Performance and Energy Consumption Evaluation of the VVC ALF Classification Step Across Different Programming Paradigms and Hardware Platforms

Vitória Fabricio, Iago Storch, Guilherme Corrêa, Daniel Palomino
vitoria.sfabricio@gmail.com,{icstorch,gcorrea,dpalomino}@inf.ufpel.edu.br
Video Technology Research Group
Federal University of Pelotas, Brazil

ABSTRACT

The Adaptive Loop Filter (ALF) in the Versatile Video Coding (VVC) standard enhances visual quality but introduces a significant computational burden, posing a challenge for its implementation on resource-constrained devices. This paper presents a comprehensive evaluation of the ALF classification step, analyzing its execution time and energy consumption across multiple programming paradigms (scalar, SIMD, CUDA) on both a high-performance desktop and an embedded platform. Results show that on the high-performance desktop, SIMD optimization provides a substantial speedup over the scalar baseline with lower energy consumption, while the discrete GPU's performance is limited by data transfer overhead. Conversely, the embedded system demonstrates a different landscape: its ARM CPU offers superior energy efficiency when compared to the desktop, and its integrated GPU's runtime is dominated by kernel execution rather than data transfers. These findings underscore that the optimal ALF implementation is highly platform-dependent, hinging on the specific design trade-offs between processing speed and energy efficiency.

KEYWORDS

Versatile Video Coding, Adaptive Loop Filter, Embedded Systems

1 INTRODUCTION

Achieving efficient video compression has always demanded significant computational resources [19]. In response to the continuous demand for higher performance, standards like Versatile Video Coding (VVC) have been developed. VVC integrates novel tools to deliver superior compression gains when compared to its predecessors [4]. An innovation within VVC is the Adaptive Loop Filter (ALF), which offers considerable improvements in coding efficiency, but these benefits come with the trade-off of a higher computational workload during encoding [12]. This high computational demand translates not only into long processing times but also into significant energy consumption, a critical concern for battery-powered mobile devices and resource-constrained embedded systems [15].

The ALF operates by applying a spatial filter to the reconstructed samples with the goal of reducing artifacts introduced during encoding. This process involves classifying each block of the image based on its textural properties and using such a classification to derive filter coefficients to optimize the visual quality [8].

Although ALF was initially conceived for the High Efficiency Video Coding (HEVC) standard, it was not adopted then due to its large computational requirements. With the increasing necessity of coding efficiency and the availability of better hardware, the ALF was ultimately integrated into the VVC standard. Although VVC has multiple complex encoding tools, the ALF procedure still represents a significant portion of the overall processing time. Hence, its acceleration remains an open challenge. To address this challenge, several researchers have proposed optimization methods: Fang *et al.* [7] proposed an optimization algorithm that reduces the number of filter sets and accelerates the ALF procedure, while Yin *et al.* [21] proposed an algorithm that simplifies block classification and employs early terminations to reduce encoding complexity. Although these works attenuate the impacts of ALF in encoding time, they consider algorithmic changes in the ALF procedure under a fixed programming paradigm. However, the massively parallel computing paradigm through GPUs is gaining popularity, with CPU+GPU computing systems becoming increasingly available [18].

The trend of using GPUs for general-purpose computing [11, 13, 20] is a result of their highly parallel architecture [17]. This adoption is supported by mature programming interfaces like CUDA [14] and OpenCL [1] that expose their computational capabilities. The ALF tool, which consists of applying filters to reconstructed blocks independently, exhibits significant data parallelism that aligns well with the execution model of GPUs.

A preliminary study on this matter [6] presented an evaluation of the processing time of the ALF classification algorithm across three programming paradigms on a desktop platform: a sequential scalar implementation on the CPU, a SIMD-vectorized version on the same CPU, and a massively parallel implementation on a discrete GPU. Although [6] provided insights into the processing time trade-offs between programming paradigms on a desktop, the performance under embedded systems and the overall energy efficiency were not addressed. Conversely, [16] also proposed a GPU-based parallelization of the ALF filter on a resource-constrained platform. However, their implementation was based on the VVdeC software decoder, whereas the present work utilizes the VTM reference software.

Hence, this work builds upon our initial study [6] and expands the scope in two dimensions. First, the focus is moved beyond time-performance to include a detailed analysis of energy consumption. Second, the evaluation is extended to multiple hardware platforms, contrasting a high-performance desktop system with a power-efficient embedded platform, the NVIDIA Jetson TX2. By analyzing the interplay between performance and energy across

In: Proceedings of the Brazilian Symposium on Multimedia and the Web (WebMedia'2025). Rio de Janeiro, Brazil. Porto Alegre: Brazilian Computer Society, 2025.
© 2025 SBC – Brazilian Computing Society.
ISSN 2966-2753

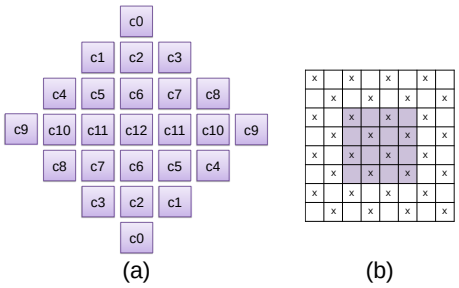

Figure 1: Diamond Shape filters for (a) Filter shape of Luma component (b) subsampling pattern used for classification

these paradigms and platforms, this work aims to provide a detailed characterization of the ALF classification step.

2 ADAPTIVE LOOP FILTER IN VVC

According to VVC, each frame is initially divided into Coding Tree Units (CTUs), which are typically 128×128 blocks. These CTUs are then recursively subdivided into smaller Coding Units (CUs) that can reach a minimum size of 4×4 samples [5]. This block-based processing, combined with the quantization stage, introduce visual distortions known as coding artifacts [12]. VVC specifies three in-loop filters to attenuate these artifacts: Deblocking Filter (DBF), Sample Adaptive Offset (SAO), and Adaptive Loop Filter (ALF) [12].

The ALF procedure employes Wiener-based filters to minimize prediction error. It begins by partitioning each reconstructed CTU into 4×4 blocks. Each 4×4 block is classified according to its texture properties, and then, this class is used to derive the filtering coefficients for the respective block. The Luma component is filtered with diamond-shaped 7×7 filters, as illustrated in Fig. 1. The Luma filter coefficients $c_0, c_1, \ldots, c_{12}$ can be selected from a set of predefined filters or customized according to the video content [12]. The Chroma component can be filtered with its own set of coefficients, or it can be derived from the collocated Luma filter [8]. This work focuses on the Luma component due to its primary impact on the encoding process. Furthermore, the scope is set to the classification step within ALF to ensure consistency with the initial study [6], which allows for a comparison between performance and energy. Hence, other facets of the ALF procedure and the Chroma filtering are not covered in this work.

The classification of a 4×4 block results in one out of 25 available classes, which is shared by all samples of the block. The class index C is calculated according to (1). C is a function of the block's texture direction (D) and a quantized sample activity ($\tilde{A}$), which is derived from the original activity A.

$$C = 5D + \tilde{A} \tag{1}$$

Both D and $\tilde{A}$ are computed based on the frame's texture gradient, which is determined using a 1-D Laplacian operator. Therefore, the initial step consists of computing the gradient sums in four directions: horizontal (g_h), vertical (g_v), main diagonal (g_{d1}), and secondary diagonal (g_{d2}), as defined by equations (2)-(5).

Algorithm 1: ALF Direction Decision Process

```
if (g_hv^max ≤ t1 · g_hv^min) and (g_d^max ≤ t2 · g_d^min) then
    D ← 0;
else
    if (g_hv^max / g_hv^min) > (g_d^max / g_d^min) then
        if g_hv^max > t2 · g_hv^min then
            D ← 2;
        else
            D ← 1;
    else
        if g_d^max > t2 · g_d^min then
            D ← 4;
        else
            D ← 3;
```

$$g_h = \sum_{k=i}^{i+5} \sum_{l=j-\min Y}^{j+\max Y} |R_{k,l-1} - 2R_{k,l} + R_{k,l+1}| \tag{2}$$

$$g_v = \sum_{k=i-2}^{i+2} \sum_{l=j}^{j+\max Y} |R_{k-1,l} - 2R_{k,l} + R_{k+1,l}| \tag{3}$$

$$g_{d1} = \sum_{k=i}^{i+5} \sum_{l=j-\min Y}^{j+\max Y} |R_{k-1,l-1} - 2R_{k,l} + R_{k+1,l+1}| \tag{4}$$

$$g_{d2} = \sum_{k=i}^{i+5} \sum_{l=j-\min Y}^{j+\max Y} |R_{k-1,l+1} - 2R_{k,l} + R_{k+1,l-1}| \tag{5}$$

In this context, (i, j) represents the coordinates of the upper-left sample of a given 4×4 block, and $R_{k,l}$ is the reconstructed sample at position (k, l). The calculation considers a two-sample halo defined by $maxY = 5$ and $minY = 2$. To reduce the computational complexity, VVC specifies that the gradients must be computed in a subsampled fashion, as depicted in Fig. 1. The gradient only considers the samples at locations (k, l) where k and l are either both even or both odd (samples denoted by the × in Fig. 1).

The maximum and minimum gradient values are identified within two categories: horizontal/vertical and diagonal gradients. This process yields $g_{hv}^{\max}$ and $g_{hv}^{\min}$ for the horizontal/vertical directions, and $g_d^{\max}$ and $g_d^{\min}$ for the diagonal directions. These gradient extrema are inputs to the procedure detailed in Algorithm 1, which calculates the block's texture direction, $D \in [0, 1, 2, 3, 4]$. The thresholds $t_1 = 2$ and $t_2 = 9$ are specified by the VVC standard[9, 10].

Then, the local sample activity, A is computed as defined by (6). Here, (i, j) denotes the coordinates of the top-left corner of the 4×4 block. Note that the calculation iterates over an 8×8 window, which encompasses the 4×4 block plus a two-sample halo around it (i.e., k ranges from $i-2$ to $i+5$, while l ranges from $j-2$ to $j+5$). $V_{k,l}$ and $H_{k,l}$ represent the vertical and horizontal gradients, respectively, while BD refers to the bit-depth of the image. Finally, $\tilde{A}$ is derived by mapping A to an integer between 0 and 4, according to (7).

$$A = \left(\sum_{k=i-2}^{i+5} \sum_{l=j-2}^{j+5} \left(V_{k,l} + H_{k,l} \right) \right) \gg (BD - 2) \tag{6}$$

$$\tilde{A} = Q_{min}(A, 15), \quad \text{such that,}$$
$$\{Q_n\} = \{0, 1, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 4\} \tag{7}$$

The ALF classification step is inherently parallel, as its core tasks, computing gradients and classifying 4×4 blocks, have minimal data dependencies. Consequently, a comprehensive evaluation of different programming paradigms is essential, analyzing not only processing time but also energy consumption. This dual analysis is critical for developing future ALF acceleration solutions that are both fast and energy-efficient.

3 METHODOLOGY

The objective of this work is to present a comprehensive analysis of the execution time and energy consumption of the ALF classification stage. To this end, ALF is implemented and evaluated across distinct hardware platforms and programming paradigms, ranging from sequential processing to massive parallelism. The following sections detail these implementations, offering a comparative assessment to identify the trade-offs between different approaches. The VVC standard is supported by a reference software implementation, the VTM encoder [2], which includes all encoding tools available in the standard – including ALF.

This work uses the VTM implementation to establish a performance baseline across two distinct hardware systems. The first is a conventional desktop platform, equipped with an x86 CPU and a discrete desktop GPU. The second is an ARM-based NVIDIA Jetson TX2 embedded board, which features an integrated GPU alongside its ARM CPU. The VTM encoder provides both scalar and SIMD-accelerated code paths for the CPU; however, the SIMD version is only available for x86 architectures and ARM is not supported.

On the x86 desktop CPU, two paradigms are considered. The first, named $\mathbf{VTM_{scalar}}$, employs a loop-based, sequential methodology. The second paradigm, $\mathbf{VTM_{SIMD}}$, leverages specialized SIMD instructions to accelerate gradient computation by processing multiple samples concurrently. Since the VTM's SIMD implementation does not support the ARM architecture of Jetson TX2, only the scalar implementation ($\mathbf{VTM_{scalar}}$) is considered for this platform.

Finally, the third paradigm explores the massive parallelism afforded by GPUs. For this paradigm, a custom CUDA implementation is developed. The CPU-based methods are discussed in Section 3.2, while the GPU approach is described in Section 3.3. Importantly, all implementations produce identical outcomes and do not alter the final coding efficiency.

3.1 Evaluation Setup

All CPU encodings employ the VTM encoder. To ensure fair comparisons, the tests adhere to the Common Test Conditions (CTC) [3]. The CTCs provide a standardized framework for evaluating VVC implementations, defining test sequences, encoder configurations (All Intra, in this case), and Quantization Parameters (QPs). For this study, the first 32 frames of a series of video sequences are processed using the QPs recommended by the CTC: 22, 27, 32, and 37. Sequences with resolutions of 480p, 720p, 1080p, and 4K are evaluated. Exceptionally, the CPU encoding on Jetson TX2 is limited to 480p and 720p sequences due to the embedded system's hardware constraints.

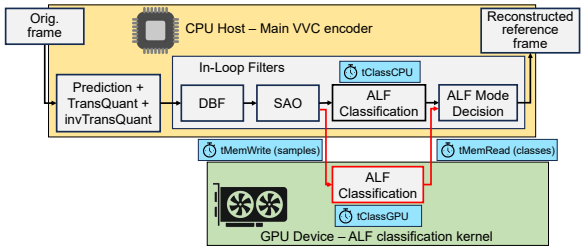

Figure 2: CPU+GPU encoding system proposed in [6]. Red outlines depict the GPU bypass used for ALF classification.

The desktop system is equipped with an Intel Core i7-8700 CPU (featuring the AVX2 SIMD instruction set), 16 GiB of RAM, and an NVIDIA RTX 3080 discrete GPU (Driver 555.42.06, CUDA 12.5), running on Ubuntu 20.04. The embedded NVIDIA Jetson TX2 board counts with an ARM Cortex-A57 CPU and an integrated 256-core Pascal GPU, operating on Ubuntu 18.04 with CUDA Toolkit 10.2.

An overview of the generic VVC encoding pipeline is provided in Fig. 2. The process begins with frame prediction, followed by the transform and quantization (TransQuant) stages. The frame is subsequently reconstructed via inverse transform and quantization (invTransQuant), after which in-loop filters are applied to mitigate compression artifacts. The diagram illustrates the data flow for different paradigms: black arrows denote the path for CPU-only processing, while red arrows depict the data transfers between the CPU and GPU required for the GPU-based implementation. For the CPU-based approaches ($\mathbf{VTM_{scalar}}$ and $\mathbf{VTM_{SIMD}}$), the measured time corresponds solely to the ALF classification stage (**tClassCPU**). For the GPU-based approach, the total time includes not only the kernel execution (**tClassGPU**) but also the data transfer overhead for moving frame samples to the GPU (**tMemWrite**) and returning the classification results to the CPU (**tMemRead**).

Energy is measured using platform-specific hardware monitors. RAPL (Running Average Power Limit) is used for the desktop CPU, the NVIDIA NVML library is used for the desktop discrete GPU, while for the case of Jetson TX2, the on-board INA3221 power monitor ICs are polled through their respective 'sysfs' entries.

3.2 Scalar and SIMD Implementations

For many tools, the VTM provides both scalar and SIMD-accelerated code paths targeting different architectures. This work utilizes these existing, publicly available implementations to benchmark the performance of CPU-based ALF classification.

The first baseline, designated $\mathbf{VTM_{scalar}}$, employs a loop-based, sequential methodology. In this version, each CTU is processed in order, and within a given CTU, its 4×4 blocks are also classified sequentially. The gradient is calculated once for every 32×32 region of the CTU and then used by the corresponding 4×4 blocks, with the computation itself proceeding one position at a time.

The second baseline, named $\mathbf{VTM_{SIMD}}$, also follows a sequential, loop-based approach to process each 4×4 block. The fundamental difference is its use of specialized SIMD instructions (for instance,

`_mm256_set1_epi32` and `_mm_lddqu_si128`) to compute the gradient. This allows multiple samples to be processed concurrently, accelerating the calculation.

3.3 Massively Parallel Programming

To assess the performance of a massively parallel approach, a GPU-based solution was developed using the NVIDIA CUDA platform [14]. This solution is divided into two kernels: **computeGradient** concurrently calculates the gradient for all frame samples, and **classifyBlock** concurrently classifies all 4×4 blocks.

The hybrid CPU-GPU processing pipeline is depicted in Fig. 2, where red arrows indicate data transfers. The CPU is responsible for initial encoding stages such as prediction, transform, and quantization. For ALF classification, the host CPU offloads the reconstructed frame to GPU memory. Once the GPU kernels execute and determine the class for each 4×4 block, the results are transferred back to the CPU for subsequent in-loop filtering stages.

The **computeGradient** kernel assigns one thread block of 1024 threads to each 32×32 frame region. To handle border dependencies, a 34×34 area of reconstructed samples (the 32×32 region plus a one-sample halo) is first loaded from global memory into the block's shared memory in a coalesced manner. After a thread synchronization barrier, each thread independently computes the gradient for a single sample, leveraging the low-latency shared memory. This strategy ensures an even workload distribution and minimizes global memory bandwidth by maximizing data reuse. The resulting gradients are then written to global memory.

The **classifyBlock** kernel is employed for the classification task, where each thread block, consisting of 16 threads, is assigned to process a 4×4 block of samples. Initially, each thread loads the gradient of one sample into shared memory, with memory accesses arranged to be efficient. After all threads in the block synchronize, a single designated thread performs the classification. This is achieved by first aggregating the gradients in a subsampled fashion, then determining their minimum and maximum values, and finally computing the result using Equation (1). The final classification for each 4×4 block is then copied from the GPU to the host memory.

4 RESULTS

The performance results, detailing execution time and energy consumption for each platform and implementation, are presented in Table 1. Here, **Comm. share** and **Exec. share** represent the share of the processing time consumed by CPU-GPU communication and kernel execution, respectively, for the GPU implementations. Analyzing Table 1 reveals a performance hierarchy on the desktop system. The implementation $\mathbf{VTM_{SIMD}}$ is the fastest, followed by the Desktop GPU (NVIDIA RTX 3080), and finally the baseline $\mathbf{VTM_{Scalar}}$ implementation. As expected, the overall performance of the embedded Jetson TX2 system is slower than the desktop counterpart. A characteristic of the embedded platform is that its ARM CPU is faster than its integrated GPU for this specific task, even though only $\mathbf{VTM_{Scalar}}$ is considered.

The results for the CPU desktop implementations are unfavorable for the GPU desktop due to significant communication overhead. For instance, when processing a 1080p video, the data transfer between the CPU and the GPU accounts for **86%** of the total time

Table 1: Time and energy consumption results, alongside the time share required for communication in GPUs.

Device	Metrics	480p	720p	1080p	4k
Desktop CPU Scalar Intel i7-8700	Time [ms]	1.600	3.950	8.950	35.97
	Energy [J]	0.032	0.064	0.138	0.523
Desktop CPU SIMD Intel i7-8700	Time [ms]	0.300	0.930	1.900	8.525
	Energy [J]	0.020	0.021	0.042	0.176
Desktop GPU NVIDIA RTX 3080	Time [ms]	1.305	3.244	6.512	27.40
	Energy [J]	0.043	0.106	0.051	0.899
	Comm. share	82%	84%	86%	89%
	Exec. share	18%	16%	14%	11%
Jetson TX2 CPU ARM Cortex A57	Time [ms]	5.300	12.00	N/A	N/A
	Energy [J]	0.006	0.013	N/A	N/A
Jetson TX2 GPU NVIDIA 256 cores	Time [ms]	45.42	73.25	222.1	277.2
	Energy [J]	0.007	0.011	0.059	0.076
	Comm. share	34%	18%	20%	28%
	Exec. share	66%	82%	80%	72%

(Comm. share). This indicates that the bottleneck is not the GPU's computational power but rather the time required to communcate with the CPU. On the other hand, the Jetson TX2's integrated GPU spends a smaller share of the processing time with communication. This is because, as an integrated GPU (iGPU), it shares main memory with the CPU, eliminating the need for data transfers over a slower PCIe bus. For the same 1080p resolution, the communication share on the Jetson GPU is only **20%**, with the actual kernel execution accounting for the remaining **80%** of the time.

Although the Desktop GPU is hampered by data transfer overhead, it still outperforms the baseline $\mathbf{VTM_{Scalar}}$ implementation. This can be attributed to its massively parallel architecture, featuring **3584 CUDA cores**, which stands in stark contrast to the Jetson TX2's integrated GPU with only **256 CUDA cores**.

While the Jetson platform is slower, its main advantage is its remarkable energy efficiency. When processing at 720p, the Jetson TX2 CPU consumes approximately **38%** less energy than the most efficient desktop implementation ($\mathbf{VTM_{SIMD}}$). Under the same conditions, the Jetson's GPU is even more efficient, consuming about **48%** less energy. This highlights the trade-off between raw performance and power efficiency, positioning the embedded system as a superior choice for energy-constrained applications.

5 CONCLUSION

This paper presented a performance and energy evaluation of the VVC ALF classification step across diverse hardware platforms. The analysis revealed that on a high-performance desktop, an optimized CPU implementation using $\mathbf{VTM_{SIMD}}$ delivers the highest processing speed, outperforming a discrete GPU. It was demonstrated that the GPU's potential is severely constrained by data communication overhead, which becomes the primary performance bottleneck. In contrast, the embedded Jetson TX2 platform, while slower, offers superior energy efficiency, with its integrated GPU being limited by kernel execution time rather than data transfers. These findings underscore that the optimal implementation strategy for the ALF filter is not universal but is highly platform-dependent, and the trade-off between raw performance and energy consumption must be taken into account based on the application's constraints.

REFERENCES

[1] 2023. *The OpenCL Specification, Version 3.0.14.* Doc. Khronos Group.

[2] 2023. *VTM VVC Reference Software.* https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM

[3] Frank Bossen, Jill Boyce, Karsten Suehring, Xiang Li, and Vadim Seregin. 2020. *JVET-T2010: VTM common test conditions and software reference configurations for SDR video.* Doc. JVET.

[4] Frank Bossen, Karsten Sühring, Adam Wieckowski, and Shan Liu. 2021. VVC Complexity and Software Implementation Analysis. *IEEE Transactions on Circuits and Systems for Video Technology* 31, 10 (2021), 3765–3778. https://doi.org/10.1109/TCSVT.2021.3072204

[5] Benjamin Bross, Ye-Kui Wang, Yan Ye, Shan Liu, Jianle Chen, Gary J. Sullivan, and Jens-Rainer Ohm. 2021. Overview of the Versatile Video Coding (VVC) Standard and its Applications. *IEEE Transactions on Circuits and Systems for Video Technology* 31, 10 (2021), 3736–3764. https://doi.org/10.1109/TCSVT.2021.3101953

[6] Vitória Fabricio, Iago Storch, and Daniel Palomino. 2025. Processing Time Evaluation of the Classification Step in the Adaptive Loop Filter of VVC under Multiple Programming Paradigms. In *2025 IEEE 16th Latin America Symposium on Circuits and Systems (LASCAS)*, Vol. 1. 1–5. https://doi.org/10.1109/LASCAS64004.2025.10966364

[7] Jiayue Fang and Fuzheng Yang. 2023. The Optimization of Adaptive Loop Filter Based on the Reduction of the Filter Sets. In *2023 International Conference on Ubiquitous Communication (Ucom).* 166–170. https://doi.org/10.1109/Ucom59132.2023.10257583

[8] Ibrahim Farhat, Wassim Hamidouche, Adrien Grill, Daniel Menard, and Olivier Deforges. 2022. Efficient HW Design of Adaptive Loop Filter for 4k ASIC VVC Encoder. In *2022 Picture Coding Symposium (PCS).* 1–5. https://doi.org/10.1109/PCS56426.2022.10018078

[9] Ibrahim Farhat, Wassim Hamidouche, Adrien Grill, Daniel Ménard, and Olivier Déforges. 2022. Adaptive Loop Filter Hardware Design for 4K ASIC VVC Decoders. *IEEE Transactions on Consumer Electronics* 68, 2 (2022), 107–118. https://doi.org/10.1109/TCE.2022.3146272

[10] ITU-T. 2023. *Recommendation H.266: Versatile Video Coding.* Doc. ITU-T.

[11] Sainathan Ganesh Iyer and Anurag Dipakumar Pawar. 2018. GPU and CPU accelerated mining of cryptocurrencies and their financial analysis. In *2018 2nd International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC) I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC), 2018 2nd International Conference on.* IEEE, 599–604.

[12] Marta Karczewicz, Nan Hu, Jonathan Taquet, Ching-Yeh Chen, Kiran Misra, Kenneth Andersson, Peng Yin, Taoran Lu, Edouard François, and Jie Chen. 2021. VVC In-Loop Filters. *IEEE Transactions on Circuits and Systems for Video Technology* 31, 10 (2021), 3907–3925. https://doi.org/10.1109/TCSVT.2021.3072297

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2017. Imagenet classification with deep convolutional neural networks. *Commun. ACM* 60, 6 (2017), 84–90.

[14] NVIDIA. 2022. *CUDA C++ Programming Guide.* Doc. NVIDIA.

[15] Anup Saha, Wassim Hamidouche, Miguel Chavarr'ıas, Fernando Pescador, and Ibrahim Farhat. 2023. Performance analysis of optimized versatile video coding software decoders on embedded platforms. *Journal of Real-Time Image Processing* 20, 6 (2023), 120.

[16] Anup Saha, Nuno Roma, Miguel Chavarr'ıas, Tiago Dias, Fernando Pescador, and V'ıctor Aranda. 2023. GPU-based parallelisation of a versatile video coding adaptive loop filter in resource-constrained heterogeneous embedded platform. *Journal of Real-Time Image Processing* 20, 3 (2023), 43.

[17] Iago Storch, Daniel Palomino, and Sergio Bampi. 2022. GPU-Acceleration of Affine Prediction in the Versatile Video Coding. In *2022 IEEE International Symposium on Circuits and Systems (ISCAS).* 429–433. https://doi.org/10.1109/ISCAS48785.2022.9937704

[18] Iago Storch, Nuno Roma, Daniel Palomino, and Sergio Bampi. 2023. GPU Acceleration of MIP Intra Prediction in VVC. In *2023 31st European Signal Processing Conference (EUSIPCO).* 600–604. https://doi.org/10.23919/EUSIPCO58844.2023.10290037

[19] Iago Storch, Nuno Roma, Daniel Palomino, and Sergio Bampi. 2025. Improving Coding Efficiency of Massive Parallel Intra Prediction Using Alternative References. *IEEE Transactions on Circuits and Systems I: Regular Papers* (2025), 1–14. https://doi.org/10.1109/TCSI.2025.3564455

[20] Chao Yang, Wei Xue, Haohuan Fu, Lin Gan, Linfeng Li, Yangtong Xu, Yutong Lu, Jiachang Sun, Guangwen Yang, and Weimin Zheng. 2013. A peta-scalable CPU-GPU algorithm for global atmospheric simulations. *ACM SIGPLAN Notices* 48, 8 (2013), 1–12.

[21] Wenbin Yin, Kai Zhang, and Li Zhang. 2023. Extended Adaptive Loop Filter Beyond VVC. In *2023 IEEE International Conference on Visual Communications and Image Processing (VCIP).* 1–5. https://doi.org/10.1109/VCIP59821.2023.10402795