to creating and improving the software stack that controls these devices, considering that introducing or modifying already deployed hardware is generally not possible.

The current literature on this topic presents a panoply of strategies to achieve this objective, ranging from improving the device execution units utilization, optimizing the number of threads per block, or performing the computations with reduced precision operands, to tuning the applied voltage-frequency levels to the current GPU state and running application. The followed path was to further investigate this last option, targeting the improvement of current implementations of GPUs DVFS systems. In fact, enhancing the DVFS systems allows to benefit a broader range of both devices and applications, since the improvement does not come from optimizing the application to the target hardware, but instead, from better matching the hardware to the running application.

At this point, the state of the art implementations of DVFS systems make use of many techniques to choose the most appropriate *default* V-F pair for the current device state. However, one could go one step back and argue that the manufacturer's chosen V-F configurations do not allow for extracting the best energy-efficiency of GPUs. Having this premise in mind, the first developed task devised a methodology to analyze the non-conventional V-F configuration's impact on the different architectural components and DVFS domains. The methodology was tested on two architectural different AMD GPUs with great success. Both devices allow for up to 21% voltage reduction, depending on the tested frequency and stressed component. Hence, by running all the benchmarks that compose the methodology was possible to define an usable execution space, consisting of a new set of V-F pairs that, even though they are not the default ones, they do not compromise the GPU architecture. Moreover, the use of those non-conventional V-F configurations allowed for a decreasing the device's energy consumption of up to 40% without (in some cases, it can even improve) penalizing the performance. This also resulted in energy-efficiency gains, measured through the EDP metric, of up to 43%.

After demonstrating that such configurations are so beneficial, the next step was to create a V-F optimization mechanism that explores all those new V-F pairs and, without testing all of them, find and set the most appropriate one for the running application and current GPU state. This iterative optimization mechanism takes advantage of the natural code repetition pattern found in most GPGPU applications to be continually monitoring, adjusting and selecting the fittest V-F configuration.

Deep learning applications, and more specifically, convolutional neural networks (CNNs), were chosen as a target application to demonstrate the added benefits of defining a usable execution space that comprises non-conventional V-F pairs, with the devised iterative V-F optimization mechanism. This application was first demonstrated with the execution of the two most energy-consuming CNN layers, to assess the validity of non-conventional V-F pairs in concrete applications. Naturally, this algorithms' usable execution space corresponds to the one exhibited by the component that they stress the most. Second, the computational errors that emerge when running at the minimum allowed voltage, V_{min} , were studied by comparing the layer output when running conventional V-F pairs versus the non-conventional ones. It was observed that the single output maximum relative error did not exceed 0.14% and that, even in these circumstances, up to 99.99% of the results are shown to be accurate. Finally, the training procedure of four CNN models were executed while exploring the usable execution space, observing no relevant effect on the final testing accuracy being measured when undervolting the GPU core. Moreover, it was observed that using the novel V-F pairs yields an EDP improvement of up to 44% over the regular automatic DVFS system of the Vega 10 GPU. With the adaptation of the V-F mechanism to the training procedure of the CNN, the discovery of the configuration that achieves the best energy efficiency is performed automatically.

The adoption of the devised automatic optimization mechanism, which builds on the benefits of using non-conventional V-F pairs, is this dissertation answer to the objective of improving the energy-efficiency of already deployed GPUs.

6.1 Future Work

Two significant advancements to the state of the art are put forward by this thesis, contributing positively to increase the energy-efficiency of GPUs. However, the devised methodologies still have room to be further explored.

AMD is currently the only manufacturer allowing, through their software stack (ROCm and rocm-smi), independent control over the voltage and frequency pairs used by the DVFS system. However, this manufacturer is currently not the leading player in the GPU space. As such, the demonstration of non-conventional V-F scaling cannot be done on the most used and fastest devices available. Other manufacturers (mainly, the leading GPU manufacturer NVIDIA) must provide similar software tools to allow researchers to continue to explore and create novel solutions. In addition to making these tools available, manufacturers could develop a control API that allows developers to hint their DVFS system (through CUDA, HIP or OpenCL directives) with information regarding application characteristics or the target optimization metric. The manufacturer developed DVFS system would then use that new information to better optimize the V-F pairs values and the selection criteria for the same.

The proposed methodology to characterize the architecture to non-conventional V-F scaling could be further expanded by going one step below and one step above. Going one step below, other new kernels to characterize the architecture in more detail could be added to the methodology. These would use Assembly calls to determine which operations inside each architectural component are actually limiting the usable exploration space. Going one step above, the exploration of the V-F configurations for each benchmark could be done more intelligently, by creating an algorithm that would indicate which configurations to test (in the current implementation, it is necessary to examine all possible configurations).

The devised V-F execution mechanism already considers the application performance and energy-consuming characteristics to choose the most appropriate V-F pair. However, the generation of the V-F pair to be tested is randomly determined. By monitoring each iteration of the user application's performance counters, such information could be used to guide the generation of the new V-F configuration. To allow that, a model that relates the performance counters results with the observed performance and energy-consuming behavior to non-conventional V-F scaling needed to be added to the current optimization mechanism.

Bibliography

- [1] J. Leng, A. Buyuktosunoglu, R. Bertran, P. Bose, and V. J. Reddi. Safe limits on voltage reduction efficiency in GPUs: A direct measurement approach. In *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 294–307, Dec. 2015. doi: 10.1145/2830772.2830811. ISSN: 2379-3155.
- [2] A. Shrestha and A. Mahmood. Review of Deep Learning Algorithms and Architectures. *IEEE Access*, 7:53040–53065, 2019. ISSN 2169-3536. doi: 10.1109/ACCESS.2019.2912200.
- [3] Q. Zhang, T. Wang, Y. Tian, F. Yuan, and Q. Xu. ApproxANN: An approximate computing framework for artificial neural network. In *Design, Automation Test in Europe Conference Exhibition 2015*, pages 701–706, Mar. 2015. ISSN: 1558-1101.
- [4] Z. Tang, Y. Wang, Q. Wang, and X. Chu. The Impact of GPU DVFS on the Energy and Performance of Deep Learning: an Empirical Study. In *Proceedings of the Tenth ACM International Conference on Future Energy Systems, e-Energy '19*, pages 315–325, Phoenix, AZ, USA, June 2019. Association for Computing Machinery. ISBN 978-1-4503-6671-7. doi: 10.1145/3307772.3328315.
- [5] Jon Peddie Research releases its Q2, 2019 global add-in board report, 2018. URL <https://www.jonpeddie.com/press-releases/jon-peddie-research-releases-its-q2-2019-global-add-in-board-report/>.
- [6] H. Mujtaba. AMD Radeon Graphics Cards Gain Market Share Versus NVIDIA GeForce, Sept. 2019. URL <https://wccftech.com/amd-radeon-nvidia-geforce-graphics-card-gpu-market-share-q2-2019/>.
- [7] NVIDIA. CUDA C++ Programming Guide, 2008. URL <http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html>.
- [8] AMD. AMD Accelerated Parallel Processing: OpenCL Programming Guide, 2008. URL http://developer.amd.com/wordpress/media/2013/07/AMD_Accelerated_Parallel_Processing_OpenCL_Programming_Guide-rev-2.7.pdf.
- [9] N. Jing, Y. Shen, Y. Lu, S. Ganapathy, Z. Mao, M. Guo, R. Canal, and X. Liang. An Energy-efficient and Scalable eDRAM-based Register File Architecture for GPGPU. In *Proceedings of the 40th Annual International Symposium on Computer Architecture, ISCA '13*, pages 344–355, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2079-5. doi: 10.1145/2485922.2485952. URL <http://doi.acm.org/10.1145/2485922.2485952>.

- [10] NVIDIA. CUDA Zone, July 2017. URL <https://developer.nvidia.com/cuda-zone>.
- [11] OpenCL - The open standard for parallel programming of heterogeneous systems, July 2013. URL <https://www.khronos.org/opencv/>.
- [12] W.-m. W. Hwu. *Heterogeneous System Architecture: A New Compute Platform Infrastructure*. Morgan Kaufmann, Amsterdam : Waltham, MA, 1 edition edition, Dec. 2015. ISBN 978-0-12-800386-2.
- [13] RadeonOpenCompute/ROCm, Nov. 2019. URL <https://github.com/RadeonOpenCompute/ROCm>. original-date: 2016-03-18T00:24:29Z.
- [14] S.-L. Zhang and M. Östling. Metal Silicides in CMOS Technology: Past, Present, and Future Trends. *Critical Reviews in Solid State and Materials Sciences*, 28(1):1–129, Nov. 2003. ISSN 1040-8436. doi: 10.1080/10408430390802431. URL <https://doi.org/10.1080/10408430390802431>.
- [15] W. Schemmert and G. Zimmer. Threshold-voltage sensitivity of ion-implanted m.o.s. transistors due to process variations. *Electronics Letters*, 10(9):151, 1974. ISSN 00135194. doi: 10.1049/el:19740115. URL https://digital-library.theiet.org/content/journals/10.1049/el_19740115.
- [16] R. Thomas, K. Barber, N. Sedaghati, L. Zhou, and R. Teodorescu. Core tunneling: Variation-aware voltage noise mitigation in GPUs. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 151–162, Mar. 2016. doi: 10.1109/HPCA.2016.7446061. ISSN: 2378-203X.
- [17] S. S. Sapatnekar. What happens when circuits grow old: Aging issues in CMOS design. *2013 International Symposium on VLSI Technology, Systems and Application (VLSI-TSA)*, pages 1–2, 2013. doi: 10.1109/VLSI-TSA.2013.6545621.
- [18] D. Wolpert and P. Ampadu. Temperature Effects in Semiconductors. In *Managing Temperature Effects in Nanoscale Adaptive Systems*, pages 15–33. Springer New York, New York, NY, 2012. ISBN 978-1-4614-0747-8 978-1-4614-0748-5. doi: 10.1007/978-1-4614-0748-5_2.
- [19] J. F. Freijedo, J. Semião, J. J. Rodriguez-Andina, F. Vargas, I. C. Teixeira, and J. P. Teixeira. Modeling the Effect of Process, Power-Supply Voltage and Temperature Variations on the Timing Response of Nanometer Digital Circuits. *Journal of Electronic Testing*, 28(4):421–434, Aug. 2012. ISSN 0923-8174, 1573-0727. doi: 10.1007/s10836-012-5297-0.
- [20] R. Gonzalez, B. Gordon, and M. Horowitz. Supply and threshold voltage scaling for low power CMOS. *IEEE Journal of Solid-State Circuits*, 32(8):1210–1216, Aug. 1997. ISSN 1558-173X. doi: 10.1109/4.604077.
- [21] X. Mei, Q. Wang, and X. Chu. A Survey and Measurement Study of GPU DVFS on Energy Conservation. *Digital Communications and Networks*, 3(2):89 – 100, 2017. ISSN 2352-8648. doi: <https://doi.org/10.1016/j.dcan.2016.10.001>.
- [22] S. Hong. *Modeling performance and power for energy-efficient GPGPU computing*. PhD thesis, Georgia Institute of Technology, 2012.

- [23] S. Hong and H. Kim. An Integrated GPU Power and Performance Model. In *Proceedings of the 37th Annual International Symposium on Computer Architecture, ISCA '10*, pages 280–289, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0053-7. doi: 10.1145/1815961.1815998.
- [24] M. Boyer. *Improving Resource Utilization in Heterogeneous CPU-GPU Systems*. PhD thesis, University of Virginia, Apr. 2013.
- [25] K. SeongKi and K. Young. GPGPU-Perf: efficient, interval-based DVFS algorithm for mobile GPGPU applications. *Springer Berlin Heidelberg*, Apr. 2015. ISSN 1432-2315. doi: <https://doi.org/10.1007/s00371-015-1111-1>.
- [26] C. Da-Ren, C. Young-Long, and C. You-Shyang. Time and Energy Efficient DVS Scheduling for Real-Time Pinwheel Tasks. *Journal of Applied Research and Technology*, 12(6):1025 – 1039, 2014. ISSN 1665-6423. doi: [https://doi.org/10.1016/S1665-6423\(14\)71663-3](https://doi.org/10.1016/S1665-6423(14)71663-3).
- [27] H. Hoffmann. Racing and Pacing to Idle: An Evaluation of Heuristics for Energy-aware Resource Allocation. In *Proceedings of the Workshop on Power-Aware Computing and Systems, HotPower '13*, pages 13:1–13:5, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2458-8. doi: 10.1145/2525526.2525854.
- [28] D. H. Kim, C. Imes, and H. Hoffmann. Racing and Pacing to Idle: Theoretical and Empirical Analysis of Energy Optimization Heuristics. In *2015 IEEE 3rd International Conference on Cyber-Physical Systems, Networks, and Applications*, pages 78–85, Aug. 2015. doi: 10.1109/CPSNA.2015.23.
- [29] AMD. The Polaris Architecture |. page 22, 2017.
- [30] J. Guerreiro, A. Ilic, N. Roma, and P. Tomás. DVFS-aware application classification to improve GPGPUs energy efficiency. *Parallel Computing*, 83:93–117, 2019. ISSN 0167-8191. doi: 10.1016/j.parco.2018.02.001. Publisher: Elsevier.
- [31] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi. GPUWatch: Enabling Energy Optimizations in GPGPUs. In *Proceedings of the 40th Annual International Symposium on Computer Architecture, ISCA '13*, pages 487–498, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2079-5. doi: 10.1145/2485922.2485964.
- [32] Y. Jiao, H. Lin, P. Balaji, and W. Feng. Power and Performance Characterization of Computational Kernels on the GPU. In *2010 IEEE/ACM Int'l Conference on Green Computing and Communications Int'l Conference on Cyber, Physical and Social Computing*, pages 221–228, Dec. 2010. doi: 10.1109/GreenCom-CPSCom.2010.143.
- [33] R. Ge, R. Vogt, A. J. Majumder, M. A. U. Alam, M. Burtscher, and Z. Zong. Effects of Dynamic Voltage and Frequency Scaling on a K20 GPU. pages 826–833, Oct. 2013. doi: 10.1109/ICPP.2013.98.
- [34] Y. Abe, H. Sasaki, M. Peres, K. Inoue, K. Murakami, and S. Kato. Power and Performance Analysis of GPU-accelerated Systems. In *Proceedings of the 2012 USENIX Conference on Power-Aware Computing and Systems, HotPower'12*, pages 10–10, Berkeley, CA, USA, 2012. USENIX Association.

- [35] X. Mei, L. Yung, K. Zhao, and X. Chu. A measurement study of GPU DVFS on energy conservation. In *Proceedings of the Workshop on Power-Aware Computing and Systems, HotPower 2013*, Nov. 2013. doi: 10.1145/2525526.2525852.
- [36] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *2009 IEEE International Symposium on Workload Characterization (IISWC)*, pages 44–54, Austin, TX, USA, Oct. 2009. IEEE. ISBN 978-1-4244-5156-2. doi: 10.1109/IISWC.2009.5306797. URL <http://ieeexplore.ieee.org/document/5306797/>.
- [37] J. Cha, J. Kim, and Y. Park. Core-level DVFS for Spatial Multitasking GPUs. In *TENCON 2018 - 2018 IEEE Region 10 Conference*, pages 1525–1528, Oct. 2018. doi: 10.1109/TENCON.2018.8650072. ISSN: 2159-3450, 2159-3442.
- [38] J. A. Stratton, C. Rodrigues, I.-J. Sung, N. Obeid, L.-W. Chang, N. Anssari, G. D. Liu, and W.-m. W. Hwu. Parboil: A Revised Benchmark Suite for Scientific and Commercial Throughput Computing. page 12.
- [39] PolyBench/C – Homepage of Louis-Noël Pouchet. URL <http://web.cs.ucla.edu/~pouchet/software/polybench/>.
- [40] A. Sethia and S. Mahlke. Equalizer: Dynamic Tuning of GPU Resources for Efficient Execution. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 647–658, Dec. 2014. doi: 10.1109/MICRO.2014.16. ISSN: 2379-3155.
- [41] W. Thomas and R. D. Daruwala. Application aware Scalable Architecture for GPGPU. *Journal of Systems Architecture*, 89:73–83, Sept. 2018. ISSN 1383-7621. doi: 10.1016/j.sysarc.2018.07.003. URL <http://www.sciencedirect.com/science/article/pii/S1383762117304241>.
- [42] S. Akiki, Z. Yang, C. Liu, J. Tang, and S. Liu. Energy-Aware Automatic Tuning of Many-Core Platform via Gradient Descent. In *2018 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computing, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCOM/IOP/SCI)*, pages 1199–1203, Oct. 2018. doi: 10.1109/SmartWorld.2018.00209.
- [43] Y. Huang, B. Guo, and Y. Shen. GPU Energy Consumption Optimization With a Global-Based Neural Network Method. *IEEE Access*, 7:64303–64314, 2019. ISSN 2169-3536. doi: 10.1109/ACCESS.2019.2915380.
- [44] G. Papadimitriou, A. Chatzidimitriou, D. Gizopoulos, V. J. Reddi, J. Leng, B. Salami, O. S. Unsal, and A. C. Kestelman. Exceeding Conservative Limits: A Consolidated Analysis on Modern Hardware Margins. *IEEE Transactions on Device and Materials Reliability*, 20(2):341–350, June 2020. ISSN 1558-2574. doi: 10.1109/TDMR.2020.2989813. Conference Name: IEEE Transactions on Device and Materials Reliability.
- [45] J. Leng, Y. Zu, M. Rhu, M. Gupta, and V. J. Reddi. GPUVolt: modeling and characterizing voltage noise in GPU architectures. In *Proceedings of the 2014 international symposium on Low power electronics and design - ISLPED '14*, pages 141–146, La Jolla, California, USA, 2014. ACM Press.

- ISBN 978-1-4503-2975-0. doi: 10.1145/2627369.2627605. URL <http://dl.acm.org/citation.cfm?doid=2627369.2627605>.
- [46] F. Nakhaee, M. Kamal, A. Afzali-Kusha, M. Pedram, S. M. Fakhraie, and H. Dorosti. Lifetime improvement by exploiting aggressive voltage scaling during runtime of error-resilient applications. *Integration*, 61:29–38, Mar. 2018. ISSN 0167-9260. doi: 10.1016/j.vlsi.2017.10.013. URL <http://www.sciencedirect.com/science/article/pii/S0167926017306363>.
- [47] J. Tan, S. L. Song, K. Yan, X. Fu, A. Marquez, and D. Kerbyson. Combating the Reliability Challenge of GPU Register File at Low Supply Voltage. In *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation - PACT '16*, pages 3–15, Haifa, Israel, 2016. ACM Press. ISBN 978-1-4503-4121-9. doi: 10.1145/2967938.2967951. URL <http://dl.acm.org/citation.cfm?doid=2967938.2967951>.
- [48] X. Jiao, M. Luo, J.-H. Lin, and R. K. Gupta. An assessment of vulnerability of hardware neural networks to dynamic voltage and temperature variations. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 945–950, Nov. 2017. doi: 10.1109/ICCAD.2017.8203882. ISSN: 1558-2434.
- [49] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pages 361–372, Minneapolis, MN, USA, June 2014. IEEE. ISBN 978-1-4799-4394-4 978-1-4799-4396-8. doi: 10.1109/ISCA.2014.6853210. URL <http://ieeexplore.ieee.org/document/6853210/>.
- [50] J. Guerreiro, A. Ilic, N. Roma, and P. Tomas. GPGPU Power Modeling for Multi-domain Voltage-Frequency Scaling. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 789–800, 2018. doi: 10.1109/HPCA.2018.00072.
- [51] D. E. Vaughan, S. H. Jacobson, S. N. Hall, and L. A. McLay. Simultaneous Generalized Hill-Climbing Algorithms for Addressing Sets of Discrete Optimization Problems. *INFORMS Journal on Computing*, 17(4):438–450, Nov. 2005. ISSN 1091-9856, 1526-5528. doi: 10.1287/ijoc.1040.0064. URL <http://pubsonline.informs.org/doi/10.1287/ijoc.1040.0064>.
- [52] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671, May 1983. doi: 10.1126/science.220.4598.671. URL <http://science.sciencemag.org/content/220/4598/671.abstract>.
- [53] D. Li, X. Chen, M. Becchi, and Z. Zong. Evaluating the Energy Efficiency of Deep Convolutional Neural Networks on CPUs and GPUs. In *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom)*, pages 477–484, Oct. 2016. doi: 10.1109/BDCloud-SocialCom-SustainCom.2016.76.
- [54] Y. Bengio, A. Courville, and P. Vincent. Representation Learning: A Review and New Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, Aug. 2013. ISSN 1939-3539. doi: 10.1109/TPAMI.2013.50.

- [55] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, Jan. 2015. ISSN 0893-6080. doi: 10.1016/j.neunet.2014.09.003. URL <http://www.sciencedirect.com/science/article/pii/S0893608014002135>.
- [56] S. Dong and D. Kaeli. DNNMark: A Deep Neural Network Benchmark Suite for GPUs. In *Proceedings of the General Purpose GPUs, GPGPU-10*, pages 63–72, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4915-4. doi: 10.1145/3038228.3038239. URL <http://doi.acm.org/10.1145/3038228.3038239>. event-place: Austin, TX, USA.
- [57] G. B. Orr and K.-R. Müller, editors. *Neural Networks: Tricks of the Trade*. Lecture Notes in Computer Science. Springer-Verlag, Berlin Heidelberg, 1998. ISBN 978-3-540-49430-0. doi: 10.1007/3-540-49430-8. URL <https://www.springer.com/gp/book/9783540494300>.
- [58] C. LeCun, Yann and Cortes. MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges, 1999. URL <http://yann.lecun.com/exdb/mnist/>.
- [59] A. Krizhevsky. Learning Multiple Layers of Features from Tiny Images. pages 32–35, 2009.
- [60] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, June 2009. doi: 10.1109/CVPR.2009.5206848. ISSN: 1063-6919.
- [61] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, Oct. 1986. ISSN 1476-4687. doi: 10.1038/323533a0. URL <https://www.nature.com/articles/323533a0>. Number: 6088 Publisher: Nature Publishing Group.
- [62] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv:1609.04747 [cs]*, June 2017. URL <http://arxiv.org/abs/1609.04747>. arXiv: 1609.04747.
- [63] A. Jain, A. A. Awan, Q. Anthony, H. Subramoni, and D. K. D. Panda. Performance Characterization of DNN Training using TensorFlow and PyTorch on Modern Clusters. In *2019 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 1–11, Sept. 2019. doi: 10.1109/CLUSTER.2019.8891042. ISSN: 2168-9253.
- [64] S. Mittal and S. Vaishay. A survey of techniques for optimizing deep learning on GPUs. *Journal of Systems Architecture*, 99:101635, Oct. 2019. ISSN 1383-7621. doi: 10.1016/j.sysarc.2019.101635. Original Publisher: Elsevier.
- [65] S. M. Nabavinejad, H. Hafez-Kolahi, and S. Reda. Coordinated DVFS and Precision Control for Deep Neural Networks. *IEEE Computer Architecture Letters*, 18(2):136–140, July 2019. ISSN 2473-2575. doi: 10.1109/LCA.2019.2942020.
- [66] J. Khan, P. Fultz, A. Tamazov, D. Lowell, C. Liu, M. Melesse, M. Nandhimandalam, K. Nasyrov, I. Perminov, T. Shah, V. Filippov, J. Zhang, J. Zhou, B. Natarajan, and M. Daga. MIOpen: An Open Source Library For Deep Learning Primitives. *arXiv:1910.00078 [cs, stat]*, Sept. 2019. arXiv: 1910.00078.

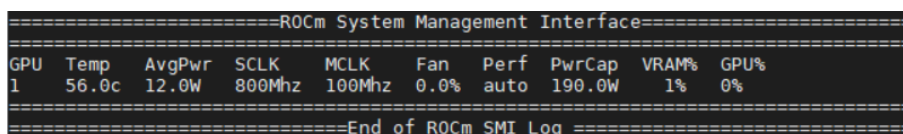
- [67] I. Masliah, A. Abdelfattah, A. Haidar, S. Tomov, M. Baboulin, J. Falcou, and J. Dongarra. High-Performance Matrix-Matrix Multiplications of Very Small Matrices. In P.-F. Dutot and D. Trystram, editors, *Euro-Par 2016: Parallel Processing*, volume 9833, pages 659–671. Springer International Publishing, Cham, 2016. ISBN 978-3-319-43658-6 978-3-319-43659-3. doi: 10.1007/978-3-319-43659-3_48. Series Title: Lecture Notes in Computer Science.
- [68] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov. 1998. ISSN 1558-2256. doi: 10.1109/5.726791. Conference Name: Proceedings of the IEEE.
- [69] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv:1409.1556 [cs]*, Apr. 2015. URL <http://arxiv.org/abs/1409.1556>. arXiv: 1409.1556.
- [70] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>.
- [71] S. Zagoruyko and N. Komodakis. Wide Residual Networks. *arXiv:1605.07146 [cs]*, June 2017. URL <http://arxiv.org/abs/1605.07146>. arXiv: 1605.07146.

Appendix A

How to control and set the desired V-F pair with rocm-smi

As introduced in this dissertation, the independent control of frequency and voltage is not widely available, and only recently, AMD allows it through their ROCm software stack, more specifically through rocm-smi - ROCm system management interface. However, even the most recent version of this tool (presented on ROCm 3.8) does not provide an easy and utterly understandable way of setting the desired frequency and voltage pair. This appendix intends to describe how to command the rocm-smi tool to control and set the desired V-F pair in the two tested GPUs, AMD Vega 10 Frontier Edition and AMD Radeon 5700 XT.

As described in Chapter 2, rocm-smi is a Linux command line application. When launch, it prompts the interface exhibited in Figure A.1 where it is possible to check the device temperature, average device power, core domain clock - `sclk`, DVFS domain clock - `mclk`, percentage of fan speed, `perf` - indicating current DVFS setting as automatic or manual, current device power cap and finally, DRAM and Core utilization.



```
=====ROCm System Management Interface=====
GPU  Temp  AvgPwr  SCLK  MCLK  Fan  Perf  PwrCap  VRAM%  GPU%
1    56.0c  12.0W  800Mhz 100Mhz 0.0% auto  190.0W  1%   0%
=====End of ROCm SMI Log=====
```

Figure A.1: Rocm-smi interface.

The V-F settings' control depends on the current device underuse due to different implementations of their DVFS system. However, `perf` configuration and device power cap have to be changed regardless of the implementation of the DVFS system.

The following enumeration describes the steps to be made regardless of the implementation of the DVFS system, stressing the reasons for the same. The following sections detail the working and tuning of the V-F pair for each of the tested GPUs.

1. **Change device performance level:** Disable automatic DVFS system.

```
rocm-smi --setperflevel manual
```

2. **Get device maximum power cap:** Get maximum allowed power consumption (PowerCap) allowed by the device.

```
rocm-smi --showmaxpower
```

3. **Change device power cap:** Change the current device maximum power consumption to the value obtained on the previous command.

```
rocm-smi --setpoweroverdrive X --autorespond yes
```

It is necessary to change the default PowerCap value to fully unlock manual tuning of the DVFS system. If not changed, the device will not allow sustained configuration of the highest frequency and voltage levels.

A.1 AMD Vega 10 Frontier Edition

The Vega 10 GPU presents the concept of discrete performance levels, containing eight possible configurations for the core DVFS domain. When editing the frequency and voltage values of each performance level, three rules must be followed. If that is not the case, even though the system accepts the configuration, the `perf` configuration (performance level) switches to automatic. The rules are

1. the desired frequency and voltage values must be within the valid ranges for the target device;
2. the desired frequency and voltage values cannot have the same value as the default for the given performance level;
3. the selected frequency and voltage values of the performance level i need to be at least 1 unit (1 MHz or 1 mV) above the performance level $i - 1$ and one unit below the performance level $i + 1$.

To guarantee that the desired V-F pair is selected, it is necessary to write on performance level 0 to 6 V-F pairs below all configurations that will be used and apply on the performance level 7 the desired V-F pair. In this way, even some automatic control systems that may remain active are forced to select the wanted configuration. Table A.1 presents the pre-defined voltage values for the performance levels.

Level	Frequency [MHz]	Voltage [mV]
0	853	801
1	860	810
2	870	820
3	880	830
4	890	840
5	900	850
6	910	860
7	Desired Frequency	Desired Voltage

Table A.1: Vega 10 GPU core performance levels general configuration.

The following enumeration indicates the `rocm-smi` commands that should be executed to select the desired voltage-frequency configuration.

4. **Change performance levels 0 to 6 to pre defined values:**

```
rocm-smi --setslevel 0 853 801 --autorespond yes
```

```

rocm-smi --setslevel 1 860 810 --autorespond yes
rocm-smi --setslevel 2 870 820 --autorespond yes
rocm-smi --setslevel 3 880 830 --autorespond yes
rocm-smi --setslevel 4 890 840 --autorespond yes
rocm-smi --setslevel 5 900 850 --autorespond yes
rocm-smi --setslevel 6 910 860 --autorespond yes

```

5. Change performance levels 7 to desired values:

```
rocm-smi --setslevel 7 $frequency $voltage --autorespond yes
```

6. Select core performance level 7:

```
rocm-smi --setsclk 7 --autorespond yes
```

7. Confirm the current voltage level: To note that this command shows the voltage value that corresponds to $\max(V_{Core}, V_{DRAM})$, so, if a voltage value is selected on the core domain is smaller than the current applied on the DRAM domain, the tool will report the voltage value of the DRAM domain.

```
rocm-smi --showvoltage
```

A.2 AMD Radeon 5700 XT

The Radeon 5700 XT DVFS system does not present the concept of performance levels. Instead, three V-F pairs indicated, and a quadratic regression is made to create the function $V(f)$ (chart presented in Chapter 2 Figure 2.8). Similarly to the Vega 10 GPU, the three V-F pairs also need to respect the three indicated rules.

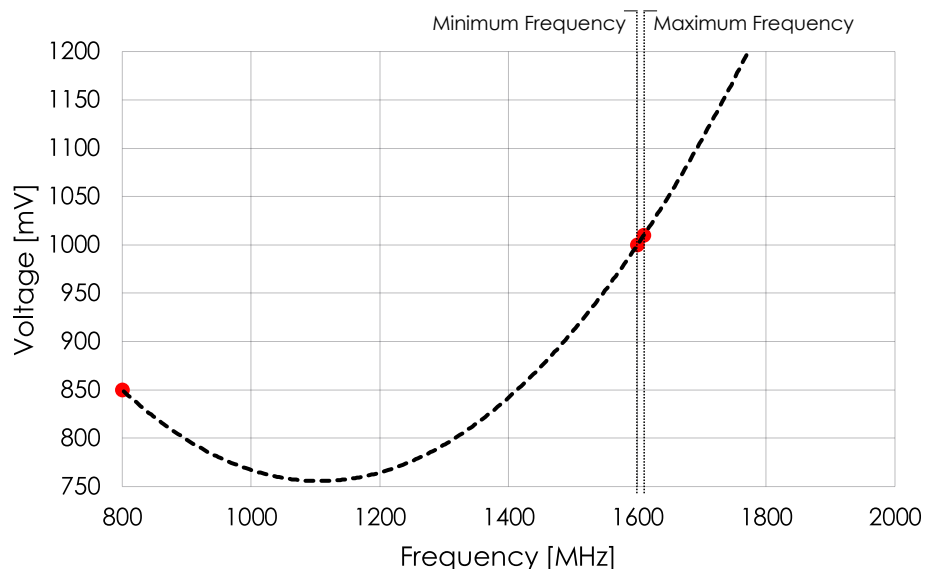


Figure A.2: Change in voltage curve to select a given V-F pair.

Overall, to be able to select and apply a specific V-F pair, the rocm-smi interface presents one other command that indicates to the DVFS system, which is the minimum and maximum frequency that should be used. Taking advantage of that interface, the devised method to select a given frequency is presented

in Figure A.2, where the first V-F pair is not changed, and the remaining two are set following Table A.2. The value of +10 is the pre-defined allowed variation of the parameters and was decided taking into account the range of frequency and voltage values.

V-F pair number	Frequency [MHz]	Voltage [mV]
0	800	850
1	Desired Frequency	Desired Voltage
2	Desired Frequency + 10	Desired Voltage + 10

Table A.2: Radeon 5700 XT GPU core V-F pairs general configuration.

The following enumeration indicates the rocm-smi commands that should be executed to select the desired voltage-frequency configuration.

4. Change the V-F pair values:

```
rocm-smi --setvc 0 800 850 --autorespond yes
rocm-smi --setvc 1 $frequency $voltage --autorespond yes
rocm-smi --setvc 2 ${frequency + 10} ${voltage + 10} --autorespond yes
```

5. Change allowed frequency range limits:

```
rocm-smi --setsrange 0 $frequency --autorespond yes
rocm-smi --setsrange 1 ${frequency + 10} --autorespond yes
```

6. Confirm the current voltage level: To note that this command shows the voltage value that corresponds to $\max(V_{Core}, V_{DRAM})$, so, if a voltage value is selected on the core domain is smaller than the current applied on the DRAM domain, the tool will report the voltage value of the DRAM domain.

```
rocm-smi --showvoltage
```