

ARCHITECTURAL STRATEGIES TO ENHANCE THE LATENCY AND ENERGY EFFICIENCY OF MOBILE CONTINUOUS VISUAL LOCALIZATION SYSTEMS



UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Raúl Taranco Serna

Department of Computer Architecture
UNIVERSITAT POLITÈCNICA DE CATALUNYA

A thesis submitted for the degree of
DOCTOR OF PHILOSOPHY

Advisors: José-Maria Arnau and Antonio González

June, 2024

Barcelona, Spain

Abstract

The emergence of new applications such as autonomous machines (e.g., robots or self-driving cars) and XR (Extended Reality) promises to revolutionize how society interacts with technology in the rapidly advancing digital era. These technologies, deployed on edge devices, often rely on mobile or embedded SoCs (Systems-on-a-Chip) operating CV (Continuous Vision) pipelines that periodically capture and analyze environmental light.

A typical CV SoC comprises a frontend for image capture and a backend for processing vision algorithms. The frontend typically includes an off-chip camera sensor and a specialized SoC component, the ISP (Image Signal Processor), that processes the pixel stream, transforming raw sensor data into high-quality, color-balanced images. The backend (e.g., CPU (Central Processing Unit), GPU (Graphics Processing Unit), or an accelerator) processes the image stored in the main memory's framebuffer to extract perception insights and perform advanced decision-making.

Existing research identifies visual localization, object detection, and tracking as the primary bottlenecks in these emerging applications. Those algorithms face two principal challenges when deployed in mobile CV systems: latency and energy consumption. For example, an XR headset employs visual localization to track the user's head motion to correctly render the XR frame based on the estimated user's perspective. Latency can disrupt the quality of the user experience, leading to discomfort or disorientation. In a self-driving car, localization determines the vehicle's position with centimeter-level precision, unattainable through other means, such as GPS (Global Positioning System), to be later used for motion planning and actuation. A delay in the processing can compromise safety, aggravating as the vehicle's speed increases. Furthermore, energy consumption can severely restrict the operation of mobile battery-operated systems with limited autonomy.

This thesis embarks on a strategic journey to elevate mobile CV systems' performance

and energy efficiency.

We begin by analyzing a state-of-the-art visual localization engine on general-purpose CPU backends. The localization engine takes camera images as input, extracts interesting features such as landmarks, and tracks them to estimate the camera pose. Our evaluations show that the feature extraction phase constitutes the main application bottleneck, taking between 60% and 90% of the total localization latency.

Then, we explore the highly specialized hardware accelerator designs employed in the image processing domain. Our first thesis contribution introduces LOCATOR (Low-power ORB aCcelerator for AuTonomOus caRs), a high-performance, energy-efficient, and functionally accurate hardware accelerator tailored for detecting the ORB (Oriented FAST and Rotated BRIEF) features prevalent in many localization frameworks. Their extraction involves two stages: feature detection and feature description. LOCATOR processes image tiles on the fly using two parallel synchronized pipelines, one for each stage. The feature extraction pipeline can steadily process one pixel per cycle, achieving peak performance. On the contrary, the description generation stage requires hundreds of independent accesses to on-chip buffers for each detected feature, stalling the pipeline and severely impacting overall performance.

LOCATOR ameliorates pipeline stalls by combining several techniques to reduce the description generation latency. We employ a genetic algorithm to establish an optimal static bank access pattern, integrate a caching mechanism for recurrent accesses, and selectively replicate ports based on the overall rBRIEF (Rotated BRIEF) pattern characteristics. These methods collectively address the core bottlenecks of the baseline accelerator without significant cost increases. As a result, LOCATOR significantly outperforms other specialized designs in energy efficiency and achieves $16.8\times$ speedup for ORB feature extraction, $1.9\times$ end-to-end speedup, and $2.1\times$ end-to-end energy reduction per frame, compared to running the complete localization system on the baseline mobile CPU.

Next, we realize that while specialized architectures are necessary for highly constrained mobile systems, there is a pressing need for more programmable and versatile solutions adaptable to the present and future evolving application needs. CPUs are positioned on the other side of the spectrum, offering high programmability at the cost of sacrificing performance and energy efficiency. Even those platforms' SIMD (Single Instruction, Multiple Data) VPUs (Vector Processing Units) miss critical optimization opportunities.

In the second contribution of the thesis, we identified convolutions and stencil oper-

ations as foundational primitives in image analysis, which typically utilize a sliding window dataflow. Thus, we target these operations to propose SLIDEX (SLIDing window EXtension for image processing), a domain-specific vector ISA (Instruction Set Architecture) extension to exploit SWP (Sliding Window Processing) in conventional CPUs. SWP reinterprets the layout of the vector registers not as a sequence of isolated elements but as a sequence of overlapped windows of elements. SLIDEX instructions leverage this model to perform multiple parallel computations sourced from the overlapping, shifted window copies of the elements within a vector register, maximizing the DLP (Data Level Parallelism) achievable per instruction while maintaining the same vector length. Furthermore, it significantly reduces the need for data access, movement, and alignment, decreasing memory and register file accesses compared to traditional SIMD designs. SLIDEX achieves significant speedups in vital tasks such as 2D convolutions for image filtering and stencil operations for feature extraction, leading to an overall speedup of $\sim 1.2\times$ and up to 19% energy reduction.

Subsequently, we broadened our research scope beyond isolated backends to leverage the unique inherent properties of vision input streams and propose novel SoC-level optimizations. The CV frontend captures successive similar images with gradual positional and orientational variations. Consequently, many regions between consecutive frames yield nearly identical results when processed in the backend. Despite this, current systems process every image region at the camera’s sampling rate, overlooking the fact that the actual rate of change in these regions could be significantly lower. In the third contribution of the thesis, we introduce δ LTA (*don’t Look Twice, it’s Alright*). This novel approach decouples camera frame sampling from backend processing by extending the camera with the ability to identify redundant image regions before they enter subsequent CV pipeline stages. δ LTA informs the backend about the image regions that have notably changed, allowing it to focus solely on processing these distinctive areas and reusing previous results to approximate the outcome for similar ones. As a result, the backend processes each image region using different processing rates based on its temporal variation. δ LTA removes a significant fraction of unneeded frontend and backend memory accesses and redundant backend computations, reducing the localization tail, average latency, and energy consumption by 7.2%, 15.2%, and 17%, respectively.

Finally, we notice that many frontend image manipulation byproducts, potentially valuable for later CV processing, are routinely discarded due to the isolated operation of the frontend and backend stages. This isolation forces the backend to process the image without prior insights in an arbitrary (e.g., raster-scan) order, regardless of the amount of

detail of the different image regions. In the last contribution of the thesis, we leverage these observations to propose IRIS (Image Region ISP-Software Prioritization), which optimizes CV by harnessing and repurposing existing computations executed by the frontend’s ISP. IRIS segments and ranks image regions based on their detail level and perceived motion. Our proposal enables computational backends to implement an iterative image-processing approach that focuses on the more relevant regions and obtains results earlier, potentially reducing latency and energy consumption and achieving zero-latency overhead even in the worst-case scenario. IRIS enables effective iterative visual localization, reducing tail, average latency, and energy consumption by 9%, 20%, and 16%, respectively.

Keywords

Continuous Vision Systems, Hardware Accelerator, SIMD, ISP, SLAM, Localization, Autonomous Driving, Augmented Reality, Extended Reality.

Acknowledgements

I want to express my sincere gratitude to my advisors, Prof. Antonio González and Dr. Jose María Arnau, for their pivotal roles throughout these years. Their patience, guidance, and unwavering belief in my potential have profoundly shaped my growth as a researcher. They have taught me how to conduct research at the highest standards. I am genuinely thankful for the invaluable opportunity to collaborate with them.

I extend my gratitude to my thesis committee members: Ramon Bevide, Cristina Barrado, and Vijay Janapa Reddi. Ramon Bevide was my professor during my undergraduate studies at Cantabria and unknowingly influenced my decision to pursue a PhD in Barcelona through his praise and stories about exceptional research in computer architecture at UPC during his courses. I am grateful for his participation in my committee as I complete this journey. I appreciate the insightful comments and suggestions of Cristina Barrado and Vijay Janapa Reddi. Special thanks to Vijay for inviting me to his amazing lab at Harvard.

I am deeply thankful to the various funding sources that supported my research: the CoCoUnit ERC Advanced Grant (grant No 833057) from the European Union's Horizon 2020 program; grants TIN2016-75344-R and PID2020-113172RB-I00 from the Spanish State Research Agency (MCIN/AEI); grant 2021SGR00383 from the Catalan Agency for University and Research (AGAUR); the ICREA Academia program; and the FPU grant FPU18/04413.

I thank my senior colleagues Reza, Albert, Martí, Andreas, and Dennis for their wise advice and inspiration. Special thanks to all members of the ARCO group, including professors Jordi, Juan Luis, Josep-Llorenç, and my labmates Mojtaba, Mohamad, Bahareh, Rodrigo, Aurora, Imad, and Nitesh, for their support and constructive feedback during our weekly meetings. Thanks also to Diya, who started the PhD journey with me, for all the good experiences and support before the deadlines. Special mentions go to Marc, Fran (a.k.a. Félix), and Pedro for their stimulating discussions and enjoyable coffee breaks. Pedro, thank

you for the countless hours spent discussing research ideas and life.

Heartfelt thanks to my friends for their support. I thoroughly enjoyed the various plans and social activities over the years that provided much-needed breaks from my research. Your patience during my busy periods is deeply appreciated—special thanks to Jesús, Rodri, and Emilio for their understanding and friendship.

I am grateful to my closest confidant and partner, Mariona, for her support, encouragement, and understanding during the final years of my PhD. Your virtuous values and encouragement have inspired me to bring out my best during this period.

Finally, my deepest gratitude goes to my family. My mother and brother have been pillars in my life, always believing in me and allowing me to choose my path. My mother taught me invaluable lessons on kindness, optimism, and enthusiasm despite life's challenges. I will always admire her dedication, hard work, and generosity. My brother always protected me and patiently listened to my long "monologues". He always helped me and offered me great insights about the world. I am immensely grateful for all the opportunities they have given me.

*This thesis is dedicated to my beloved mother and brother.
For their unwavering support, endless sacrifices, and unconditional love.*

Contents

1	Introduction	27
1.1	Thesis Motivation	27
1.2	Problem Statement, Objectives, and Contributions	29
1.2.1	Image Feature Extraction Acceleration	31
1.2.2	Sliding Window Support for Generic CPUs	33
1.2.3	Decouple Camera Sampling from Processing	35
1.2.4	Leverage Synergistic Frontend and Backend Cooperation	37
1.3	State-of-the-art in Mobile Continuous Vision	38
1.3.1	Visual Applications	39
1.3.2	Hardware Specialization	40
1.3.3	Image Signal Processors	40
1.3.4	Programmability and Architectural Flexibility	42
1.3.5	Exploiting Spatio-temporal Similarity	43
1.3.6	Frontend-Backend Collaboration	44
1.4	Thesis Organization	45
2	Background on Continuous Vision and Localization	47
2.1	Architecture of Modern Continuous Vision Systems	47
2.1.1	Camera Sensor	48
		11

2.1.2	Image Signal Processor	49
2.1.3	Computation Backends	53
2.2	Visual Localization Overview	53
2.2.1	Overview of ORB-SLAM	54
2.2.2	Automatic Map Initialization	55
2.2.3	Tracking	55
2.2.4	Local Mapping	56
2.2.5	Loop Closing	57
2.2.6	ORB Feature Extraction	57
3	Methodology	63
3.1	Baseline Overview	63
3.2	LOCATOR Specific Modeling and Evaluation	65
3.3	SLIDEX Specific Modeling and Evaluation	66
3.4	DLTA and IRIS Specific Modeling and Evaluation	66
3.5	Visual Localization Engine	67
3.6	Dataset	68
4	LOCATOR	69
4.1	Low-power Accelerator for ORB Feature Extraction	69
4.2	Implementation	71
4.2.1	Hardware Architecture Overview	71
4.2.2	Basic Sliding Window Structure	71
4.2.3	rBRIEF Unit	73
4.2.4	FAST Detector Unit	85
4.2.5	Non-Maximal Suppression Unit	86
4.2.6	Gauss Unit	86

4.2.7	Rotation Unit	86
4.3	Experimental Results	88
4.4	Conclusions	94
5	SLIDEX	95
5.1	A Novel Architecture for Sliding Window Processing	95
5.2	Implementation	97
5.2.1	Overview	97
5.2.2	Sliding Window Processing	98
5.2.3	Architectural State	99
5.2.4	Instruction Set Extension	100
5.2.5	SLIDEX Programability	100
5.2.6	SLIDEX Execution Unit	102
5.3	Experimental Results	107
5.3.1	Baseline Characterization	109
5.3.2	Performance Analysis	110
5.3.3	Energy, Power and Area Analysis	112
5.4	Conclusions	114
6	δLTA	117
6.1	Introduction	117
6.2	Decoupling Camera Sampling from Processing	119
6.3	Implementation	121
6.3.1	General Overview	121
6.3.2	Frontend	122
6.3.3	ISP Implementation	125
6.3.4	Backend	129

6.4	Experimental Results	132
6.4.1	Region Coverage and Accuracy Analysis	133
6.4.2	Latency Analysis	134
6.4.3	Energy Analysis	136
6.4.4	Frame Rate Scaling Analysis	136
6.4.5	Area and Power Analysis	137
6.5	Conclusions	137
7	IRIS	139
7.1	Unleashing ISP-Software Cooperation	139
7.2	Implementation	141
7.2.1	General Overview	141
7.2.2	Augmenting the Vision Frontend	143
7.2.3	Architectural Support	145
7.2.4	Augmenting the Vision Backend	148
7.3	Experimental Results	152
7.3.1	Region and Feature Coverage Analyses	153
7.3.2	Accuracy Analysis	154
7.3.3	Latency Analysis	155
7.3.4	Energy Analysis	157
7.3.5	Area and Power Analysis	157
7.4	Conclusions	157
8	Conclusions	159
8.1	Conclusions	159
8.2	Open-Research Areas	160

List of Tables

1.1	Energy consumption of different components in picojoules per pixel (adapted from [84]).	36
1.2	Comparison with previous works. PPW stands for Performance Per Watt.	41
3.1	Simulation parameters for the baseline platform.	65
3.2	Default parameters used in ORB-SLAM3.	67
4.1	Parameters used for the GA optimization of conflicts.	76
4.2	Hardware parameters for LOCATOR, H-CPU, and H-GPU platforms.	89
5.1	SLIDEX (SLIDing window EXtension for image processing) architectural visible state.	99
5.2	SLIDEX new proposed instructions.	100
5.3	Area and power for the baseline CPU and the different versions of SLIDEX.	113

List of Figures

1.1	High-level depiction of a Continuous Vision pipeline.	30
1.2	Feature extraction significantly dominates the execution time in visual localization systems.	32
1.3	Histogram illustrating the spatial distribution of features, showing the frequency of grid regions containing varying numbers of features.	38
2.1	Depiction of a CV pipeline, highlighting the front and backend that embeds a visual localization system.	48
2.2	Fundamental structure of an image sensor, featuring a standard Bayer photosensitive array configuration.	49
2.3	Traditional Image Signal Processor (ISP) pipeline.	50
2.4	Scheme of a typical 2D line buffer employed on image processing specialized hardware.	50
2.5	ORB-SLAM system overview [113]	54
2.6	Illustration of ORB feature extraction on a KITTI dataset image, with green points marking the detected features.	58
2.7	(2.7a) Pyramid of images generated from an original image. (2.7b) Bresenham circle of radius 3 showing the pixel access pattern for FAST around the pixel P	59
2.8	Example of the BRIEF (\mathcal{B}) operation to form the signature 01...0.	60

3.1	Illustration of our mobile CV (Continuous Vision) System on Chip (SoC), featuring an Image Signal Processor (ISP), a cluster of CPU, main memory, and an optional LOCATOR accelerator.	64
3.2	Methodology followed for designing and evaluating the RTL components. . .	66
4.1	The basic architecture of LOCATOR (Low-power ORB aCcelerator for AuTonomOus caRs).	72
4.2	7×7 sliding window structure.	72
4.3	rBRIEF (Rotated BRIEF) Unit architecture overview.	74
4.4	ORB (Oriented FAST and Rotated BRIEF) Window architecture that allows parallel access to multiple pairs (stored in the banks shown in dark grey) per cycle.	74
4.5	Example of convergence of the genetic algorithm to a local minimum for a <i>gsize</i> = 8.	77
4.6	The basic structure of the duplication cache (new cache banks shown filled with a line pattern) for a group size of 4 pairs. Note that only the first point of each pair PN_1 is represented.	79
4.7	rBRIEF pattern renaming example. The left table shows a segment of the original rBRIEF pattern where points with ID 0 and 4 are repeated. The assignation algorithm determined that the coordinates to store the value of the ID 0 point in the duplication cache are (0, 37). Thus, the static renaming changes the coordinates for the subsequent non-first accesses, such as ID 4, to point to these coordinates (right table). The renaming avoids points ID 4 and ID 5 conflict in the original pattern (marked in red).	80
4.8	(a) Percentage of total conflicts (Y-axis) generated for each bank on average for all angles. The X-axis shows the index of the banks in the range [-18, 17]. The bank with an index of zero is the central bank. (b) The slowdown of the average access latency of a replica when increasing the number of external single-ported banks. The parallel access group had a size of 4 pairs in this experiment.	82
4.9	The proposed stages and pipeline mechanism of each replica of the rBRIEF. The FIFOs between the stages have a length of 2.	83

4.10	(a) Sensitivity to the length of the FIFOs in the replica pipelining stages of the rBRIEF unit for a group size of 4 (G4) and 8 (G8) pairs processed in parallel. (b) The pipelined architecture of the Rotation Unit.	84
4.11	Architecture of the FAST (Features from Accelerated Segment Test) unit.	85
4.12	(a) rBRIEF replica speedup respect to the baseline implementation ($G1$). GN refers to the maximum number of pairs accessed in parallel. (b) Box plot representing the distribution of LOCATOR processing latencies (Cycles Per Pixel) of the frames of KITTI sequences for different versions of the accelerator.	90
4.13	(a) 99th Percentile Latency measured in Cycles Per Pixel (CPP) with all the optimizations proposed applied when processing the KITTI dataset. (b) LOCATOR speedup in the worst-case scenario compared to the baseline. The red dotted line indicates the minimum speedup threshold needed to reach our real-time consideration.	91
4.14	Average number of penalty cycles due to bank conflicts when computing an rBRIEF descriptor. The results correspond to the latency incurred when accessing the baseline implementation with and without the GA optimization.	92
4.15	(a) speedup achieved by the accelerator compared with the BASELINE-CPU. (b) Power dissipation of ORB FE (Feature Extraction and Matching) across different platforms and the accelerator.	93
5.1	Different execution models utilizing registers R_m and R_a for computing output in register R_d . SLIDEX (SLIDing window EXtension for image processing) instructions process multiple windows in parallel to compute the neighborhood operation (e.g., convolution) reduced results.	96
5.2	A high-level system overview. The SLIDEX unit is a pipeline placed inside the Execution stage of the in-order CPU.	98
5.3	General overview of the SLIDEX augmentations showing the state (VRF, PRF) visible to the programmer (Table 5.1) and the SLIDEX Execution unit. ROT modules rotate the input in the direction of the arrow.	103
5.4	$WCONV$ V0, VZR, V0, V15 simplified cycle chronogram (WSIZE=5). Note how, for example, the register value V0[3] slides in space across the ALU (Arithmetic Logic Unit) inputs (Cycle 2) and in time at the cycle change.	104

5.5	(5.5a) general diagram of the variable step shift register element used in ORS (Operand Rotation Stage). (5.5b) general diagram of the shift register element used in PCS (Partial Computation Stage). (5.5c) Pipeline diagram of SLIDEX Exec Unit with PWO=3 and WSIZE=5. The pipeline can completely hide the ORS latency.	105
5.6	(5.6a) SLIDEX column able to perform PWO=3 parallel window operations each cycle. (5.6b) SLIDEX reduction unit.	106
5.7	(5.7a) Cycles Per Instruction (CPI) of SLIDEX Exec. unit for different PWO (Parallel Window Operations) and WSIZE (Window Size) values. (5.7b) SLIDEX Exec. unit ALU utilization percentage for different PWO values and window sizes (WSIZE) assuming a fixed VL=64. (5.7c) Normalized ALU operations per instruction performed by SLIDEX-16, -32, and -64 solutions for PWO=3 and different WSIZE values relative to the operations per instruction performed by a vector processor with corresponding VL (Vector Length) value. The nomenclature SLIDEX-VL specifies the VL parameter of each SLIDEX version.	108
5.8	Relative end-to-end execution time breakdown of the main ORB-SLAM tasks.	109
5.9	Normalized main hardware metrics extracted during the execution of the Gaussian convolution and FAST feature extraction comparing the behavior of the vectorized version and that of different flavors of SLIDEX.	111
5.10	End-to-end latency of our localization system for the different systems tested in this work.	113
5.11	End-to-end average energy per frame for the vector and SLIDEX implementations.	114
6.1	(6.1a) Similarity study at different region sizes across adjacent frames for sequence 00 of the KITTI [52] dataset. (6.1b) Reduction in Normalized Temporal Gradient with increasing frame rate (FPS).	118
6.2	The figure compares two consecutive KITTI frames and highlights regions that exhibit spatio-temporal similarity. Yellow regions represent areas with low 2D frequency and high similarity. The blue regions correspond to distant areas in the scene that move slowly due to parallax.	119

6.3	General overview of the proposed vision pipeline.	122
6.4	Overview of δ LTA (δ ont't Look Twice, it's Alright) technique for discarding redundant regions in the frontend.	124
6.5	Correlation between the normalized Temporal Gradient and the percentage of similar regions obtained with δ LTA similarity detection method at 10 FPS (6.5a) and 30 FPS (6.5b) for the sequence 00 of KITTI [52] benchmark. . . .	126
6.6	Microarchitecture of the new ISP (Image Signal Processor) components: δ LTA unit and Signature Buffer.	127
6.7	δ LTA incorporates a new ISP stage for BRIEF (Binary Robust Independent Elementary Features) Similarity Filtering, which overlaps with other ISP stages.	128
6.8	Illustration of how the backend CPU reuses the computations for the redundant regions (R) and only processes the non-similar regions (C).	130
6.9	(6.9a) Percentage of similar regions. (6.9b) Maximum Absolute Pose Error (APE) for different thresholds.	134
6.10	Comparison of Normalized Latencies (6.10a), Normalized Tail Latencies (6.10b), and Normalized Energy (6.10c) for Baseline, Software Caching (SC), and δ LTA.	135
6.11	(6.11a) Percentage of region coverage for different camera rates. (6.11b) Amount of computations required when increasing frame rate.	136
7.1	General overview of the proposed vision pipeline.	142
7.2	Visualization of IRIS (Image Region ISP-Software Prioritization) prioritization maps. (7.2a) The foundational KITTI image. (7.2b) The EDM (Edge Density Map) highlights areas with dense attributes such as building details and cars. (7.2c) The MM, illustrating forward camera motion, approximates scene depth. (7.2d) The PPM (Priority Processing Map) combines EDM and MM, prioritizing regions with features and closer to the camera (compared with the EDM).	143
7.3	Integration of the new IRIS unit in the ISP to ensure zero-latency overhead.	146
7.4	Architecture of the IRIS unit to compute coarse-grained saliency maps in real-time.	147

7.5	Analysis of the FE stage and the characteristics of our iterative solution to perform it.	149
7.6	Comparison of baseline, software-only, and IRIS (not at scale).	152
7.7	Sensitivity analyses of image region and feature coverage. Figure 7.7a contrasts IRIS with random sampling and the ideal minimum. Figure 7.7b evaluates their response for different values of MFI (Marginal Feature Increase) threshold during the iterative FE execution.	154
7.8	Maximum Absolute Pose Error (APE) for different MFI thresholds.	155
7.9	Comparison of Normalized Latencies (7.9a), Normalized Tail Latencies (7.9b), and Normalized Energy (7.9c) for Baseline, Software-Only (SO), and IRIS. .	156

Glossary

δ LTA *don't Look Twice, it's Alright*. 5, 21, 36, 37, 42, 44–46, 66, 67, 117, 119–138, 160

AD Autonomous Driving. 28, 29, 31, 36, 38, 39, 53, 54, 138

ADC Analog-to-Digital Converter. 48, 51

ALU Arithmetic Logic Unit. 19, 20, 42, 96, 97, 104–108

AMX Advanced Matrix Extensions. 42, 43

APE Absolute Pose Error. 133

AR Augmented Reality. 28

BMA Block-Matching Algorithm. 51, 67

BRIEF Binary Robust Independent Elementary Features. 21, 60, 120, 123–129, 131, 137

CCD Charge-Coupled Devices. 48, 49

CE Convolution Engine. 42

CGRA Coarse-Grained Reconfigurable Arrays. 41

CMOS Complementary Metal-Oxide-Semiconductor. 48, 49

CNN Convolutional Neural Network. 34, 39, 41, 43–45

CPU Central Processing Unit. 3–5, 29, 31, 34, 37, 43–45, 53

CSI Camera Serial Interface. 48

CV Continuous Vision. 3, 5, 6, 18, 27, 29–31, 33–39, 41, 42, 44, 46–48, 53, 64, 95, 96, 117–122, 124, 125, 134, 137, 138, 159–161

DLP Data Level Parallelism. 5, 34, 96, 97, 160

DMA Direct Memory Access. 127, 128

DSP Digital Signal Processor. 53

EDM Edge Density Map. 21, 140, 141, 143–145, 147, 148

EE Edge Enhancement. 52, 143, 146

FAST Features from Accelerated Segment Test. 19, 20, 40, 58, 59, 71–74, 85, 86, 88, 90, 100, 102, 109–114

FB Framebuffer. 29, 36, 119, 120, 123, 124, 126, 129–132

FE Feature Extraction and Matching. 19, 22, 30, 31, 47, 53, 54, 56–58, 93, 109, 148–150, 154

FPGA Field-Programmable Gate Array. 41

FPS Frames Per Second. 46, 118, 120

GeMM General Matrix Multiply. 42, 43

GPS Global Positioning System. 3

GPU Graphics Processing Unit. 3, 30, 40, 53

HDR High Dynamic Range. 41, 127

IMU Inertial Measurement Unit. 39

IRIS Image Region ISP-Software Prioritization. 6, 21, 22, 38, 42, 45, 46, 66, 67, 139–143, 145–148, 151–158, 160

ISA Instruction Set Architecture. 5, 34, 42, 45, 95, 97, 99, 114, 160

ISP Image Signal Processor. 3, 6, 21, 29, 36, 38, 40–46, 48–52, 63, 64, 66, 67, 120–128, 131, 132, 134, 136–148, 151, 153, 155, 160

LOCATOR Low-power ORB aCcelerator for AuTonomOus caRs. 4, 18, 19, 31, 32, 35, 40, 45, 64, 65, 69–72, 88–91, 93, 94, 159

ME Motion Estimation. 51, 143, 144, 146, 147

MFI Marginal Feature Increase. 22, 150, 152–155

MIPI Mobile Industry Processor Interface. 48

MM Motion Map. 140, 141, 143–145, 147, 150

MV Motion Vector. 51, 157

NMS Non-Maximal Suppression. 58, 59, 71

NPU Neural Processing Unit. 53

OPT Optimization. 30, 31, 53–56

ORB Oriented FAST and Rotated BRIEF. 4, 18, 19, 31–33, 39, 40, 45, 54, 57, 58, 65, 69, 71, 73–75, 77, 78, 86, 88, 90–93, 159

ORS Operand Rotation Stage. 20, 104–107

PCS Partial Computation Stage. 20, 104, 105, 107

PPA Performance Per Area. 114

PPM Priority Processing Map. 21, 140, 141, 143–146, 148, 150–153, 155, 157

PPW Performance Per Watt. 114

PWO Parallel Window Operations. 20, 103–108, 110

rBRIEF Rotated BRIEF. 4, 18, 19, 32, 33, 45, 58, 60, 61, 65, 67, 70, 71, 73–75, 77, 78, 80, 81, 83–86, 88, 90, 92, 93

RS Reduction Stage. 104, 106, 107

RTL Register Transfer Level. 65–67

SAD Sum of Absolute Differences. 51

SC Software Cache. 43, 129–134, 136, 137

SIMD Single Instruction, Multiple Data. 4, 5, 34, 42, 96–98

SLAM Simultaneous Localization and Mapping. 30, 39, 54, 67

SLIDEX SLIDing window EXtension for image processing. 5, 15, 19, 20, 34, 35, 42, 45, 66, 95–100, 102–115, 160

SNR Signal-to-Noise Ratio. 51

SoC System-on-a-Chip. 3, 5, 29, 30, 35–37, 42, 45, 48, 49, 63, 64, 67, 140, 142, 143, 157, 158

SWP Sliding Window Processing. 5, 34, 42, 95, 97–99, 102, 103, 110, 114, 160

TD Temporal Denoising. 51, 140, 143, 146, 148

TG Temporal Gradient. 118, 125

TPU Tensor Processing Unit. 53

TSS Three-Step Search. 51

VIO Visual-Inertial Odometry. 30, 39

VL Vector Length. 20, 42, 99, 103, 104, 107–110, 112, 114

VLIW Very Long Instruction Word. 42

VPU Vector Processing Unit. 4, 34, 97

VR Virtual Reality. 28

WSIZE Window Size. 20, 99, 100, 102, 107, 108

XR Extended Reality. 3, 28, 29, 31, 39, 53, 54, 138

1

Introduction

This chapter outlines the driving forces behind this thesis. It highlights the significance of mobile CV (Continuous Vision) systems in the present and future computing landscapes. Subsequently, the chapter details the thesis' research scope and main contributions, concluding with a review underscoring how our research uniquely advances beyond prior state-of-the-art methods.

1.1 Thesis Motivation

Historically, computing machines operated within the confines of simplistic models, interacting with the world through textual or constrained graphical user interfaces. Today, they are evolving to perceive and interpret their surroundings with a complexity akin to living organisms, powered by advancements in computational power, energy efficiency, human-computer interfaces, machine learning, and algorithms.

From the personal computer revolution to the smartphone era, technology has continually moved closer to humans, becoming more integrated into our daily lives. With their

myriad sensors and cameras in small form factors and low-power profiles, mobile systems have set the stage for the next leap in innovation, pushing the boundaries of what machines can understand and how they interact with the world.

We are entering a new era mediated by the convergence of several influential trends. First, the abundant computational power and data available enable the development of increasingly complex algorithmic and machine-learning models with unprecedented perception and reasoning capabilities. Second, with the cessation of Dennard scaling and Moore’s law, application-driven specialization has emerged as a vital architectural technique to meet the requirements of the associated emerging applications. Third, designers increasingly deploy these novel applications on resource-constrained edge devices that directly interface with the end user and the physical world. This paradigm shift can reduce response latency and increase energy efficiency since computations are distributed and locally run on the edge rather than on the cloud. It also facilitates real-time processing and decision-making in various novel scenarios and form factors, in which edge machines transition from passive observers to active participants, seamlessly incorporated into our everyday routines.

Among the numerous emerging applications, we highlight two prominent representatives: XR (Extended Reality) and autonomous machines, specifically AD (Autonomous Driving). XR promises to amplify human abilities, providing immersive experiences that enhance learning [33], entertainment [60, 138], and remote collaboration [122], ultimately changing how we interact with digital content [26]. AD will revolutionize transportation by enhancing safety, reducing traffic congestion, and increasing mobility for those unable to drive [170]. The market trajectories of these technologies reflect their potential and significance. Reports forecast the XR market to reach US\$38.6 billion in 2024, with an annual growth rate of 10.77%, resulting in a projected market volume of US\$58.1 billion by 2028 [12]. Apple and Meta recently released AR (Augmented Reality)/VR (Virtual Reality)/XR Vision Pro [11] and Quest [110] headsets, respectively. Similarly, projections indicate that the AD market will grow from US\$41.06 billion in 2024 to US\$61.8 billion by 2026 [55]. Key players in the AD industry include Tesla [14], Mobileye [111], Waymo [135], Cruise [34], and Nvidia [136].

The advancements in XR and AD technologies highlight the increasing need for sophisticated machine perception capabilities. Vision, in particular, stands out as a pivotal modality in machine perception. Just as vision is crucial for animals to navigate, identify, and interact with their surroundings, it has become a cornerstone for machines to understand the

physical world comprehensively. Through the lens of advanced CV system, machines can discern intricate details, recognize patterns, and make informed decisions based on visual input.

Visual perception primarily encompasses object detection, tracking, and localization tasks [93]. Localization—the ability of a system to understand and place itself within its environment [47]—is foundational. This critical capability provides spatial awareness, enabling effective and meaningful interactions with the physical world. For example, in AD, precise positioning is crucial as a frame of reference to assess collision avoidance, pedestrians and other vehicles’ relative positions, and mission operations. In XR context, accurate head tracking is fundamental and one of the most critical operations since the whole rendered world depends on it for an adequate immersive user experience. As we will see in the following sections, these CV systems are typically deployed on mobile SoC (System-on-a-Chip) and face significant challenges in delivering accurate results in a timely and energy-efficient manner [93, 94, 169, 85].

This thesis explores architectural strategies to enhance mobile CV localization to meet the growing demands of the next edge computing era. Through this research, we aim to contribute to a future where machines can see, understand, and interact with the world in ways that were once the sole domain of living beings.

1.2 Problem Statement, Objectives, and Contributions

The general CV pipeline, shown in Figure 1.1, comprises three essential stages: sampling, imaging, and processing. The pipeline splits into two main modules: the frontend, managing sampling and imaging, and the backend, focusing on processing.

The frontend initiates the vision pipeline. At a high level, the sampling stage transforms the light from the world scene into electric signals. Following this, the imaging stage processes the raw image captured to enhance visual quality, converting it into a format suitable for further analysis or display. In Chapter 2, we will delve into the role of the ISP (Image Signal Processor), a specialized hardware component that typically completes this stage. Subsequently, the pipeline places the processed image in a FB (Framebuffer) or temporary storage for future use.

Later, the backend processes and analyzes the stored images to derive high-level visual insights. This stage utilizes a blend of computational resources, such as CPUs (Central

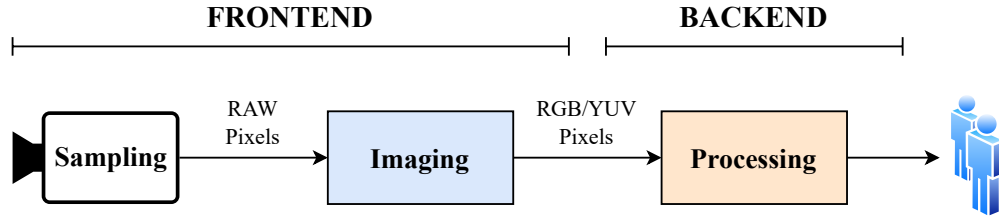


Figure 1.1: High-level depiction of a Continuous Vision pipeline.

Processing Units), GPUs (Graphics Processing Units), and specialized accelerators, which work together to process the image data.

The backend’s computational capabilities are vital for advanced applications such as visual localization, where real-time, accurate image analysis is paramount. This analysis is essential for promptly determining an agent’s position and orientation within a specific frame of reference using onboard cameras. While dense and sparse localization methods exist, the latter are more popular due to their computational efficiency and lower resource requirements [83]. Modern sparse visual localization systems comprise two primary phases: FE (Feature Extraction and Matching) and OPT (Optimization) [35, 23, 50]. While the discussed localization method relies on a pre-existing map, other alternative modes support SLAM (Simultaneous Localization and Mapping) [83, 28], which generates and updates the map in real-time. These methods can also integrate multi-modal fusion techniques like VIO (Visual-Inertial Odometry) [124].

As we will further detail in Chapter 2, the FE phase extracts 2D features from every incoming image. These features are points of interest within the image that the system can reliably and robustly recognize and track under various viewing conditions using an associated descriptor that encodes its appearance from the captured images. After FE, the system tries to find correspondences between the 2D features and previously seen 3D points from a map that the system can use to estimate the camera pose accurately. The OPT stage refines the predicted camera pose by minimizing the reprojection error, which is the difference between the projected positions of the 3D map points on the camera plane and their corresponding observed 2D features’ positions.

As edge devices integrate an increasing number of video cameras, each capable of capturing images at progressively higher resolutions and faster frame rates [75, 166, 48, 115], CV SoCs encounter multiple significant challenges.

The major challenge for CV systems is latency, especially in worst-case scenarios.

For instance, CV systems must react quickly to environmental changes in AD vehicles to ensure safe and efficient navigation. With cars capable of traversing 2-3 meters in just 100 ms, even the slightest delay can have dramatic consequences [37, 68]. In the XR domain, minimizing latency is crucial to avoid user discomfort and motion sickness [40]. The system’s response time should match or exceed the speed of the vestibulo-ocular reflex, the fastest human reflex, which stabilizes vision in under ten milliseconds during head movements [32]. For a satisfactory XR experience, industry experts recommended keeping latency below 20 milliseconds, with 60 milliseconds being the upper limit for acceptable performance [88].

Energy consumption also imposes significant limitations and exacerbates previous challenges, particularly in battery-powered edge systems where autonomy is critical. Energy-efficient and low-power devices are essential, as they reduce energy consumption and maintain user-friendly thermal conditions, thereby diminishing the need for intensive cooling. In autonomous driving (AD), just the computing engine can decrease driving range by 6% [93]. In the XR context, the demand for batteries can increase the weight of headsets or necessitate an external battery pack for prolonged use, as seen with products like Apple’s Vision Pro [11].

This thesis explores innovative architectural strategies to reduce future visual localization systems’ latency and energy consumption on edge devices. The following sections specify the objectives necessary to achieve this ambitious goal and explain how our contributions effectively tackle the challenges associated with mobile CV.

1.2.1 Image Feature Extraction Acceleration

Our first goal is to maximize visual localization’s hardware performance and energy efficiency by fully leveraging architectural specialization. Initially, we analyze a leading localization engine to identify critical performance bottlenecks. Figure 1.2 illustrates that the FE stage accounts for over 60% of the total execution time, a figure that persists as the target number of extracted features rises. Other studies have also reported similar results across different configurations and platforms [50, 7, 156, 95], highlighting FE’s significant impact on the system’s computational demands.

In response to this challenge, we introduce a heterogeneous architecture that combines LOCATOR (Low-power ORB aCcelerator for AuTonomOus caRs) for image feature extraction and a mobile CPU for the remaining tasks, such as tracking OPT, local mapping and loop closing [113]. LOCATOR specializes in the extraction of ORB (Oriented FAST and Rotated BRIEF) [134] features, a process that involves detecting corners in the image.

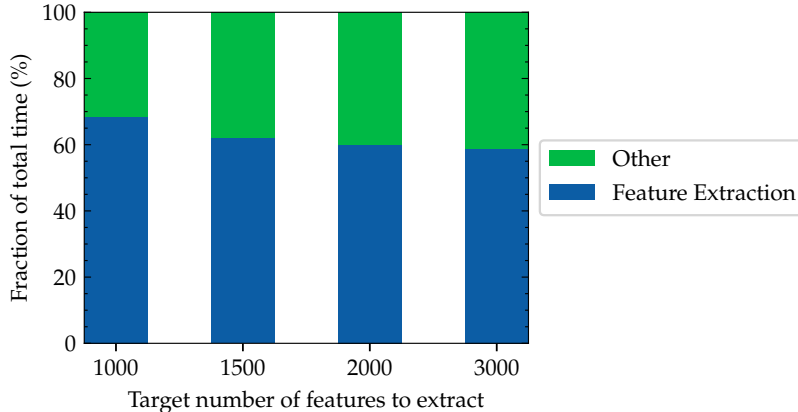


Figure 1.2: Feature extraction significantly dominates the execution time in visual localization systems.

This method is widely used in visual localization frameworks for its computational efficiency and robust detection capabilities across varying illumination, viewing angles, and rotation conditions.

ORB’s effectiveness largely stems from using rBRIEF (Rotated BRIEF) descriptors, which apply pixel subsampling within a patch surrounding the feature point to create reliable descriptors. Due to the irregular memory access patterns, computing this descriptor in hardware is the most challenging part. The computation consists of 256 comparisons between pairs of pixels in each corner’s neighborhood. The locations of the 256 pairs of pixels do not follow any regular pattern and change dynamically based on the orientation angle of the corner, which relates to the direction of the brightness gradient. Section 2.2.6 provides further details about this process.

Previous ORB accelerators modify the rBRIEF algorithm [95, 143] to obtain a more hardware-friendly version or fully replicate the rBRIEF hardware unit [160]. The first option incurs a significant accuracy loss, adversely affecting application outcomes. The latter incurs significant area and power costs.

LOCATOR replicates the rBRIEF unit but introduces several techniques to reduce its latency (reducing the number of required replicas) and its cost. To cut down latency, a static scheduling method developed through a genetic algorithm optimizes the mapping between the rBRIEF subsampling pattern and bank organization, reducing bank access conflicts. A lightweight duplication cache mechanism enhances this technique by augmenting the number of memory banks with duplicates of the repeated pixels in the rBRIEF pattern, further

reducing conflicts and latency. To lower replication costs, we observe that the distribution of access to banks is not uniform, and some banks have low utilization. Our design selectively employs a different number of ports for each bank based on this observation, reducing the cost of replication with a controlled impact on latency penalization. The rBRIEF unit also incorporates a multi-stage pipeline architecture, enhancing throughput by overlapping conflict resolution with pixel access.

The accelerator achieves $16.8\times$ speedup for ORB feature extraction, $1.9\times$ end-to-end speedup, and $2.1\times$ end-to-end energy reduction per frame, compared to running the complete localization system on an ARM Cortex A72.

This work has been published in the proceedings of the *33rd IEEE International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)* [148]:

- Raúl Taranco, José-Maria Arnau, and Antonio González. “A Low-Power Hardware Accelerator for ORB Feature Extraction in Self-Driving Cars”. In: *2021 IEEE 33rd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. ISSN: 2643-3001. Belo Horizonte, Brazil, Oct. 2021, pp. 11–21. DOI: [10.1109/SBAC-PAD53543.2021.00013](https://doi.org/10.1109/SBAC-PAD53543.2021.00013). URL: <https://ieeexplore.ieee.org/document/9651662>

Furthermore, an extended version of this work has been published in the *Journal of Parallel and Distributed Computing* [149]:

- Raúl Taranco, José-Maria Arnau, and Antonio González. “LOCATOR: Low-power ORB accelerator for autonomous cars”. In: *Journal of Parallel and Distributed Computing* 174 (Apr. 2023), pp. 32–45. ISSN: 0743-7315. DOI: [10.1016/j.jpdc.2022.12.005](https://doi.org/10.1016/j.jpdc.2022.12.005). URL: <https://www.sciencedirect.com/science/article/pii/S0743731522002507>

1.2.2 Sliding Window Support for Generic CPUs

While specialized solutions deliver unmatched performance and energy efficiency, crucial for highly constrained edge devices, they lack the flexibility to adapt to the evolving landscape of CV algorithms. This adaptability is particularly vital in this area, where computational platforms must readily adjust to future and unforeseen algorithms.

Our second goal is to enhance CPUs with domain-specific image processing capabilities, aiming to reduce the general overhead without compromising as much flexibility as accelerators. CPUs are ubiquitous in CV platforms, typically run visual localization engines, and are the most adaptable computational platform to date. However, they incur significant overheads. Hameed et al. [61] suggest that a modern CPU spends approximately 94% of its energy on instruction supply generality, with only 6% devoted to actual instruction execution.

To achieve this, we combine CPUs’ SIMD (Single Instruction, Multiple Data) processing with core image processing primitives. Most mainstream CPUs feature SIMD VPUs (Vector Processing Units) such as NEON [13], SVE [140], Intel AVX [71], and RISC-V [130], which amortizing instruction management costs (fetch, decode, issue) across multiple operations. However, despite their importance, existing SIMD solutions do not fully exploit the available DLP (Data Level Parallelism) available in foundational image processing operations like convolutions, crucial in CNNs (Convolutional Neural Networks) or image filtering, and stencil operations required for morphological operations or feature extraction. Both operations rely on a sliding window data flow to analyze each image pixel using a window of neighboring pixels that substantially overlap with the windows of adjacent elements. Unfortunately, current SIMD solutions generally manipulate the elements within a vector register independently of each other. While this provides excellent generality, it implies that, for example, for a simple 1D convolution, the programmer needs to load the data, perform element-wise multiplication with the first kernel component, execute a data shift, conduct a multiplication with the subsequent kernel component, accumulate the results, and so on. As a result, there is a notable gap between the available DLP and its actual utilization in these crucial image-processing tasks.

In response, we propose SLIDEX (SLIDing window EXtension for image processing), a novel high-performance and energy-efficient vector ISA (Instruction Set Architecture) extension to exploit SWP (Sliding Window Processing) in conventional CPUs. SWP extends the conventional vector SIMD execution model, treating vector registers like variable-sized groups of overlapping pixel windows. SLIDEX-enabled VPU processes multiple windows simultaneously, maximizing the DLP achievable per instruction while maintaining the same vector length. Furthermore, it significantly reduces the need for data access, movement, and alignment, decreasing memory and register file accesses compared to traditional SIMD designs.

SLIDEX significantly boosts processing speed for vital operations like 2D convolutions and stencil operations for feature extraction, achieving around a $1.2\times$ overall performance increase and up to 19% energy savings compared to standard vector extensions on a low-power ARM Cortex A55 core.

This proposal has been presented as a poster and published as a short paper in the proceedings of the *32nd IEEE International Conference on Parallel Architectures and Compilation Techniques (PACT)* [152]:

- Raúl Taranco, José-María Arnau, and Antonio González. “SLIDEX: Sliding Window Extension for Image Processing”. In: *2023 32nd International Conference on Parallel Architectures and Compilation Techniques (PACT)*. Vienna, Austria, Oct. 2023, pp. 332–334. DOI: [10.1109/PACT58117.2023.00039](https://doi.org/10.1109/PACT58117.2023.00039). URL: <https://ieeexplore.ieee.org/document/10364589?signout=success>

The full version of the work has been accepted for publication in the proceedings of the *38th ACM International Conference on Supercomputing (ICS)* [150]:

- Raúl Taranco, José-María Arnau, and Antonio González. “SLIDEX: A Novel Architecture for Sliding Window Processing”. In: *Proceedings of the 38th ACM International Conference on Supercomputing*. ICS ’24. Kyoto, Japan: Association for Computing Machinery, June 2024. ISBN: 979-8-4007-0610-3/24/06. DOI: [10.1145/3650200.3656613](https://doi.org/10.1145/3650200.3656613)

1.2.3 Decouple Camera Sampling from Processing

After the specialization-programmability backend exploration targeted in the previous objectives, we zoom out to focus on SoC-level architectural optimization. Frontend and backend communication costs in CV pipelines are three orders of magnitude higher than actual backend computation costs, as presented in Table 1.1. Consequently, our architectural strategy targets reducing these communication costs.

While previous optimizations with LOCATOR and SLIDEX have effectively reduced memory accesses by improving data locality, they are limited by the need to process all regions of every frame at the camera’s fixed capture rate. This approach does not account for the inherent smoothness in camera movement, where the camera typically captures images

Table 1.1: Energy consumption of different components in picojoules per pixel (adapted from [84]).

Component	Energy (pJ/pixel)
Sampling	595 [29, 42]
Communication (SoC - DRAM)	2800 [66, 74, 109, 118, 1]
Storage	677 [162, 107, 157]
Computation (per MAC)	4.6 [61]

of the same scene with gradual changes in position, angle, and orientation. This smoothness results in consecutive frames having a high degree of temporal similarity. For example, when an AD vehicle stops at a red traffic light, it processes similar frames. Likewise, when the vehicle begins to move again, some image regions, such as the sky or road, change minimally.

We leverage this smoothness property of input vision streams to propose δ LTA (*don't Look Twice, it's Alright*). δ LTA optimizes CV by decoupling frontend camera frame sampling from the backend processing. We introduce a new CV frontend ISP functionality that utilizes frame-to-frame similarity to discard image regions in the very early stages of the vision pipeline during the frontend camera sampling. Instead of writing the entire frame to the FB every time the camera captures an image, δ LTA frontend selectively updates only the regions of the frame that differ from the previous version and informs the backend which regions are different. This mechanism allows the backend to focus solely on processing the unique and distinctive regions of the image at their rate of change and not always at the arbitrary uniform camera sampling rate.

δ LTA significantly enhances efficiency compared to current systems by reducing frontend FB updates, minimizing redundant backend computations, and decreasing backend FB accesses, resulting in lower end-to-end latency and reduced energy consumption.

A pivotal advantage emerges when scaling the camera frame rate. Frame-to-frame similarity increases as the frame rate grows since the camera has less time to capture variations in the scene between consecutive frames up to a certain point when, for a given scene, more frame rate provides only redundant information. Hence, δ LTA can progressively filter the increasingly similar regions of these frames to limit the backend workload to the actual scene information. The decoupling empowers CV SoCs to not only avoid ineffectual computations but also to increase the system responsiveness since the backend can virtually process higher frame rates only for the regions that require it while saving resources for the ones

that do not.

In our localization engine, δ LTA significantly cuts down unnecessary memory accesses and redundant computations at both the front and backend, achieving a 15.2% reduction in application latency and a 17% decrease in energy consumption compared with the baseline mobile ARM Cortex A72 backend processor.

We published this contribution in the proceedings of the *56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)* [151]:

- Raúl Taranco, José-María Arnau, and Antonio González. “ δ LTA: Decoupling Camera Sampling from Processing to Avoid Redundant Computations in the Vision Pipeline”. In: *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO ’23. Toronto, Canada: Association for Computing Machinery, Dec. 2023, pp. 1029–1043. ISBN: 9798400703294. DOI: [10.1145/3613424.3614261](https://doi.org/10.1145/3613424.3614261). URL: <https://dl.acm.org/doi/10.1145/3613424.3614261>

1.2.4 Leverage Synergistic Frontend and Backend Cooperation

Our last objective aims to unleash ISP-software collaboration to enhance SoC-level efficiency. Current CV systems’ frontend and backend operations are significantly disconnected, as these two SoC components function without any synergistic collaboration in their image processing activities, missing optimization opportunities to reduce computation and communications costs.

We propose a strategy that repurposes the ISP’s internal imaging byproducts, usually discarded after processing each frame, to provide early insights for other components within the SoC. Backends, like CPUs, operate independently, often processing the entire image in a predetermined order, like raster-scan, to either pre-process the image for identifying key regions or post-process it for high-level semantic outputs. However, not every region in an image contains equally valuable information. For example, the image features tracked in visual localization tend to be concentrated around specific objects rather than spread uniformly across the image.

Figure 1.3 presents a histogram that illustrates the distribution of image features across equally sized grid regions, using data extracted from the KITTI [52] dataset sequences. Notably, a significant number of these regions, approximately 40% on average, contain no

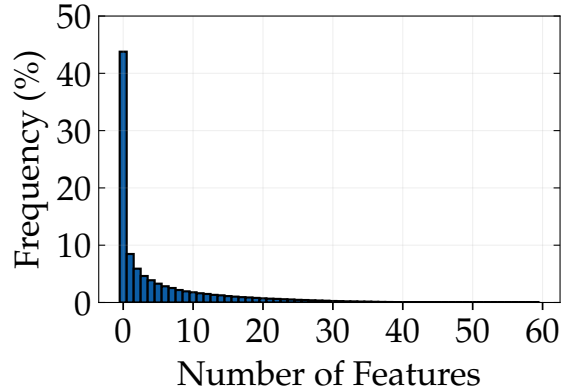


Figure 1.3: Histogram illustrating the spatial distribution of features, showing the frequency of grid regions containing varying numbers of features.

features. Thus, early insights from the ISP could identify these areas of interest, allowing the system to sidestep the access and processing of irrelevant regions (e.g., the sky or the road in AD), thereby reducing latency and energy consumption.

Building on this rationale, we introduce a novel approach named IRIS (Image Region ISP-Software Prioritization) to unleash ISP-software collaboration. IRIS enables the frontend to expose a priority map of the image regions based on their prominence, leveraging already computed operations along the ISP pipeline like edge detection and optical flow. With this priority image map, the backend can adopt an iterative or incremental image processing approach where the first iterations of the algorithms begin processing the most salient regions, yielding relevant insights before the baseline system.

We evaluated IRIS to enable effective iterative feature extraction for visual localization on a mobile ARM Cortex A72 backend processor. Our results show a reduction in tail, average latency, and energy consumption of 9%, 20%, and 16%, respectively. A paper describing IRIS has been submitted to publication and is currently under review:

- Raúl Taranco, José-María Arnau, and Antonio González. “IRIS: Unleashing ISP-Software Cooperation to Optimize the Continuous Vision Pipeline”. *Under review*.

1.3 State-of-the-art in Mobile Continuous Vision

This section reviews the latest advancements in Mobile CV systems. As edge devices increasingly process complex tasks locally, the demand for sophisticated vision algorithms

and efficient hardware intensifies. This overview highlights key technological developments, addresses the prevailing challenges, and identifies emerging opportunities, contextualizing the contributions of this thesis to the field.

1.3.1 Visual Applications

The research community has increasingly focused on enhancing the performance of CV applications, driven by the growing interests in XR and AD. Localization, object recognition, and object tracking represent three major active research areas for visual applications.

State-of-the-art visual localization systems encompass many approaches, including multi-camera SLAM, which enhances spatial perception; multi-modal systems that integrate various sensors for improved robustness; and learning-based methods that leverage deep neural networks for enhanced accuracy and adaptability in complex environments [28, 83, 168].

ORB-SLAM [23, 113], a cornerstone in visual SLAM, employs ORB features to deliver real-time, reliable localization and mapping with a system capable of handling monocular, stereo, and RGB-D cameras. Its robustness and flexibility have made ORB-SLAM a benchmark in visual localization research and applications, influencing our decision to use it in our experiments.

SOFT2 [35], a recent feature-based localization method, employs advanced stereo matching and map optimization to deliver precise visual localization. It rivals the accuracy of lidar-based systems and achieves top ranking on the KITTI dataset.

Multi-modal sensor VIO systems, systems like VINS-Mono [124] combine inertial measurements with visual data to improve pose estimation, especially when visual information alone is insufficient. By fusing camera and IMU (Inertial Measurement Unit) data, VINS-Mono offers increased robustness against rapid movements and feature-poor environments, demonstrating the strength of multi-sensor integration.

Furthermore, substantial research has leveraged deep learning for mobile object recognition and object tracking [78]. Learning-based methods are also increasingly popular in enhancing visual localization, with deep neural networks improving feature extraction, data association, and pose estimation. These methods offer adaptability and learning capabilities beyond what traditional techniques can achieve. For example, SuperPoint [39] employs CNNs to detect and describe key points across various scenes, ensuring robust localization

features. DeepVO [159] uses Recurrent Convolutional Neural Networks (RCNNs) to infer camera poses directly from the sequence of images.

However, despite their effectiveness, these learning-based approaches have substantial computational and energy costs, especially during inference, which often involves networks with millions of parameters and numerous layers. These requirements can significantly strain resources, particularly in environments where GPUs and other accelerators struggle to process other crucial perception tasks like object detection and tracking.

Researchers have proposed multiple deep learning optimizations, including the use of simplified models [127, 67], model compression techniques [62, 30, 108], and quantization methods [76, 54].

1.3.2 Hardware Specialization

Numerous solutions have been proposed in recent years to optimize the feature-based localization discussed in this thesis, including FPGAs [160, 143, 95, 144, 156], ASIC accelerators [87, 173], frameworks for automatically generating synthesizable accelerators [64, 96, 154], and accelerator-host communication approaches [147]. Another vital research direction involves hardware support for multi-sensor fusion and synchronization, such as camera-IMU combination [171, 50]. Most of these solutions try to accelerate feature extraction (e.g., FAST (Features from Accelerated Segment Test) [133]), speed up descriptor generation (e.g., ORB), or improve localization accuracy through multi-sensor data fusion.

Table 1.2 compares quantitatively LOCATOR with previous works. LOCATOR achieves high performance and shows the best energy efficiency, greatly improving Performance Per Watt (PPW).

Other machine learning techniques have increasingly received tailored hardware support, as evidenced by developments in dedicated accelerators designed to enhance their energy efficiency and performance [27, 82, 119, 6, 137].

1.3.3 Image Signal Processors

Most ISPs follow a fixed-function design described in Chapter 2. Despite this, there is a significant push within the research community to unlock greater flexibility in ISPs beyond conventional designs [78].

Table 1.2: Comparison with previous works. PPW stands for Performance Per Watt.

Work	Algorithm	Implementation	Performance	mW	PPW
[120]	FAST-BRIEF	ASIC, 130nm, 78.3k gates, 128kB SRAM	122fps, FHD, 200MHz	182	670
[173]	ORB-like	ASIC, 65nm, 127k gates, 205kB MEM	135fps, FHD, 200MHz	87.5	1542
[160]	ORB	FPGA, Arria V GX, 449 DPS, 206000 LEs, 231973 REGs, 1047kB BRAM	110.9fps, FHD, 230MHz	5340	20
[43]	ORB	FPGA, Stratix V, 8 DPS, 25648 LUTs, 21791 REGs, 1208kB BRAM	67fps, VGA, 203MHz	4559	14
[87]	FAST-BRIEF	ASIC, 65nm, 28kB SRAM	2170fps, VGA	1131	1918
[144]	ORB	FPGA, XCZU9EG, 33 DPS, 28168 LUTs, 9528 REGs, 188kB BRAM	108fps, FHD, 200MHz	873	123
[95]	FAST+RS-BRIEF	FPGA, XCZ7045, 80 DPS, 76424 LUTs, 101694 REGs, 120 BRAM	62fps, VGA, 100MHz	1963	28
[156]	ORB	FPGA, ZU3EG, 111 DPS, 56954 LUTs, 67809 REGs, 78 BRAM	55.87fps, VGA, 150MHz	4600	N.A.
LOCATOR	ORB	ASIC, 45nm, 32kB SRAM	120fps, FHD, 400MHz	10.84	15260

Vasilyev et al. [155] propose a programmable CGRA (Coarse-Grained Reconfigurable Arrays) framework to enhance the adaptability of ISPs. Additionally, numerous studies have effectively implemented custom ISPs on FPGAs (Field-Programmable Gate Arrays) [64, 65].

Originally intended to capture images for people, CV systems now generate images primarily for machine analysis. As a result, there is a shift towards simplifying or removing specific ISP stages, enabling algorithms to directly process RAW camera data to reduce latency and energy consumption. Liu et al.[97] developed an ISP design that selectively deactivates stages according to the application’s specific needs. Buckler et al. [20] explore how various ISP stages influence application performance and suggest modifications to simplify the ones without significant effect. Their research shows that color transformation stages generally have little effect on task performance and accuracy. They emphasize the importance of demosaicing, denoising, and gamma compression for tasks such as object detection (via CNNs). Hansen et al.[63] also highlight the critical role of tone mapping, especially for HDR

(High Dynamic Range) imagery in classification tasks.

While bypassing the ISP in the CV pipeline is appealing, the full impact on vision applications remains an area of ongoing exploration [78]. This thesis assumes the presence of a conventional ISP, but its core contributions remain valid regardless of the ISP’s role.

The δ LTA approach can operate as a standalone SoC component without an ISP. The IRIS implementation utilizes the ISP’s temporal denoising and edge enhancement, proven beneficial (as stated before) for machine image processing. Furthermore, IRIS shows that early-stage processing in the vision pipeline can provide essential data to the backend, facilitating reductions in communication and computational costs. These insights could be advantageous even for systems operating without an ISP.

1.3.4 Programmability and Architectural Flexibility

Prior work has investigated how the degree of programmability affects performance and energy efficiency in imaging hardware solutions [31, 15]. Remarkably, the CE (Convolution Engine) [123] is, like the SLIDEX unit, a programmable processor specialized for convolution-like data flows. CE partially exploits some SWP. However, it only supports a fixed set of window sizes (for 1D: 4, 8, and 16) and reductions (e.g., 4:1 and 8:1). Supporting more sizes requires extra interconnection circuitry for each kernel size, significantly raising the cost of the solution. SLIDEX differs from CE in its microarchitecture, allowing SWP to be exploited for any odd sizes from 1 to VL (Vector Length) natively and with a reduced cost that scales linearly with VL value, making our solution more general, programmable, scalable, and requiring fewer instructions. Moreover, SLIDEX exploits more inter-window parallelism as it computes partial results for all windows that fit in the processed register simultaneously.

Other proposals [25, 155] include VLIW (Very Long Instruction Word) capabilities to increase the ALU (Arithmetic Logic Unit) utilization of SIMD implementations.

Furthermore, Intel announced the inclusion of AMX (Advanced Matrix Extensions) [71] to their ISA. AMX aims to support GeMM (General Matrix Multiply) architecture through a tile register file composed of eight 1KB registers and instructions to manipulate them. This way, the programmer can express functionality with fewer instructions of high semantic value (e.g., load a tile or multiply two tile registers). This approach is related to our SLIDEX extension since, in both cases, the proposed ISA extensions manipulate data with

greater granularity, providing higher performance and lower instruction supply overhead. However, unlike AMX, our proposal exploits the operand overlap inherent in sliding window operations in a way that does not require introducing costly architectural extensions (e.g., tile registers of several KB), which could not be possible in embedded domains. Our proposal introduces a novel microarchitecture that minimally extends standard vector engines ubiquitous on modern processors.

Jeong et al. [79] explore the design trade-offs of integrating systolic arrays in CPUs for GeMM [79] using high semantic value instructions to manipulate the data. Our proposal also introduces new instructions to control the operation of a tightly coupled processor that resembles a systolic array.

Adding to this, domain-specific programming languages like Halide [126] or Darkroom [64] offer tailored solutions for image processing development. These languages simplify the programming of imaging pipelines, allowing developers to express complex operations and map them to different target platforms or accelerators.

1.3.5 Exploiting Spatio-temporal Similarity

Previous proposals have capitalized on temporal and spatial redundancies in real-time frames to reduce computational complexity, with approaches spanning both software and hardware domains.

Multiple software-based solutions function as a service, acting as a SC (Software Cache) to minimize system-wide computational redundancy in concurrent vision applications. Starfish [91] matches library calls with identical arguments, thereby reusing computation results. CBinfer [24] and DeepMon [69] incorporate CNN SCs, enabling computation reuse per layer by matching image regions within an equally sized grid. These methods rely on executing a similarity test on the CPU rather than the ISP, thus incurring overhead for frame transmission and similarity computation. Potluck [59] enables cross-application deduplication of image region computations, using a SC for reusing computations across spatio-temporally similar regions. DeepCache [165] allows cache hits for nearby similar image regions, running a lookup once at the input raw images and propagating reusable region boundaries across all layers, only recomputing the necessary parts.

All these solutions have a common aspect: they utilize a SC that stores a similarity metric of different regions. Some use ad-hoc metrics [24] or features like SIFT [59]. They

differ in the search methodology to identify similar neighbor regions. This search activity is very costly and may outweigh the benefits of computation reuse. They also differ in what they cache: results from image processing subroutines, CNN layer computations, etcetera. In any case, software caching transfers the entire image from the ISP to the main memory and from the main memory to the backend (e.g., CPU), at least to perform the cache lookup, which limits its efficiency and energy-saving potential.

In contrast to these previous works, δ LTA offloads similarity computation and matching to the ISP rather than relying on CPU/IP calculations. We proposed an efficient similarity detection overlapped with other ISP stages that, unlike other works, ensures that the overhead of our technique remains minimal even in the worst-case scenario.

Other previous hardware solutions also capitalize on temporal/spatial redundancies in real-time frames to streamline computations on mobile CPUs [104, 129, 44].

1.3.6 Frontend-Backend Collaboration

Recently, Kodukula et al. [84] introduced a mechanism allowing the backend to implement a policy that determines which image regions the frontend should send, as well as their spatial and temporal resolution. They evaluated its effectiveness for localization tasks. The backend (e.g., the CPU) must establish this policy without knowing future frame contents. Therefore, it must process a full frame periodically (every N frames) to prevent the accumulation of errors. In contrast, δ LTA enables the frontend to use frame-to-frame temporal similarities to discard similar regions and inform the backend, overcoming the speculative limitations inherent in the previous method.

PVF [51] harnesses the predictability of CV streams to introduce a speculative vision pipeline. This pipeline allows the frontend to predict future frames, scheduling them speculatively into the SoC’s heterogeneous IPs. When the actual frame arrives, the systems assess the predicted and actual frames similarly; if similar, the corresponding IP processing of the predicted frame may have already finished, minimizing frame latency. PVF decouples frontend processing from vision computation through speculation since the backend can anticipate the processing of future frames. Other proposals exploit motion vectors derived from classical optical flow methods [21] or hardware codecs [167] for CNN inference for object detection and tracking.

Euphrates [174] employ motion information collected from high-end ISPs to bypass

CNN inference for specific frames. The system performs full inferences of a subset of frames and uses the motion vectors extracted from the ISP to predict the results for the intermediate ones. Specifically, this technique translates the bounding boxes to their new positions based on the direction and magnitude of the perceived motion. However, this approach faces a critical issue: new objects entering the scene during frames with motion vector updates might go undetected.

Framebuffer compression is a technique to reduce SoC communication costs. Industrial examples include ARM AFBC [101], Imagination IMGIC [70], AMD DCC [53], and Chips&Media CFrame [139]. These proprietary techniques require decompression before processing by the consumer IP, implying that they do not discard redundancies or communicate with the IP about them. Compression reduces the energy consumption of the frame transmission. One of the most popular industry choices for these techniques is delta encoding lossless compression [4, 53], which can straightforwardly complement δ LTA and IRIS.

1.4 Thesis Organization

The remainder of this thesis is structured as follows:

Chapter 2 establishes the foundational knowledge for understanding this thesis, explaining the architecture of standard Continuous Vision SoCs and detailing the operation of predominant visual localization systems that operate on top of them. It also surveys the latest advancements in the field, framing the context for our contributions.

Chapter 3 outlines our experimental approach, detailing the evaluation tools used to assess the system’s performance, power, area, and energy efficiency. Additionally, it describes the image dataset used to test our localization engine.

In Chapter 4, we introduce the LOCATOR architecture aimed at improving ORB feature extraction efficiency. We address the main challenge of minimizing bank conflicts during rBRIEF descriptor generation by employing an offline genetic algorithm to find a pseudo-optimal access pattern and a static bank caching mechanism.

In Chapter 5, we present SLIDEX, an ISA extension to improve the efficiency of general-purpose CPUs to process fundamental windowed image primitives that are prevalent in visual applications. We exemplify its benefits over a localization engine’s convolutions and stencil operations.

Chapter 6 presents δ LTA, a technique to decouple camera sampling from processing, providing a scalable CV system capable of processing higher FPS (Frames Per Second) rates at lower costs. We then show the benefits of the technique by modifying our localization engine.

Chapter 7 presents IRIS, another technique to leverage already generated pre-processing information of the ISP to selectively process image regions based on their relevance in later stages of the vision pipeline. We applied it to implement iterative feature extraction.

Finally, Chapter 8 outlines some of the future steps and open research areas and summarizes the main conclusions of this thesis.

2

Background on Continuous Vision and Localization

This chapter provides an overview of our research context. We explore the architecture of contemporary mobile CV (Continuous Vision) systems. We then detail the operation of its primary hardware components and dissect them. Finally, we focus on a key CV application: camera-based visual localization, emphasizing the FE (Feature Extraction and Matching) stage, the primary bottleneck of the application.

2.1 Architecture of Modern Continuous Vision Systems

The architecture of the mobile CV systems is paramount in shaping future machines' perception capabilities and deployability. As introduced in Section 1.2, it comprises two main components: a frontend that transforms environmental light into high-quality digital images and a backend that derives high-level semantic information for perception and decision-making.

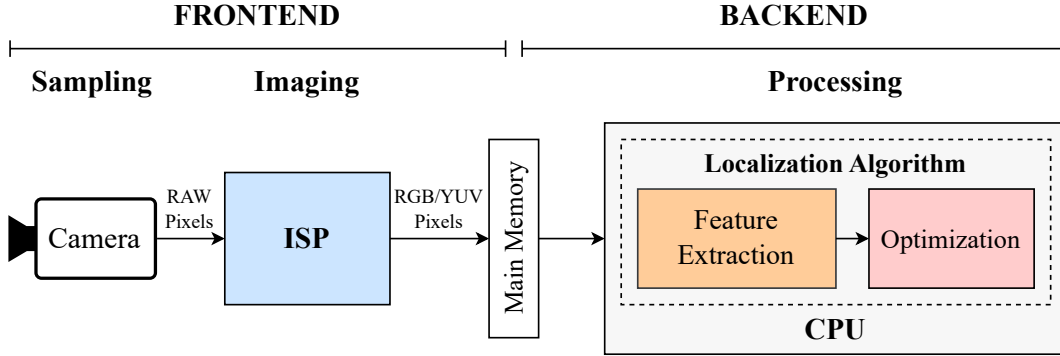


Figure 2.1: Depiction of a CV pipeline, highlighting the front and backend that embeds a visual localization system.

To support this pipeline, standard CV SoCs (Systems-on-a-Chip) typically follow the design presented in Figure 2.1. In this architecture, off-chip camera sensors capture light, generating RAW pixels transmitted to the SoC via the CSI (Camera Serial Interface), a MIPI (Mobile Industry Processor Interface) Alliance specification [92]. Within the SoC, the ISP (Image Signal Processor), a highly specialized SoC component, transforms RAW data into RGB/YUV domain pixels using algorithms like dead pixel correction, demosaicing, and white-balancing [8, 103, 63, 78]. The ISP then stores the processed image in a frame buffer in the SoC’s main memory. Subsequently, the backend analyzes these images to enable multiple applications such as object detection, object tracking, and, notably, visual localization—the central benchmark of this thesis.

The following subsections further detail each component of the described CV mobile architecture.

2.1.1 Camera Sensor

The camera sensor is an integral component of vision systems, serving as their primary interface between the physical and digital worlds. It comprises an array of photosensitive elements, typically CCD (Charge-Coupled Devices) [41] or CMOS (Complementary Metal-Oxide-Semiconductor) [41, 17] sensors. Each element within the array converts incident photons into an electrical charge proportional to the intensity of incoming light, a phenomenon essentially guided by the photoelectric effect. An ADC (Analog-to-Digital Converter) then converts these charges into digital data for processing. The design of camera sensors is vital in defining the overall image quality and sensitivity to light.

Notably, CMOS sensors have become more ubiquitous than CCDs due to their inherent advantages that align with the demands of modern technology [41]. They use less energy than CCDs, and their compatibility with standard semiconductor processes reduces production costs. CMOS technology enables on-chip integration of processing functions, simplifying camera designs and enhancing performance. Their rapid readout speeds also facilitate high frame rates and swift image capture.

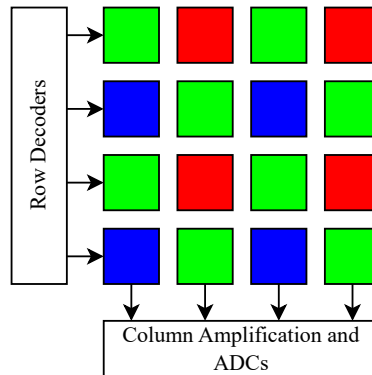


Figure 2.2: Fundamental structure of an image sensor, featuring a standard Bayer photosensitive array configuration.

Image sensors, by design, detect only brightness, not color, as they measure light intensity across their sensitivity range. To enable color detection, a color filter array overlays the pixels. A color filter array, precisely the Bayer pattern [78], is applied to introduce color sensitivity, allocating red, green, or blue filters to pixels in a ratio that reflects the human eye’s heightened sensitivity to green light. This arrangement ensures that the sensor’s color interpretation aligns with our vision. Figure 2.2 presents a typical image sensor, illustrating its photosensitive array arranged in the Bayer mosaic.

2.1.2 Image Signal Processor

An ISP is a specialized SoC component that converts RAW sensor data into high-quality images. ISPs comprise multiple signal processing stages to enhance image quality, as illustrated in Figure 2.3. The exact stages of an ISP pipeline vary and are often proprietary since vendors typically provide only a black-box abstraction of the ISP to end users through simple register values and ranges. In this section, we outline stages commonly found in most designs.

Most ISPs feature a fixed function design involving a cascaded stencil operations

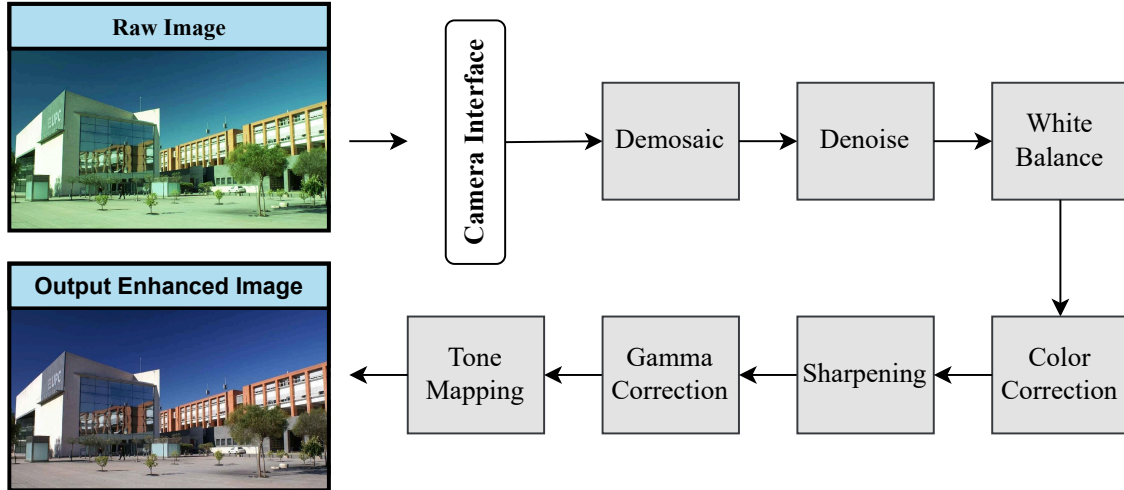


Figure 2.3: Traditional Image Signal Processor (ISP) pipeline.

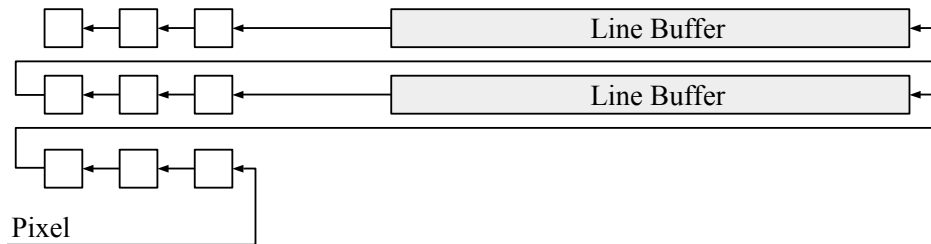


Figure 2.4: Scheme of a typical 2D line buffer employed on image processing specialized hardware.

pipeline on local SRAMs organized as line buffers [20, 64] to perform each image enhancement step. Stencil operations employ these structures [154] to process an input pixel window and generate output pixels, typically following the image storage pattern using raster scan order. Figure 2.4 shows a standard line buffer structure.

In this thesis, we base our discussion on the described ISP pipeline. We elaborate on its stages in the following sections.

Demosaicing

This process transforms raw image data from a Bayer filter array into a full-color image. It uses sophisticated algorithms to interpolate missing colors at each pixel, preserving edges and reducing artifacts.

Denoising

Denoising is a crucial ISP pipeline process that reduces image noise, enhancing image quality by filtering out unwanted signal variations or distortions without sacrificing important details. This step transforms raw sensor data into clear, usable images. There are two principal approaches: static and temporal denoising.

Static denoising tackles four primary noise sources from RAW images: shot noise from light detection, dark current noise, ADC quantization noise, and electronic read noise [78]. Researchers have formulated advanced algorithms like BM3D [36] and nonlocal means [19] to boost the SNR (Signal-to-Noise Ratio), safeguarding crucial image details such as edges and textures.

TD (Temporal Denoising) leverages the temporal correlation between consecutive frames to identify and reduce noise, improving image quality over time. This advanced method can be particularly effective in video processing or situations where multiple frames of the same scene are available. It leverages ME (Motion Estimation) techniques to detect pixel shifts across sequential frames [80]. TD uses pixel motion to replace noisy pixels with clearer ones from previous frames. Noteworthy commercial camera ISPs, such as ARM Mali C-71AE [103] and Qualcomm Spectra ISP [125], integrate these motion-aware functionalities.

ME methods can be broadly classified into differential-based and template-based methods. Differential-based methods, like optical flow [46], compute motion based on changes in pixel intensities over time. Template-based ME compares discrete blocks of pixels between successive frames to find the best match. BMA (Block-Matching Algorithm) [77] is the prevalent ME method implemented in ISPs due to its optimal blend of precision and efficiency.

BMA segments each frame into regions or macroblocks. For each one, it searches for the best-matching corresponding region in the prior frame using a similarity metric such as the SAD (Sum of Absolute Differences). The search spans a 2D window of $(2d + 1)$ pixels in both directions, with d being the search range. Some techniques, like the TSS (Three-Step Search) [81], aim to minimize computational demands. BMA determines an MV (Motion Vector) for each region, reflecting the displacement between the region and its closest match in the former frame. In our evaluations, we model an ISP implementing BMA and compute a motion metric for each region using the vector magnitude of the corresponding MV.

White Balancing, Color Correction, and Gamut Mapping

White balancing adjusts the image’s color temperature to align with the lighting conditions in the scene [78]. Camera manufacturers typically select matrix values for these transformations to achieve desired aesthetic effects.

Color correction in ISPs involves adjusting the color characteristics of an image to achieve desired accuracy and consistency, compensating for distortions introduced during image capture, such as lighting variations or sensor inaccuracies. Matrix operations or lookup tables (LUTs) typically achieve this transformation by mapping color values from one color space to another. The goal is to produce natural-looking images with consistent color reproduction across devices and viewing conditions, enhancing overall visual quality and ensuring color consistency across different applications.

Conversely, gamut mapping involves converting color values captured outside a display’s acceptable range into colors within that range.

Sharpening (Edge Enhancement)

EE (Edge Enhancement) accentuates boundaries and fine details to improve image sharpness. It involves applying an edge mask to enhance high-frequency components representing edges. Typically, ISPs use unsharp masking [38] to construct this mask by subtracting a blurred (low-pass filtered) version of the image from the original, thereby isolating the high spatial frequency components. Mathematically, this operation is defined as:

$$I_{EE}(x, y) = I(x, y) + \alpha \times (I(x, y) - I_{blurred}(x, y)) \quad (2.1)$$

Where I_{EE} represents the enhanced image, I the original, α a factor modulating the enhancement’s intensity, and $I_{blurred}$ a blurred version of the image (e.g., gaussian filtered).

Gamma correction and Tone Mapping

Gamma correction adjusts image pixel values to compensate for the nonlinear response of display devices, ensuring a perceptually linear appearance. It enhances brightness and contrast, preserving detail in shadows and highlights.

In contrast, tone mapping compresses an image’s dynamic range, enhancing visual contrast without overexposing bright areas, resulting in a pleasing image. Gamma correction ensures accurate display representation, while tone mapping aims for aesthetic effect while preserving detail.

2.1.3 Computation Backends

Computational backends are the powerhouse in CV systems, executing complex vision algorithms. These backends comprise well-known components like CPUs (Central Processing Units), GPUs (Graphics Processing Units), TPUs (Tensor Processing Units), DSPs (Digital Signal Processors), and dedicated accelerators, all crucial for processing the substantial data generated by mobile CV systems. Section 1.3 covers several backend solutions, yet the application of computational backends extends into everyday devices.

Apple, for instance, has integrated an NPU (Neural Processing Unit) [10] into its latest iPhones, enabling FaceID via facial recognition [3]. Google’s Pixel [57] smartphones employ an edge TPU for a range of vision tasks and previously incorporated the Pixel Visual Core [128], a dedicated accelerator for imaging. Similarly, Samsung and Qualcomm have recently embedded NPUs in their Exynos [73] and Snapdragon [161] chipsets to support analogous functions. Moreover, Intel’s Myriad2 [73] VPU is tailored for edge computing beyond smartphones, facilitating applications such as drone navigation.

2.2 Visual Localization Overview

As mentioned earlier in this thesis, localization is crucial in enabling a moving agent (such as AD (Autonomous Driving) vehicle or XR (Extended Reality) headset) to ascertain its position and orientation within a specific frame of reference. Modern camera-based localization systems comprise two primary stages: FE and OPT (Optimization) stages [50] (refer to Figure 2.1).

The FE stage plays a crucial role in these systems. It performs several image processing methods and extracts visual features from every incoming camera frame to find visual correspondences in consecutive observations. Based on our experiments (refer to Figure 1.2), this stage consumes approximately 60-70% of the total execution time, clearly indicating it as the primary bottleneck of the application. This bottleneck intensifies as camera resolution and frame rate increase, underscoring the need for optimized processing strategies.

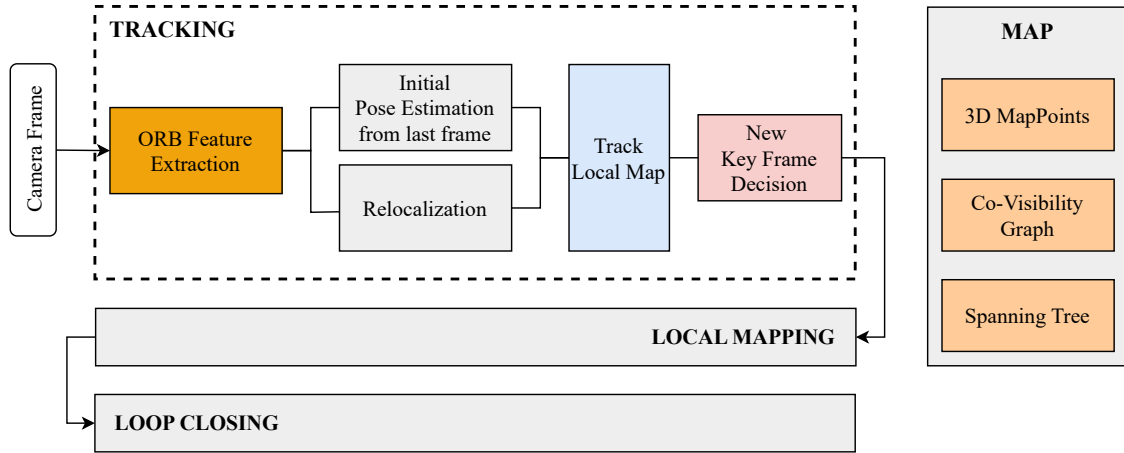


Figure 2.5: ORB-SLAM system overview [113]

On the other hand, the OPT stage calculates the agent’s position (x, y, z coordinates) and orientation (yaw, roll, and pitch) using the visual feature correspondences generated in the FE stage.

In this thesis, we focus on experimenting with a state-of-the-art visual localization approach, ORB-SLAM [23, 113], specifically in its localization-only mode. This approach, highlighted in the survey by Kazerouni [83], is utilized to underscore the benefits and applicability of the principal contributions of our research.

2.2.1 Overview of ORB-SLAM

ORB-SLAM is a prominent open-source SLAM (Simultaneous Localization and Mapping) system recognized for its efficiency and versatility, supporting monocular, stereo, and RGB-D cameras. It leverages ORB (Oriented FAST and Rotated BRIEF) [134] features for robust image keypoint detection and matching across frames. ORB-SLAM’s capacity to achieve high accuracy and low latency in pose estimation and map reconstruction, typically running on CPUs [50], makes it well-suited for AD and XR applications.

Figure 2.5 overviews the ORB-SLAM architecture, segmented into three principal threads: tracking, local mapping, and loop closing. These threads collaboratively manage the shared map structure, comprising 3D map points, keyframes, a co-visibility graph, and a spanning tree, crucial for preserving the system’s operational efficacy and state integrity.

Keyframes, selectively chosen for their significant viewpoint changes or the introduction of new, essential features, play a crucial role in building and refining environmental

maps. These frames capture vital pose information and are rich in features, enhancing robust map construction and maintenance. A co-visibility graph connects the keyframes, increasing system efficiency by showing which frames share observations of the same map points.

2.2.2 Automatic Map Initialization

Automatic map initialization is a crucial step, especially for Monocular localization, as it requires a procedure to establish an initial map due to the inability to recover depth from a single image. In ORB-SLAM, the process begins by estimating the relative pose between two frames to triangulate initial map points using ORB feature correspondences. The system computes two geometric models in parallel: a homography assuming a planar scene and a fundamental matrix assuming a non-planar scene. Then, a heuristic selects a model and attempts to recover the relative pose using a specific method tailored to each model.

Subsequently, the system checks if the camera's movements satisfy motion hypothesis specifications according to the selected model. This hypothesis is an initial assumption made by the system that a consistent and predictable motion exists between consecutive frames of the video sequence. This assumption implies that the camera undergoes smooth and continuous movement as it navigates the environment.

If the system confidently predicts camera motion between frames, it executes bundle adjustment [153] to refine map and camera poses. Otherwise, it reinitializes the map and camera poses to establish a more precise motion hypothesis. This reinitialization process may include steps to enhance feature matching or refine pose estimation.

Bundle adjustment [153], the heart of the OPT stage in ORB-SLAM, refines simultaneously the estimated camera poses and map points in the SLAM system. Bundle adjustment works by iteratively optimizing the camera poses and 3D map points to minimize the reprojection error between observed features in the images and their corresponding locations in 3D space.

2.2.3 Tracking

Tracking is the process by which the system continuously estimates the camera's position and orientation as it moves through the environment. The Tracking phase comprises several essential steps:

1. **Detecting Distinctive Features (FE):** Initially, ORB-SLAM identifies distinctive points or features in the current camera frame, crucial for subsequent tracking. We will delve deeper into this critical stage in Subsection 2.2.6.
2. **Matching Features Across Frames:** Next, the system matches the detected features in the current frame with those from the previous frame. This matching process establishes correspondences between features, enabling the system to track how the scene has changed over time.
3. **Estimating Camera Pose:** Using the correspondences between features from the last frame, the system estimates the camera's pose, including its translation and rotation relative to its previous position. If the system does not succeed, it tries to perform global relocalization using a place recognition system [49].
4. **Track Local Map:** After obtaining an initial set of feature matches and camera pose estimation, the system extends its search for correspondences by projecting potential 3D map points visible from the estimated camera pose into the current frame, leveraging the co-visibility graph.
5. **Optimization (OPT):** Utilizing all matched features, the system executes a full bundle adjustment to refine the camera pose.

After successful tracking, the systems operated according to two possible modes: SLAM mode and Localization-only mode.

In SLAM mode, the system updates the map by adding new keyframes and collaborates with the Local Mapping and Loop Closure threads. Conversely, in Localization-only mode, the system focuses solely on determining the camera's position within an existing static map. As a result, only the tracking thread remains active.

2.2.4 Local Mapping

Upon receiving a new keyframe in SLAM mode, the local mapping module springs into action, performing the following tasks:

- It incorporates the new keyframe into the co-visibility graph, establishing its connections with other keyframes.

- The stability of newly detected map points is verified to ensure their reliability.
- New map points are generated by triangulating ORB features observed in the connected keyframes.

After executing these steps, the system performs a local bundle adjustment to refine the map. Local bundle adjustment optimizes the positions of the map points and camera poses within a local area of the map. It corrects errors accumulated during mapping, improving map accuracy and camera trajectory estimation. Finally, the thread discards non-informative keyframes to maintain the algorithm's efficiency and structural integrity.

2.2.5 Loop Closing

The loop closing thread actively detects when the camera revisits previously mapped areas. Upon identifying these revisits, it works to reconcile these re-observations with the existing map. This active process significantly improves the global consistency of the map and enhances the accuracy of the camera's trajectory estimation. It ensures that the map maintains coherence and integrity by properly integrating new observations with the previously mapped environment, thereby refining the overall quality of the reconstruction and eliminating accumulated trajectory estimation errors.

2.2.6 ORB Feature Extraction

This section details the ORB feature extraction process utilized in ORB-SLAM for its computational efficiency and robustness against viewpoint, illumination, and rotation variations. Figure 2.6 displays ORB FE on an image from the KITTI dataset, with detected features highlighted in green. These features are primarily corners. Corners are invaluable in image processing as they are points where edges intersect, creating significant pixel intensity changes in multiple directions. Their geometric stability enhances recognizability across different viewpoints, preserving their angular characteristics despite perspective shifts. ORB detection techniques ensure corners are rotation and scale invariant, allowing consistent identification regardless of camera movement or zoom. Additionally, corners maintain contrast patterns more reliably under varying lighting conditions than other features. All these properties make ORB features uniquely identifiable landmarks critical for precise algorithmic matching.



Figure 2.6: Illustration of ORB feature extraction on a KITTI dataset image, with green points marking the detected features.

ORB FE stage comprises three primary steps [134]: Gaussian Pyramid Building, FAST (Features from Accelerated Segment Test) Keypoint Detection, and rBRIEF (Rotated BRIEF) Descriptor Generation, which collectively address the visual localization frontend processing bottleneck.

Gaussian Pyramid Building

FAST [132, 133] keypoint detector does not produce multi-scale features. For this reason, ORB extraction includes the generation of a scale pyramid of images before detection (see Figure 2.7a). The pyramid consists of several levels with versions of the original image reduced and smoothed using a Gaussian filter. Feature extraction occurs at every level of this pyramid.

FAST Keypoint Detection

The FAST feature extraction method, introduced in [132], is a neighborhood operator used for corner detection in images, commonly employed as features in computer vision. It tests a candidate pixel, p , to classify it as either a corner or not. This test compares the intensity of p with the intensities of the 16 pixels forming a Bresenham circle around the candidate, as illustrated in Figure 2.7b. A corner is detected at p if the intensities of at least $n = 12$ contiguous pixels out of the 16 surrounding p are all above or below its intensity by a threshold, t . The keypoint detection process involves NMS (Non-Maximal Suppression) to filter the features based on a score computed using the neighborhood of the FAST feature.

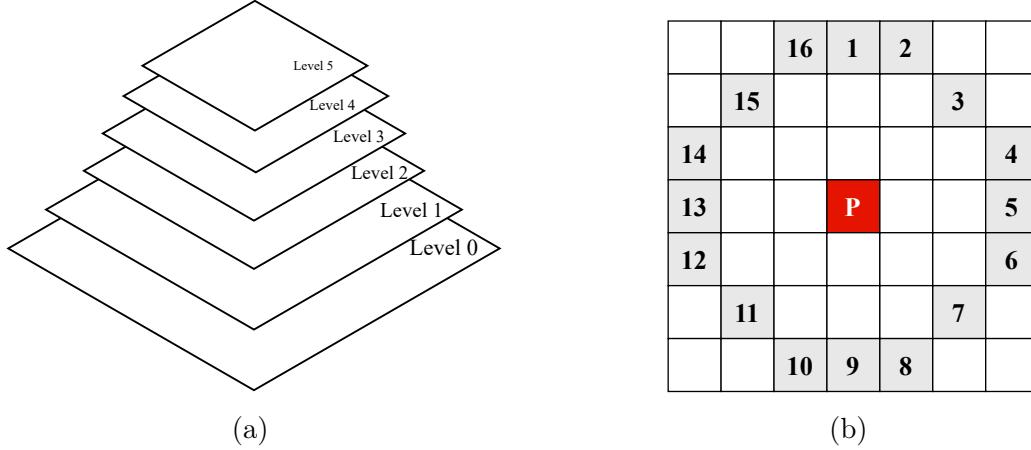


Figure 2.7: (2.7a) Pyramid of images generated from an original image. (2.7b) Bresenham circle of radius 3 showing the pixel access pattern for FAST around the pixel P .

This NMS procedure retains only the corners with the highest local maximum score within a neighborhood.

FAST features typically include a rotation component, which provides a fixed reference orientation for them. This orientation remains consistent across variations in viewpoint, image rotation, and lighting conditions, thereby enhancing the robustness and reliability of the feature descriptors. The orientation of a feature is determined using the intensity centroid with the following moment [131] calculations:

$$m_{pq} = \sum_{x,y} x^p y^q I(x, y), \quad (2.2)$$

where $I(x, y)$ represents the intensity at pixel (x, y) . The orientation centroid C is then calculated as:

$$C = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}} \right), \quad (2.3)$$

Furthermore, the orientation θ of the feature is given by ($Atan2$ is the quadrant-aware version of the arctangent):

$$\theta = atan2(m_{01}, m_{10}). \quad (2.4)$$

This orientation component will later serve as an input parameter for the feature's

rotation-aware descriptor, which we will discuss in the following section. In addition, it is possible to compute the $\sin(\theta)$ and $\cos(\theta)$ using the moments in the following way:

$$\sin(\theta) = \frac{m_{10}}{\sqrt{m_{01}^2 + m_{10}^2}}, \quad \cos(\theta) = \frac{m_{01}}{\sqrt{m_{01}^2 + m_{10}^2}} \quad (2.5)$$

rBRIEF Descriptor Generation

The rBRIEF leverages each feature’s previously introduced rotation invariant to enhance BRIEF (Binary Robust Independent Elementary Features). BRIEF is a method to generate a concise representation or signature of an image region’s pixel intensity distribution [22]. BRIEF describes regions as the concatenation of the results of intensity comparisons between sampled pixel pairs within the region.

The generation of the coordinates of the pixel pairs, n in total, is a fundamental aspect of BRIEF. This pattern is established once and consistently applied when constructing BRIEF descriptors across all the image regions intended for similarity comparison. Typically, pixel pair positions can be chosen either through a random sampling within the image region or through a more sophisticated learning-based approach where machine learning aids in selecting the optimal pairs based on training data [22].

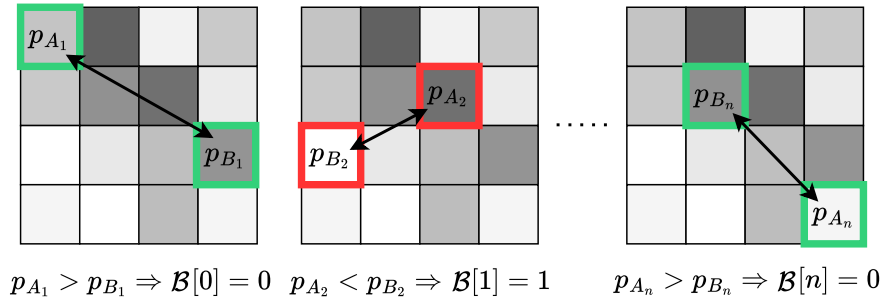


Figure 2.8: Example of the BRIEF (\mathcal{B}) operation to form the signature 01...0.

Given an image intensity function I , a binary test τ on an image patch at pixel coordinates p_A and p_B is defined as:

$$\tau(p_A, p_B) = \begin{cases} 1 & \text{if } I(p_A) < I(p_B) \\ 0 & \text{otherwise} \end{cases} \quad (2.6)$$

The BRIEF descriptor \mathcal{B} comprises n such binary tests:

$$\mathcal{B} = [\tau(p_{A_1}, p_{B_1}), \dots, \tau(p_{A_n}, p_{B_n})] \quad (2.7)$$

Figure 2.8 depicts a representation of the BRIEF method operation. For a given pixel pair, p_A , and p_B , if the intensity of p_A is greater or equal to p_B , the bit is set; otherwise, it is unset.

The rBRIEF descriptor ensures in-plane rotation invariance by rotating the coordinates for binary intensity tests according to the orientation θ , using the corresponding rotation matrix, R_θ . We express the matrix of binary test point coordinates (x_i, y_i) as:

$$S = \begin{pmatrix} x_1, \dots, x_n \\ y_1, \dots, y_n \end{pmatrix} \quad (2.8)$$

The rotated coordinates for the descriptor calculation are then:

$$S_\theta = R_\theta S \quad (2.9)$$

3

Methodology

This chapter outlines the methodologies, simulation environments, tools, and datasets for assessing our thesis contributions. It describes the specific mobile SoC (System-on-a-Chip) baseline characteristics evaluated in this thesis. We then detail the simulators and methodologies to model these characteristics, emphasizing the methods used to evaluate performance, area, power, and energy consumption. Additionally, the chapter discusses the dataset used for benchmarking, highlighting its structure and importance in the comprehensive evaluation and validation of our findings.

3.1 Baseline Overview

Our evaluation framework uses various models and simulators to analyze a modern mobile computer vision SoC's performance, energy consumption, and accuracy. This SoC includes an ISP (Image Signal Processor), a CPU cluster with a high-performance out-of-order core and a lower-power in-order core, resembling an ARM Cortex A72 and an A55, respectively. It features 8GB of LPDDR3 main memory and optionally incorporates the

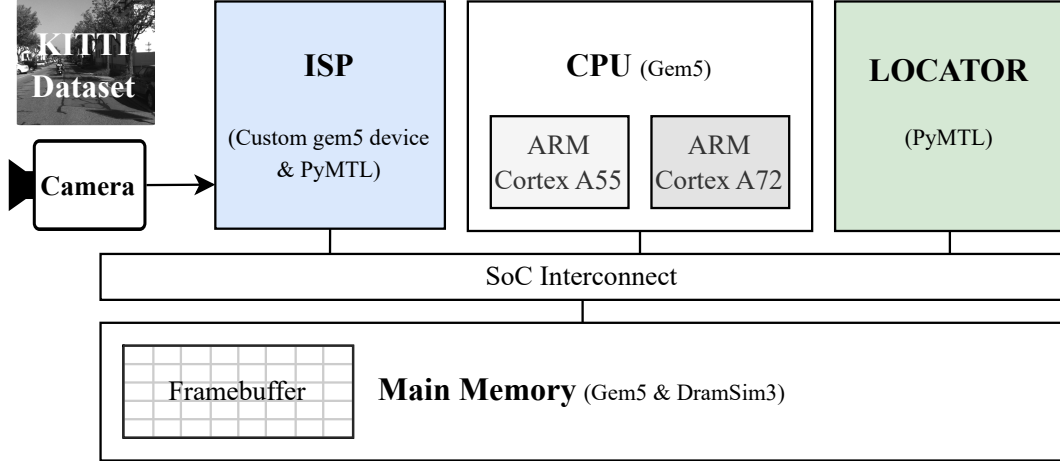


Figure 3.1: Illustration of our mobile CV (Continuous Vision) System on Chip (SoC), featuring an Image Signal Processor (ISP), a cluster of CPU, main memory, and an optional LOCATOR accelerator.

LOCATOR (Low-power ORB aCcelerator for AuTonomOus caRs) accelerator proposed in this thesis. Table 3.1 provides further details of the parameters used for each SoC component.

We employed gem5 [100] to model the core CV pipeline. We incorporate a custom gem5 device model for the ISP and DRAMsim3 [89] for the main memory modeling. We also utilized the standard CPU models (i.e., *Minor* and *O3CPU*) provided with gem5 to execute our localization engine.

We enriched the performance evaluations with energy consumption and area estimations. Like other related works [174], we used the Jetson TX2 board’s specifications to parametrize the baseline ISP consistent latency and power dissipation characteristics. We used McPAT [90] at 32nm for the backend CPU’s power and area, incorporating statistics from gem5 and DRAMsim3 to determine the main memory’s power dissipation.

Figure 3.1 represents the SoC components of our baseline mobile system. It details the models used, providing a clear and structured overview of our evaluation framework.

The following subsections detail the modeling of each proposal built upon the baseline model. We standardized results to 14nm using the method described by Stillmaker et al. [141] for consistency, as not all tools supported the same technology nodes or were available during evaluation.

Table 3.1: Simulation parameters for the baseline platform.

Component	Properties
ISP	700 MHz, 150mW, 10ms latency
A72-like	Out-of-Order ARM 64-bit, 3GHz, 3-wide fetch, 8-wide issue 90/256 int/float physical registers, 128-bit NEON vector L1: 32KB (I), 2-way + 32KB (D), 2-way; L2: 1 MB, 16-way;
A55-like	In-order ARM 64-bit, 1.8GHz, 2-wide issue, 128-bit NEON vector L1: 32KB (I), 2-way + 32KB (D), 2-way; L2: 256KB, 4-way;
Main Memory	8GB, LPDDR3-1600

3.2 LOCATOR Specific Modeling and Evaluation

We have developed a cycle-accurate RTL (Register Transfer Level) model of the ORB (Oriented FAST and Rotated BRIEF) accelerator proposed in this work and described in Chapter 4 by leveraging the PyMTL [98] framework. PyMTL is an open-source, Python-based framework that supports flexible prototyping, modeling, and hardware simulation from functional to cycle-accurate RTL models. Unlike traditional hardware description languages such as Verilog or VHDL, PyMTL facilitates concurrent simulation of components across different abstraction levels. This feature enables iterative testing to verify newly modeled hardware’s correctness and coherent behavior against higher-level models. Additionally, this framework assists in thorough performance analysis and creating complex test benches. These capabilities have significantly enhanced our hardware development processes.

Following this development framework, we evaluated different LOCATOR architecture versions, varying the number of rBRIEF (Rotated BRIEF) window replicas. Figure 3.2 illustrates the different stages of our hardware modeling development, visually representing the iterative testing and validation process.

We break down the application’s algorithm (i.e., ORB feature extraction) into its core stages. Each stage is then implemented at the functional level and validated against the reference software from the OpenCV library [18]. Subsequently, we develop a cycle-level design for each stage, verifying its functionality individually by utilizing functional models of the other stages through PyMTL’s multi-level simulation capabilities. After confirming the correctness of the cycle-level simulations, we apply the same verification process to the RTL development.

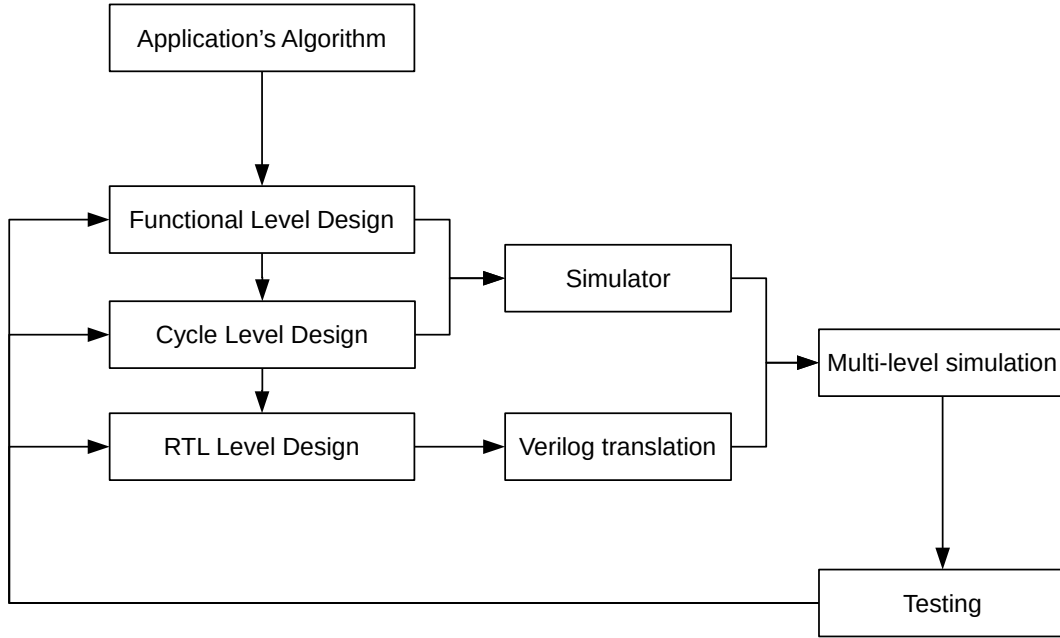


Figure 3.2: Methodology followed for designing and evaluating the RTL components.

In order to estimate area and critical path delay, we translate the PyMTL models into Verilog and synthesize them using Yosys [163] with the open-source 45 nm FreePDK45 1.4 [142]. Moreover, we obtain the power dissipation of the gate-level netlist of the accelerator using Synopsys Design Compiler [146].

3.3 SLIDEX Specific Modeling and Evaluation

We created an RTL description for the microarchitecture augmentation necessary to support SLIDEX (SLIDing window EXtension for image processing) as presented in Chapter 5, utilizing the Synopsys suite [146] and a 14nm cell library. We used CACTI [114] to compute the power dissipated by the new register banks added.

3.4 DLTA and IRIS Specific Modeling and Evaluation

We developed a detailed ISP model to evaluate the architectural augmentation of the δ LTA (δ ont't Look Twice, it's Alright) and IRIS (Image Region ISP-Software Prioritization) techniques described in Chapters 6 and 7. We constructed a bespoke gem5 device of the ISP that emulates its SoC-level interactions, particularly its framebuffer accesses. Moreover, we

Table 3.2: Default parameters used in ORB-SLAM3.

Parameter	Value
Number of Features	2000
Scale Levels	8
Scale Factor	1.2
Initial FAST Threshold	20
Minimum FAST Threshold	7

implemented another independent model of the internal pipeline of the ISP using a cycle-accurate RTL hardware description applying the same methodology explained in Section 3.2.

We modeled an RTL implementation of δ LTA and IRIS to validate its correct integration within the ISP pipeline. We ensure that the new stages do not create stalls by modeling the streaming pipeline of the ISP as a cascaded combination of stencil operations implemented using line buffers running parallel to the new unit. Additionally, we employed the Synopsys suite and a 14nm cell to estimate the power and area for δ LTA and IRIS units by synthesizing its RTL description. Finally, we used CACTI to model the internal on-chip buffers required by each technique.

To implement the specific δ LTA stage functionality, we configured the device to process images from our dataset, creating reuse and region similarity metadata using OpenCV’s rBRIEF algorithm. For the IRIS stage, we functionally generate the new region metadata of each image with the OpenCV library and an implementation of the BMA (Block-Matching Algorithm) algorithm.

3.5 Visual Localization Engine

In our evaluations, we focus on utilizing ORB-SLAM3 [23, 113] in localization-only mode as the principal application to assess the impact of our contribution to the mobile SoC vision pipeline. ORB-SLAM3 is a widely recognized and robust SLAM (Simultaneous Localization and Mapping) system that can run in real-time on CPUs.

Table 3.2 enumerates the default parameters set for ORB-SLAM3. These parameters are pivotal for ensuring a consistent and replicable analysis framework throughout our research.

3.6 Dataset

We employed the KITTI odometry benchmark [52] sequences. The KITTI dataset features 22 grayscale image sequences, each with a resolution of 1241x376 pixels, captured at 10 FPS. These sequences span various minutes of recording across different environments, totaling approximately 23,000 frames.

To assess the accuracy of our thesis contributions, we captured ORB-SLAM trajectories using the corresponding proposed techniques and compared them to baseline ORB-SLAM trajectories as our ground truth. We utilized the Evo [58] tool to analyze these comparisons and determine the pose errors in the results sections of each contribution.

4

LOCATOR

In this chapter, we propose a heterogeneous architecture for visual localization that combines LOCATOR (Low-power ORB aCcelerator for AuTonOmOus caRs) for feature extraction and a mobile CPU for the remaining tasks.

4.1 Low-power Accelerator for ORB Feature Extraction

This chapter introduces a low-power accelerator designed specifically ORB (Oriented FAST and Rotated BRIEF) feature extraction in image processing. Image processing algorithms typically involve stencil or convolution stages, focusing on localized segments by accessing neighboring pixels. Traditional designs assign dedicated hardware modules to each processing stage, often using line buffers to store data. These structures exploit data locality and minimize off-chip memory access, reading each pixel once and achieving consistently high throughput with low bandwidth requirements.

Our accelerator embraces a similar approach but focuses on the computational challenges of ORB feature extraction. It employs two synchronized datapaths: one for computing

FAST corners and the other for the Rotated BRIEF (rBRIEF) descriptor computation. Computing these descriptors is notably complex in hardware due to its irregular memory access patterns. Once the system classifies a pixel as a corner (i.e., a feature in the image), it computes an N-bit (typically N=256) descriptor. The computation consists of N comparisons between pairs of pixels in the corner’s neighborhood. The locations of the N pairs of pixels do not follow any regular pattern and change dynamically due to the rotation angle.

Previous ORB accelerators have adapted the rBRIEF (Rotated BRIEF) to obtain a more hardware-friendly version, unfortunately leading to a considerable drop in accuracy [95, 144]. Our work focuses on minimizing the cost of rBRIEF without compromising accuracy, striving to match the performance of software-based solutions.

LOCATOR stores the neighbor pixels in a multi-banked memory in the rBRIEF unit, the hardware component that computes the descriptors. Instead of using complex logic to dynamically schedule which pairs of pixels are accessed on each cycle, we develop a static scheduling method based on a genetic algorithm that minimizes the number of bank conflicts for any rotation angle. The system processes the 256 pairs of pixels in the order determined statically, reducing conflicts while requiring simple hardware. Note that for the matching between two descriptors to be valid, both must be generated using the same order since the metric used is the Hamming Distance.

Due to the low cost of the rBRIEF unit, it is possible to replicate it multiple times to achieve the target performance required to meet real-time constraints. We propose a specific combination of techniques to make a more effective replication. On the one hand, the rBRIEF pattern contains 26.75% of repeated pixels. It is possible to take advantage of this observation using a duplication cache mechanism that increases the number of banks and stores copies of the repeated pixels, resulting in a potential conflict and latency reduction. Furthermore, we observe that the distribution of access to banks is not uniform, and some banks have low utilization. Our design selectively employs a different number of ports for each bank based on this observation, reducing the cost of replication with a controlled impact on latency penalization. Finally, the rBRIEF unit employs a multi-stage pipeline design that overlaps the conflict resolution latency with subsequent pixel accesses.

4.2 Implementation

4.2.1 Hardware Architecture Overview

Figure 4.1 shows the streaming-based dataflow architecture of LOCATOR. It processes a stream of input pixels from memory or image sensors fed at a ratio of one pixel per cycle in raster scan order. The FAST, NMS, Gauss Filter, rBRIEF, and Rotation units employ on-chip buffers that support image tiling and access to a sliding window of pixels, efficiently exploiting the temporal and spatial locality of the operations. The storage capacity for the tiles depends on the tile width (see Table 4.2) and the size of the corresponding sliding window. All sliding windows of these units are synchronized so that the center pixel of the NMS (Non-Maximal Suppression) unit window corresponds to the center pixel of the rBRIEF unit.

When a new pixel arrives, the value follows two paths. The first component of the top path (see Figure 4.1) is a Delay FIFO (D-FIFO) queue that allows elements to leave the structure in FIFO order after a fixed number of cycles and is required to synchronize the top and bottom paths. The output pixels of the D-FIFO structure flow through the FAST (Features from Accelerated Segment Test) Detector unit. This unit maintains a window of pixels on which it performs the FAST segment test to detect corners and calculate its score. The NMS unit keeps a window with pixels' scores that it filters to determine salient corners. Concurrently, input pixels feed the bottom path (see figure 4.1), flowing through the Gauss Filter unit. The filtered pixels and their raw (unfiltered) version leave the Gauss Filter unit to feed the rBRIEF unit and the Rotation unit, respectively. The Rotation unit calculates the angle of an eventual feature. When the NMS detects a feature, it sends a signal to the rBRIEF unit, which begins the ORB descriptor computation process using the sine and cosine calculated in the Rotation unit and the gauss filtered window of pixels centered on the feature that the rBRIEF unit contains. The rBRIEF unit temporarily stores the descriptors in the ORB Output buffer.

4.2.2 Basic Sliding Window Structure

The sliding window structure in our design is commonly used to support 2D convolutions in image processing hardware. This structure provides temporary storage and a synchronization mechanism. Figure 4.2 illustrates a sliding window similar to the ones used

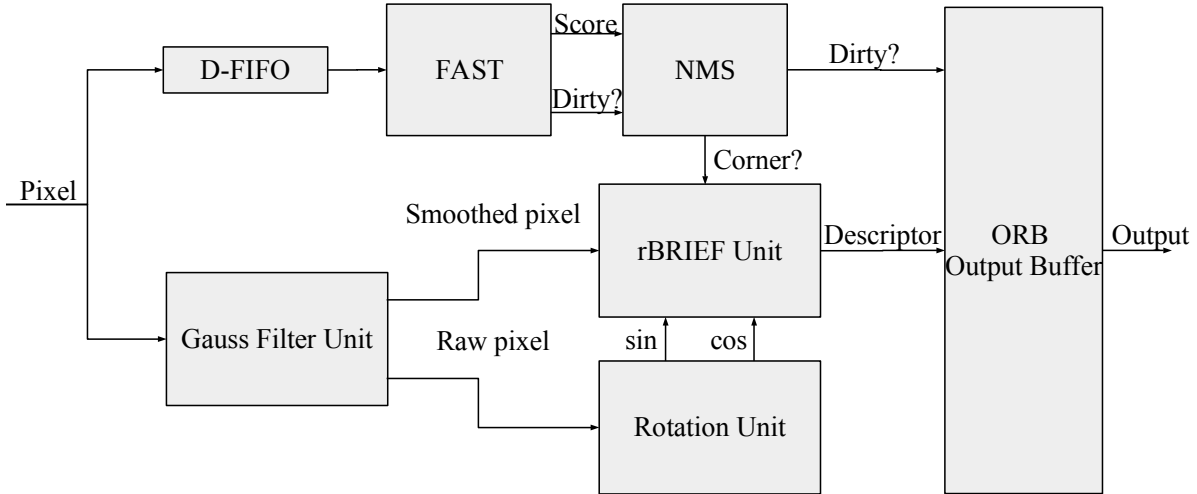


Figure 4.1: The basic architecture of LOCATOR.

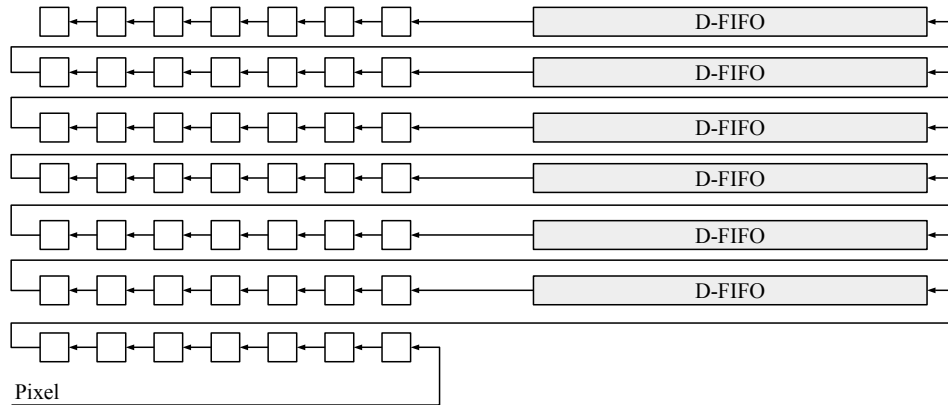


Figure 4.2: 7×7 sliding window structure.

by the FAST and Gauss Filter units, which in both cases require a kernel (window) of 7×7 pixels (fixed parameters used in ORB-SLAM).

A sliding window stores $W + (W - 1) \times Cols$ elements, where W is the square window size (7 in the example) and $Cols$ is the number of columns of the processed tile. The pixel stream flows through the elements organized according to the input data format. Initially, it takes a fixed number of cycles until the entire structure is filled and computations can begin. After this warm-up phase, the window of pixels shifts one position each cycle. When the window reaches the border of the tile, the structure takes a fixed number of cycles to realign the window to the next row.

4.2.3 rBRIEF Unit

The rBRIEF unit is responsible for implementing the most challenging part of the ORB extraction in hardware. It requires providing memory accesses to 512 positions that depend on the feature angle to compute the rBRIEF descriptor, making it hard to find an optimal access schedule.

Figure 4.3 shows the basic architecture of the rBRIEF unit. The implementation employs a 37×37 sliding window that holds the data required to perform each feature point candidate’s tests (Eq. 2.7). This window size matches the size used in the ORB software implementation. The structure must support the sliding window dataflow mechanism and, at the same time, random access to 512 points. The naive implementation allows one pair of accesses (two points) per cycle. Therefore, this design requires 256 cycles to generate a descriptor, while the FAST datapath only needs one cycle to process each pixel.

Sliding window structure replication alleviates the rBRIEF bottleneck by employing each replica to compute descriptors of different features in parallel. An Arbiter decides how to distribute the descriptor requests among the windows. The Coordinate Rotation module computes the rotated coordinates using the 0° rBRIEF pattern coordinates (stored in a LUT), the sine, the cosine, and efficient implementation of multiplications that utilizes just additions and shifts [158].

Although replication effectively reduces the rBRIEF latency and mitigates the accelerator’s bottleneck, it has a relevant impact on the area and power consumption. For this reason, we argue that it is necessary to consider alternatives that deliver the required performance at a much lower cost.

The following sections describe the design and the optimizations that we propose to increase the efficiency of the rBRIEF unit.

Exploiting Parallelism

Replication can be used to reduce the latency of a single descriptor computation by exploiting intra-descriptor Data Level Parallelism (DLP), that is, using each replica to compute a part of a single descriptor in parallel. When the FAST data path encounters a feature, it cannot continue processing the tile until the rBRIEF data path finishes the descriptor computation. Our experiments show that dedicating replicas to compute different descriptors independently to exploit inter-descriptor DLP is more effective because it enables

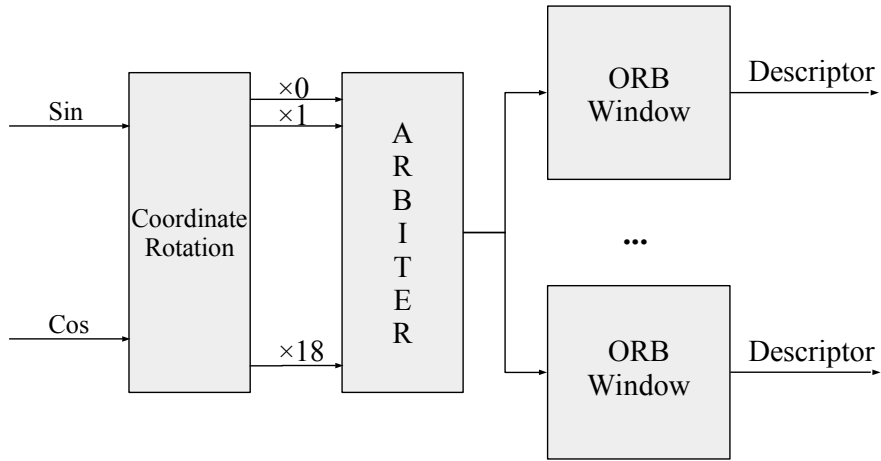


Figure 4.3: rBRIEF Unit architecture overview.

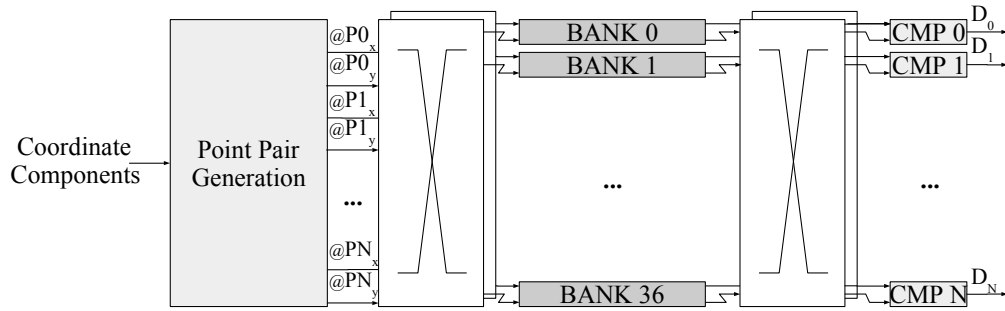


Figure 4.4: ORB Window architecture that allows parallel access to multiple pairs (stored in the banks shown in dark grey) per cycle.

decoupling the FAST and rBRIEF data paths. The FAST data path can continue processing the tile as long as there is at least one available replica not computing a descriptor. The fact that features are typically sparse helps to hide a portion of the rBRIEF latency bottleneck effectively.

However, increasing the granularity of the exploited intra-descriptor DLP is possible since the computation of each intensity test is entirely independent of the rest. Figure 4.4 illustrates the basic architecture of our initial design that supports multiple pair access per cycle. The initial design consists of a streaming-friendly architecture that segments the storage structure for each row in a separated bank of memory with two read ports. The Point Pair Generation module outputs multiple pair coordinates each cycle using the coordinate components received from the rBRIEF arbiter petition. A custom interconnection network is needed to route each point pair to the appropriate bank based on its coordinates, gather the results, and perform the intensity tests at the other end. With this approach, we could

potentially access 37×2 points (or 37 pairs) in one cycle, but the cost of the interconnection and routing for such a solution is prohibitive. Instead, we propose a design that allows parallel access to a group of pairs of a given size in each cycle. The appropriate group size was experimentally determined (see Section 4.3).

There may be a structural hazard or conflict if more than one pair has to access the same row bank and there are not enough ports. The control unit detects conflicts between pairs and orchestrates sequential access to the conflicting banks, introducing a stall in the pipeline. Non-conflicting pairs can be routed directly to the appropriate row bank. Each operand required for the intensity tests is always read from the same bank port, reducing the interconnection infrastructure's complexity.

Static Pattern Reordering

The number of conflicts varies according to the angle applied to the ORB pattern as it affects the composition of the pair groups and, hence, the static access scheduling. Since Hamming distance defines the metric space of the ORB descriptor set [134], statically rearranging the order of the pattern does not cause any issues with the quality or metric space of the descriptor. Finding an optimal order that minimizes the number of conflicts is an NP-Hard problem. For this reason, we propose to compute static scheduling based on an optimization performed with a genetic algorithm (GA). Even though other methods could have been used, a GA offers important advantages: it is highly adaptable to the problem, it is not necessary to have prior knowledge of the objective function, and it is easily parallelizable. These advantages allow us to integrate our expensive rBRIEF unit hardware models into the objective function and find pseudo-optimal solutions in hours.

The Static Pattern Schedule problem consists of a set of pairs $P = \{P_1, P_2, \dots, P_N\}$, and a set of groups $G = \{G_1, G_2, \dots, G_{\frac{N}{gsize}}\}$, where *gsize* is a fixed parameter that indicates the size of the group of pairs. An assignment is represented by a tuple $\langle P, G \rangle$. A solution consists of an assignment for every element of P .

Furthermore, we have two constraints: 1) No pair can be in more than one group, and 2) All groups must contain *gsize* number of pairs.

On the other hand, the objective function, F , is defined as the average latency of the unit for every possible rotation angle using the set of assignments as static scheduling. We assume an equiprobable distribution of angles and consider that the number of plausible

angles is bound. According to OpenCV documentation [18], the *fastatan2*, used in ORB-SLAM, uses a precision of "about 0.3 degrees".

In order to apply a GA approach as a meta-heuristic, we need to encode a candidate solution as a chromosome representation and define the Initialization and genetic operators. The chromosome chosen to represent a Static Schedule solution is a one-dimensional array A_i , where $0 \leq i < 256$ such that each element of the array represents an element of P . The group assignment is determined by A_i , the position within the array, as: $\lfloor \frac{i}{gsize} \rfloor$.

The initial population is generated by choosing random permutations of P . The genetic operators applied in each generation to this initial population are:

- **Fitness Evaluation:** In a biological sense, fitness is a quality value that measures the reproductive efficiency of chromosomes. We use the negation of the objective function, $-F$, since we want to minimize latency.
- **Selection:** We choose Tournament selection that involves running several "tournaments" among a few individuals chosen randomly from the population. The winner (fittest) of each tournament is selected for crossover.
- **Crossover:** Partially Matched Crossover (PMX) is chosen. This recombination operator generates two off-springs by matching pairs of values in a specific range of the two parents and swapping the values of those indexes [56].
- **Mutation:** The mutation is performed each generation, shuffling each chromosome of individuals with a given probability. The mutation swaps pairs between groups.

Table 4.1: Parameters used for the GA optimization of conflicts.

Parameter	Value
Crossover probability	70%
Mutation probability	20%
Population size	300
Crossover Operator	Partially Matched Crossover
Mutation Operator	Partial Shuffle Mutation
Selection Operator	Tournament Selection

Table 4.1 summarizes the parameters used for the GA. Figure 4.5 shows the evolution of the fitness of the best individual of all and current generations. In the example, the GA

converges to a local optimum in 500 iterations taking hours using an AMD Opteron 6338P with 24 threads. The optimization achieves an 18% reduction in latency compared with a random ordering.

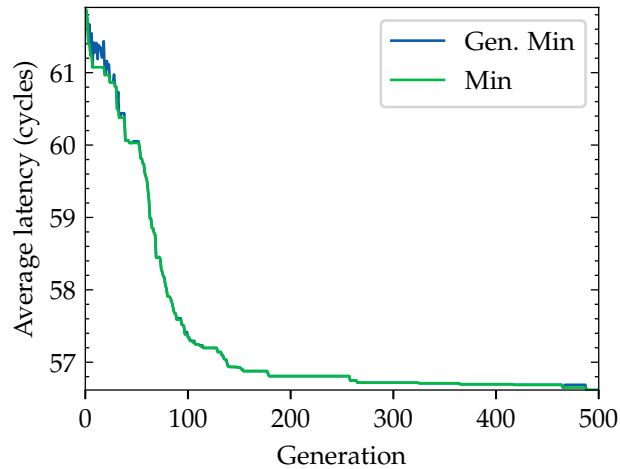


Figure 4.5: Example of convergence of the genetic algorithm to a local minimum for a $gsize = 8$.

Point Intensity Duplication Cache

An additional approach proposed to reduce conflicts is exploiting the temporal reuse of the points of the ORB pattern, as many are accessed more than once (not all points of the original pattern are unique). The original ORB pattern is composed of 375 unique points out of the 512 total points (256 pairs), giving room for a 26.75% reduction in the number of accesses. We leverage this observation by applying Point Intensity Bypass and Point Intensity Duplication Cache.

On the one hand, Point Intensity Bypass consists of a reformulation of the definition of a conflict, taking into account the reuse of points. In this way, two pairs of a group conflict if they access the same bank but to a different address within it. If the address is the same, it is possible to save one access since the two accesses target the same point. The control unit must consider both point coordinates when detecting conflicts collapsing accesses to identical coordinates. The output routing sends the read value to the output registers that originated the collapsed access requests.

On the other hand, Point Intensity Duplication Cache adds a structure to store the repeated ORB pattern points. The rBRIEF unit suffers from structural conflicts due to the

need to perform more reads than the available read ports allow. Thus, the cache increases the read ports for repeated points, reducing the conflicts since many accesses will stop being made to the rows of the sliding window and will be made to the duplication cache.

The implementation is based on adding one or more banks to the sliding window connected to the same interconnection network and accessible through the usual bank mechanism. For simplicity, the banks use the same memory structures as the rows of the sliding window. The cache is accessed using coordinates outside the range of the sliding window (without spatial meaning).

When a repeated point is accessed for the first time, the read value is used to process the rBRIEF descriptor, and at the same time, it is stored in a duplication cache bank. The static ORB pattern stored in the unit needs two new fields to implement this functionality: a bit to indicate whether the point is repeated and is the first time accessed in the pattern and the coordinates of the duplication cache banks in which the point will be stored. These fields depend on the specific rBRIEF pattern and are calculated offline using a coordinate assignment method. Figure 4.6 shows the basic structure required to access the new banks that act as duplication cache. The tables shown represent the information of two repeated points read in the cycle illustrated in the figure. *Cache X* is the address inside the cache, and *Cache Y* is used to select between the two banks of the available caches. The multiplexers that select between the values of the four pixels read in each cycle are operated with a selection signal generated by the control unit. The control unit tracks the code of each of the four possible inputs.

The coordinate assignment method statically enforces the placement and replacement policies of the cache since we can simulate its behavior for a particular ORB pattern and know the future access to repeated points. In addition, there is no hit/miss management since the correct operation is guaranteed by construction, reducing the cost of the solution. During execution, the unit only has to read the cache coordinates of each pattern element from a pre-computed table and store the point value in the corresponding position of the duplication cache if it is the first time the repeated point is accessed.

There are potential cache bank conflicts if more than one pair in a group with repeated points are assigned to the same duplication cache bank and port. In such a case, it is possible to apply optimization to minimize conflicts when applying the placement policy during the offline execution of the coordinate assignment algorithm. One way would be to distribute the points in the different banks taking into account the number of repeated points contained

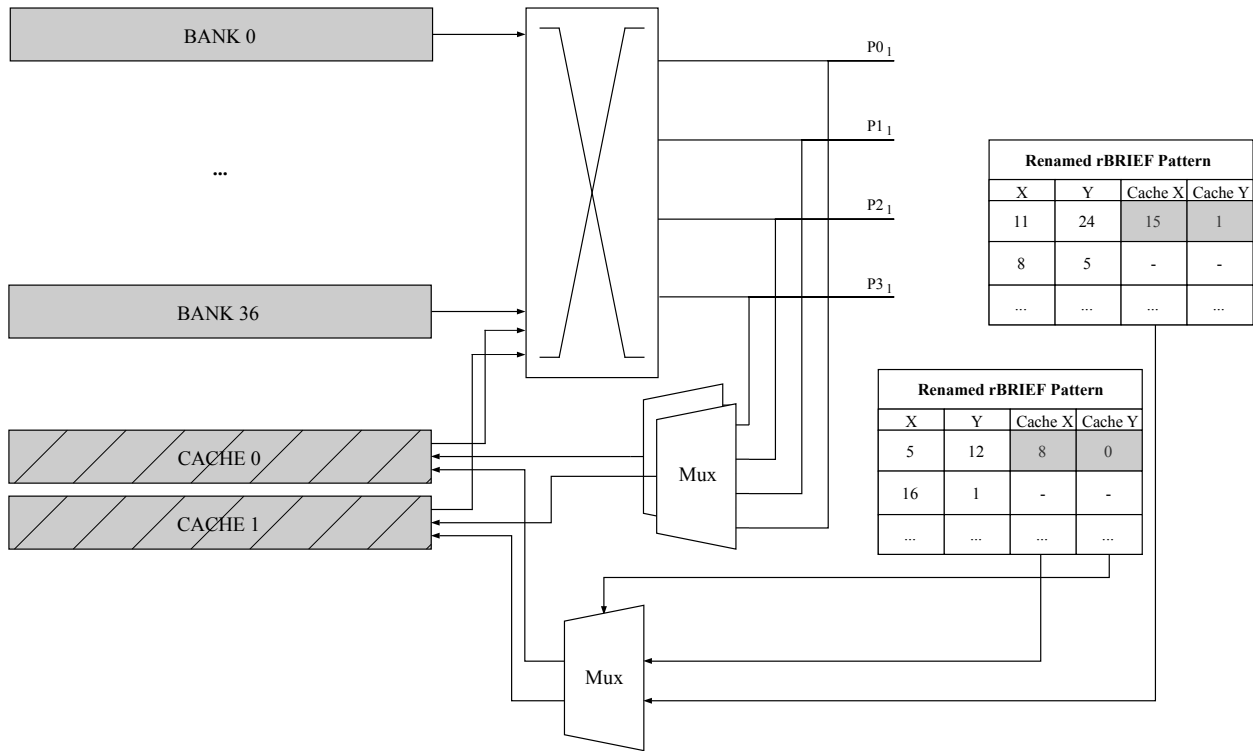
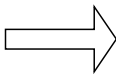


Figure 4.6: The basic structure of the duplication cache (new cache banks shown filled with a line pattern) for a group size of 4 pairs. Note that only the first point of each pair PN_1 is represented.

in the groups of the rBRIEF pattern. However, we have left this exploration for future work. In addition, we believe that an increase in conflicts and, thus, latency may signal unpromising solutions to the genetic algorithm described in subsection 4.2.3. Therefore, the generic algorithm will tend to penalize solutions with groups containing multiple conflicting repeated pairs, promoting solutions with repeated points better distributed among the pair groups.

The maximum required size for the cache is 137 bytes which could be stored in 4 banks of 37 bytes. Since all repeated points would fit in the cache, no replacement policy would be needed. However, it is possible to determine a perfect replacement policy that reduces the number of banks needed, taking into account that it is possible to statically determine which points will no longer be used in the future computation of the rBRIEF descriptor. This way, after the last access to a repeated point, its position in the cache is available to store other values. According to our experiments, the working set of repeated points fits within two banks.

Original rBRIEF Pattern					
ID	X	Y	Cache X	Cache Y	Repeated?
0	11	24	0	37	1
1	16	1	-	-	0
2	3	6			0
3	8	12			0
4	11	24	-	-	0
5	15	24	-	-	0
	19	27	1	37	1



Renamed rBRIEF Pattern					
X	Y	Cache X	Cache Y	Repeated?	
11	24	0	37	1	
16	1	-	-	0	
3	6	-	-	0	
8	12	-	-	0	
0	37	-	-	0	
15	24	-	-	0	
19	27	1	37	1	
...	

Figure 4.7: rBRIEF pattern renaming example. The left table shows a segment of the original rBRIEF pattern where points with ID 0 and 4 are repeated. The assignment algorithm determined that the coordinates to store the value of the ID 0 point in the duplication cache are (0, 37). Thus, the static renaming changes the coordinates for the subsequent non-first accesses, such as ID 4, to point to these coordinates (right table). The renaming avoids points ID 4 and ID 5 conflict in the original pattern (marked in red).

The last step to implement this proposal consists of applying a static renaming mechanism of the coordinates of the rBRIEF pattern. The renaming is applied to the coordinates of the repeated points that are not the first accesses. The coordinates of the original pattern are replaced with coordinates from the duplication cache for non-first accesses instead of using the original pattern coordinates, potentially avoiding conflicts with other points. Figure 4.7 illustrates an example of the described renaming scheme.

Selective Replication of Ports

The first design used for each sliding window replica has 37 banks which become 39 by applying the Point Intensity Duplication Cache optimization. Using two ports per bank, each of which is pinned to the same operand of the pairs, simplifies the design since it is not necessary to verify conflicts between points of the same pair or operands of different pairs pinned to the opposite port.

However, the ports are not used evenly, leading to the distribution of conflicts illustrated in Figure 4.8a. The rBRIEF pattern, is rotated according to the angle of the features found during the accelerator operation. The Y coordinate of the banks that a point can occupy ranges from $[-r, r - 1]$ where r is the radius or integer distance to the center of the patch, $(0, 0)$. The radii of action of the points with a larger radius overlap those of smaller ones, and since the points of the rBRIEF pattern are distributed roughly uniformly throughout the patch, the number of accesses to the inner banks will be greater than that of the outer banks. As corner examples, a point with a radius of 16 pixels will potentially access all banks, and one with a radius of 0 will always access the central bank.

To take advantage of this observation, we propose to selectively determine the number of ports used in each bank based on the conflict distribution observed. In this way, the outer banks can provide a single port, decreasing the area and power consumption of the sliding window replica. It would also be possible to add more ports to central banks to reduce the conflicts from accessing these banks. However, to simplify the implementation, we have decided to support banks with single or dual-port capability.

Reducing the number of ports of some banks increases the number of conflicts affecting performance. Figure 4.8b shows the performance slowdown of the replicas depending on the number of banks with only one port. The slowdown is the relative measurement obtained by normalizing the average latency of each single-ported configuration with the baseline (without single-port banks) latency. Note that the affected ports are distributed symmetrically. If we

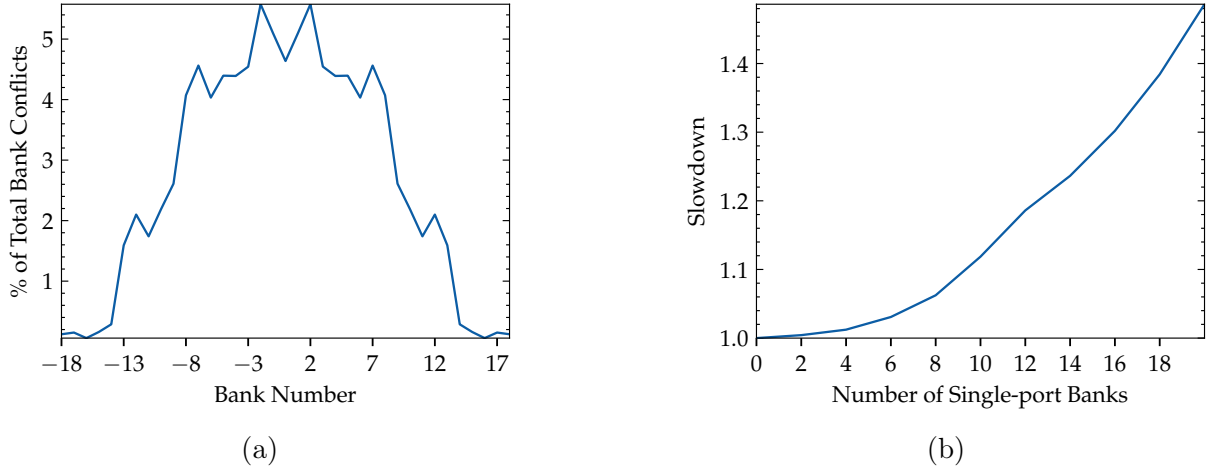


Figure 4.8: (a) Percentage of total conflicts (Y-axis) generated for each bank on average for all angles. The X-axis shows the index of the banks in the range $[-18, 17]$. The bank with an index of zero is the central bank. (b) The slowdown of the average access latency of a replica when increasing the number of external single-ported banks. The parallel access group had a size of 4 pairs in this experiment.

eliminate two ports, the affected banks are the outermost ones with Y coordinates -18 and 17, those with coordinates -17 and 16, etc.

Leaving the banks with a single port implies having to modify the control unit. Access to these ports can generate conflicts, even from the same point pair. Nonetheless, the cost incurred is reasonable due to the small number of external single-ported banks employed. In addition, the single-ported banks are connected to a multiplexer that the control unit operates to select one of the two requests that can arrive from the interconnection network.

Changing the outer four banks from dual to single ported memories allows adding new banks for the Point Intensity Duplication Cache without increasing the cost of the solution. Section 4.3 shows the resulting performance improvement and energy efficiency balance obtained by combining the duplication cache and the technique presented in this subsection.

Pipelining

When a bank access conflict occurs in the baseline, the conflicting bank accesses are serialized while all other accesses for the pair group being processed are performed in parallel. The serialized accesses to the conflicting bank determine the latency necessary to

process that group of pairs, and, in addition, the following groups are not processed until the conflicts are resolved. During the cycles of conflict resolution, there is an underutilization of resources since other pairs could be accessed using the free ports of the replica where there were no conflicts in the previous cycle.

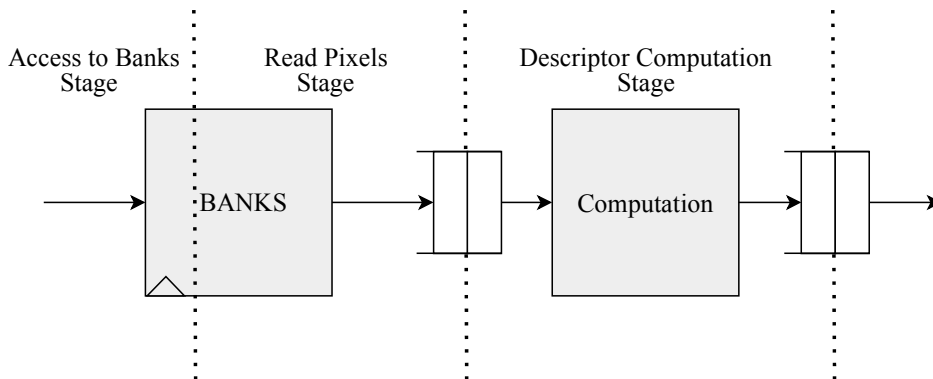


Figure 4.9: The proposed stages and pipeline mechanism of each replica of the rBRIEF. The FIFOs between the stages have a length of 2.

To exploit the existing parallelism between the computations of different groups and hide the penalty incurred due to conflicts, we have explored using a pipelining mechanism by adding blocking FIFOs between the stages that we have determined in the design. Figure 4.9 shows a diagram of the architecture of this solution with three stages: *access to banks*, *read pixels*, and *descriptor computation* stages.

In the *access to banks* stage, the banks are accessed, selecting those that will be retrieved employing the X and Y coordinate of each point. In the *read pixels* stage, the intensity values are read from the banks and stored in the corresponding FIFOs. Finally, in the *descriptor computation* stage, the pixels are read from the FIFOs when both points of each pair are available and the output FIFO has space left.

With the proposed design, each replica in the rBRIEF unit can be seen as an in-order statically-scheduled superscalar processing element that can issue multiple pair accesses with banks being a single shared resource. Each pair issue is independent since there are no data dependencies between the pixels of the pairs. Furthermore, the stages of the pipeline are synchronized through the status of the FIFOs.

When the FIFO is full, the previous stage must stall. A stage can process the input data if the head of all the necessary FIFOs (depending on the stage) contains valid data and if it can write to the output FIFO. For example, the *description computation* stage can

compute a new bit if the FIFOs of the two operands (the pair) with which to perform the intensity comparison contain an element, and the output FIFO has space. The rest of the FIFOs of the other operands do not have to be processed or do not belong to the same group. In addition, in the *access to banks* stage, a priority system is established in which the group that has been processed the longest has the highest priority. In this way, we ensure that the pixels are processed in the correct order generating valid descriptors. At the same time, this simple design enables the exploitation of the parallelism between the accesses of different pairs of groups and the overlapping between accesses and conflict resolution.

We have performed an experimental exploration to determine the length of the FIFOs that retain state between stages. Figure 4.10a shows the replication latency for different length values for the FIFOs. According to our results, we opt to use FIFOs that allow queuing the state of each stage in flight two times. This result seems reasonable since, after genetic optimization, it is rare to find groups with more than one conflict.

Each element of the FIFO between the *Read Pixels Stage* and the *Descriptor Computation Stage* contains the pairs of pixels (2×8 bits) read for each group (i.e., 64 bits for G4 and 128 for G8). The elements stored in the output FIFO contain one descriptor bit for each group (i.e., 4 and 8 bits for G4 and G6, respectively).

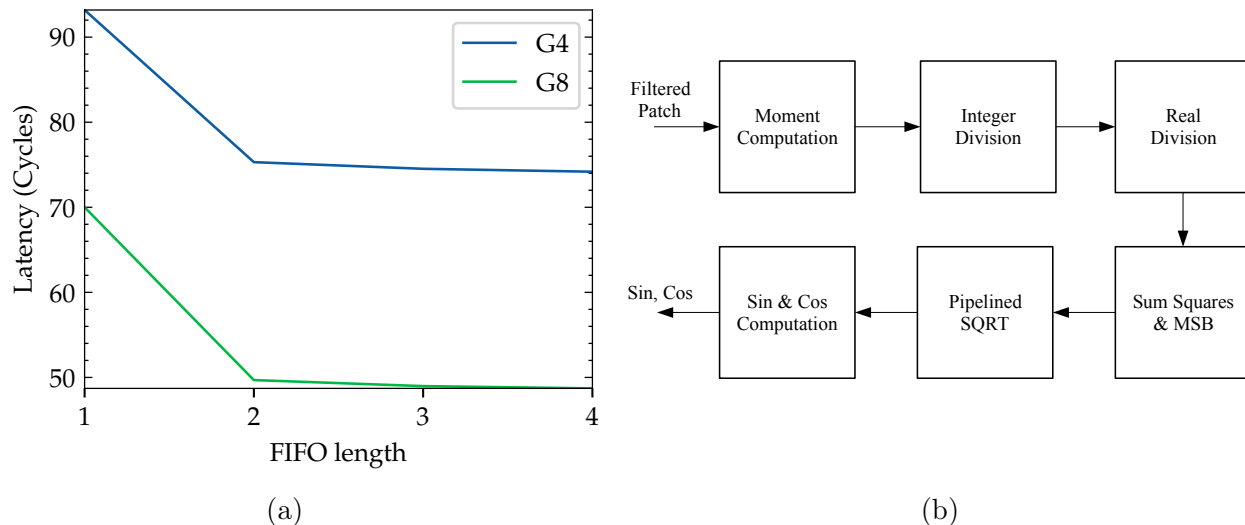


Figure 4.10: (a) Sensitivity to the length of the FIFOs in the replica pipelining stages of the rBRIEF unit for a group size of 4 (G4) and 8 (G8) pairs processed in parallel. (b) The pipelined architecture of the Rotation Unit.

Finally, it is worth noting some details about integrating the pipelining mechanism with the duplication cache optimization discussed in the subsection 4.2.3. When applying

the duplication cache technique, it is necessary to guarantee the coherence of the replication cache data and the renaming of the banks for a given rBRIEF pattern.

There can be more than one group being processed in flight by the replica due to pipelining. Thus, pixels can be stored in the duplication cache at the wrong time. To address the potential issues, we opt to follow a conservative approach that is statically applied. The renaming algorithm will only change the bank of a repeated pixel if, for every angle, the value in the cache is correct. The method employs a functional model of the replica to check it. Genetic optimization will be able to bring out the solutions that make the best use of the duplication cache.

4.2.4 FAST Detector Unit

Figure 4.11 illustrates the architecture of the FAST unit. The module employs a Sliding Window structure with the required patch size of 7×7 (see Subsection 2.2.6), achieving a throughput of one pixel tested per cycle. The FAST implementation used by ORB-SLAM applies an adjustment of the threshold at run-time, increasing the sensibility if no features are found inside a region of size 30×30 . The module detects corners speculatively with the *MinThr* until at least one corner of *IniThr* is found (if any) inside each region. If at least one corner with the default threshold is found, the descriptors generated in that region with the *MinThr* are discarded. The Dynamic Threshold module keeps track of the status of each region and is responsible for setting the dirty bit high when an *IniThr* corner is detected.

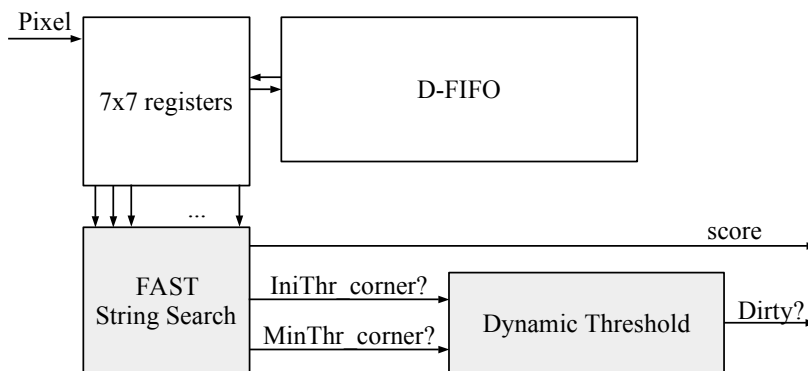


Figure 4.11: Architecture of the FAST unit.

To perform the segment test and score computation, we use a similar solution as other works [143]. Each pixel intensity of the Bresenham circumference is compared with

the central pixel obtaining a 16-bit value. An AND tree with a depth of ten levels searches for a sub-string with 12 consecutive set bits to classify the candidate pixel as a corner.

4.2.5 Non-Maximal Suppression Unit

A 3×3 Sliding Window is used to filter the FAST features. The sliding window is fed with the FAST scores. In each cycle, the center pixel of the window is compared with the eight surrounding pixels to determine if it is the local maximum. This operation is implemented with eight comparators and an AND reduction.

4.2.6 Gauss Unit

The patch around the feature point must be smoothed before computing the intensity tests to generate the rBRIEF descriptor. The Gauss Filter Unit generates every cycle a pixel and its Gaussian smoothed version. We employ fixed-point arithmetic to represent the values of the Gaussian kernel (8 bits for the integer part and 4 for the fractional one) and intermediate results (24 bits) before rounding to obtain the filtered value (8 bits).

4.2.7 Rotation Unit

The rotation unit is in charge of calculating the sine and the cosine of the angle of the vector formed joining the centroid and the candidate feature point. This angle is required to rotate the coordinates of the ORB pattern applying Equation 2.9. The precision of the calculations is key since errors in the rotation of the points in the ORB pattern produce severe degradation of the quality of the generated descriptors.

We propose the use of the inverse square root to compute it instead of a LUT of pre-computed values used in state-of-the-art solutions [160, 121, 144]. The unit uses a 37×37 Sliding Window synchronized with the rBRIEF unit. In addition, the unit is pipelined into several stages, as shown in Figure 4.10b that compute an accurate estimation of the sine and cosine computation per cycle.

The first pipeline stage of the Rotation Unit computes the moment of a window. In order to do it efficiently, Equation 2.2 can be reformulated as follows:

$$\begin{aligned}
m_{01_{n+1}} &= m_{01_n} + 18 \times (C_n + C_{n-37}) - S_n, \\
m_{10_{n+1}} &= m_{10_n} + xC_n - xC_{n-37}, \\
m_{00_{n+1}} &= m_{00_n} + C_n - C_{n-37}.
\end{aligned} \tag{4.1}$$

where,

$$\begin{aligned}
S_{n+1} &= S_n + C_n - C_{n-36}, \\
C_n &= \sum_{1 \leq x \leq 37} p_{x,n}, \\
xC_n &= \sum_{1 \leq x \leq 18} x \times (p_{38-x,n} - p_{x,n}).
\end{aligned} \tag{4.2}$$

and $p_{x,n}$ is the pixel's intensity at the coordinates within the patch indicated by row x and col n . The moment computation consists of an adder tree used to compute C_n . Furthermore, an optimized Multiple Constant Block [158] (MCM) is used to compute xC_n .

The following two stages perform the centroid division. Image moments, m_{01} and m_{10} , computed in the previous stage, could be directly used to determine the trigonometric functions. However, we divide these values by m_{00} reducing the number of bits for the image moment components representation and, therefore, the cost of its manipulation in later stages. Efficient division circuits are employed for integer and fixed-point division, with a precision of $\frac{1}{32}$. The accelerator computes the moment components using 20 bits, but after the normalization with m_{00} , the values fit into 10 bits (5 bits for the integer part and the other 5 bits for the fractional one).

The next stage is the summation of squares and estimation computation. A first estimation of the inverse square root is performed considering the Most Significant Bit (MSB) of the previous square sum. This results in a good estimation considering the following:

$$\log_2\left(\frac{1}{\sqrt{x}}\right) = -\frac{1}{2}\log_2(x) \tag{4.3}$$

Next, the fast inverse square root is computed in three stages. The inverse square root of x , the summation of squares previously determined, is computed using an approximation based on the Newton-Raphson method. Three iterations of the following formula

are employed using as y_0 the MSB-based estimation:

$$y_{n+1} = y_n \left(\frac{3}{2} - \frac{x}{2} y_n^2 \right) \quad (4.4)$$

The final stage computes the sine and cosine, applying Equation 2.5 and updating the signs of the final values. The fixed-point representation of the trigonometric values uses 16 bits for the fractional part.

4.3 Experimental Results

In this section, we compare the CPU’s and GPU’s performance, energy consumption, and different LOCATOR versions with all the optimizations presented in Section 4.2.

The configuration labeled as *BASELINE-CPU* corresponds to the high-performance OpenCV [18] software implementation of ORB feature extraction running our baseline CPU described in Table 3.1, the A72-like ARM CPU. We also compare it with a high-performance desktop CPU (labeled *H-CPU*) and a desktop GPU (labeled *H-GPU*) parametrized in Table 4.2. The ORB implementation features an optimized, vectorized FAST corner detection method. On the other hand, the configuration labeled as *GPU* corresponds to the high-performance OpenCV software implementation of ORB that leverages CUDA on the GPU described in Table 4.2 to speed up computations.

Configurations labeled with *LOCATOR* represent different versions of the accelerator. We use the following nomenclature: *LOCATOR-GN-RM*, where N indicates the group size for BRIEF descriptor computation and M shows the degree of replication. For example, *LOCATOR-G4-R2* indicates a configuration of the ORB accelerator with a group size of 4, i.e., 4 bits of the BRIEF descriptor are computed at a time, whereas the entire rBRIEF unit is replicated two times. We run the same workloads in the CPU and the different LOCATOR configurations, employing the same images and algorithmic parameters such as window size and thresholds.

On the one hand, figure 4.12a shows the speedup of a single replica of the rBRIEF unit compared to other versions. The speedup is calculated considering the latency of the *G1* case in which the unit takes 256 cycles to perform the computations to generate an rBRIEF descriptor. We have considered parallel access groups of 1, 4, and 8 pairs. The figure illustrates the impact of the Selective Replication of banks (SR), Point intensity Cache

Table 4.2: Hardware parameters for LOCATOR, H-CPU, and H-GPU platforms.

	Parameter	Value
LOCATOR	Technology, Frequency	14nm (scaled [141] from 45nm), 400MHz
	Tile width	210 columns
	Target number of features	2000
CPU	Model	Intel(R) Core(TM) i7-7700K
	Number of cores / threads	4 / 8
	Technology, Frequency	14nm, 4.2GHz
	L1, L2, L3	0.25 MiB, 1 MiB, 8 MiB
	Thermal Design Power	91W
GPU	Model	GeForce GTX1080
	CUDA cores	2560
	Memory	8GB
	Thermal Design Power	180W

(PC), and Pipelining (PL) techniques compared to using only the Genetic Algorithm (GA) optimization. The $G4$ and $G8$ versions obtain a $1.12\times$ and $1.25\times$ speedup, respectively, against the corresponding version in which only the GA technique is used. The speedup in $G8$ is more significant than in the $G4$ case. The number of conflicts increases as the group size grows, implying more opportunities for the PC and PL techniques to reduce structural conflicts within groups that the GA technique cannot avoid based on rearranging the accesses order.

On the other hand, figure 4.12b shows a box plot chart for different versions of LOCATOR. The figure depicts how the optimizations consistently improve the processing time when all the optimizations are combined. *LOCATOR-G8-R2* reduces the average processing time per frame by 2.26%. Moreover, the box plot shows how the new techniques shift the distribution of processing latencies towards lower values and reduce its range and dispersion. Note that a configuration with two replicas provides enough ports to access four pairs per cycle with few conflicts, implying that the techniques proposed in this paper have little work to do. We omitted its evaluation for clarity, only presenting results for the scenario with groups of eight pairs (*LOCATOR-G8-R2*).

In the context of self-driving cars, the tail latency of the executed tasks is essential since it can affect the worst case and cause a critical system failure. Figure 4.13a details the

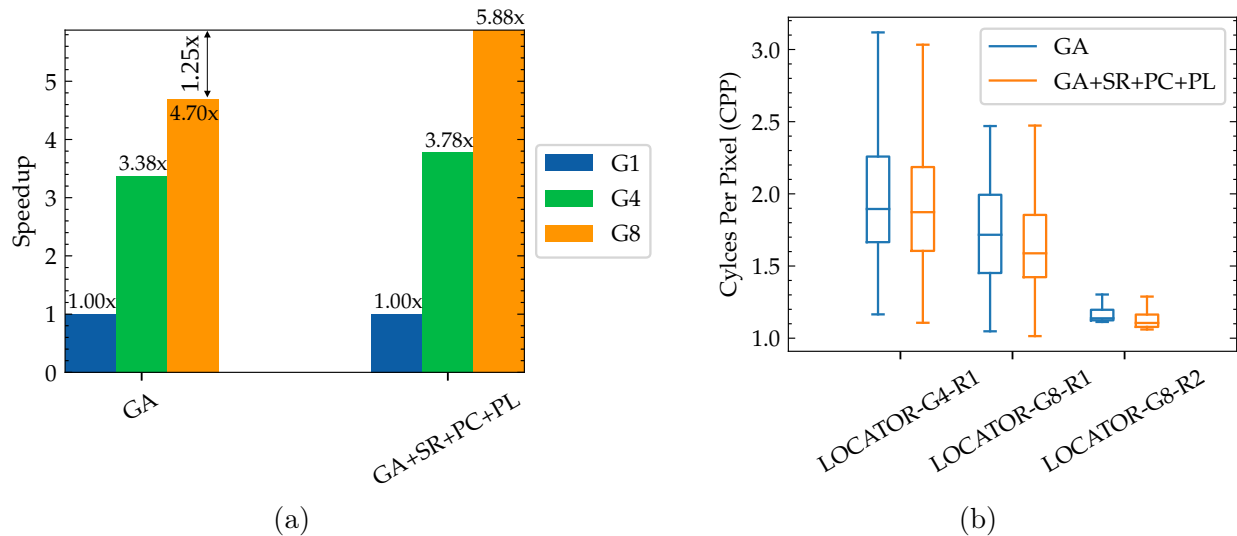


Figure 4.12: (a) rBRIEF replica speedup respect to the baseline implementation ($G1$). GN refers to the maximum number of pairs accessed in parallel. (b) Box plot representing the distribution of LOCATOR processing latencies (Cycles Per Pixel) of the frames of KITTI sequences for different versions of the accelerator.

99th tail latency of the different versions of the previously selected accelerator with the added improvements. All *LOCATOR* versions take advantage of the proposed optimization strategies, improving tail latency compared to the baseline (*BASELINE-G1-R1*). In particular, *LOCATOR-G8-R2* 99th percentile latency is 82.77% lower compared with *BASELINE-G1-R1*.

In addition to the above analysis, we have evaluated LOCATOR’s worst-case latency. The worst case has been tested by generating synthetic frames in which there is a maximum density of features, and their orientation is the one that implies the maximum processing time by the replicas of the rBRIEF unit. The maximum density with non-maximal suppression enabled is 0.25 features per pixel, or one feature every two pixels per dimension. Figure 4.13b shows the results of our experiments.

All versions of the accelerator get a substantial worst-case performance speedup. Due to the high density of features, the possibility of processing the FAST extraction and the description generation in parallel is very advantageous. It explains the great results obtained by LOCATOR versions with more than one replica. *LOCATOR-G8-R2* gets a speedup of $9.32\times$.

The red line in the figure 4.13b indicates the minimum speedup that must be obtained compared to the baseline performance to perform the ORB extraction in Full HD frames in

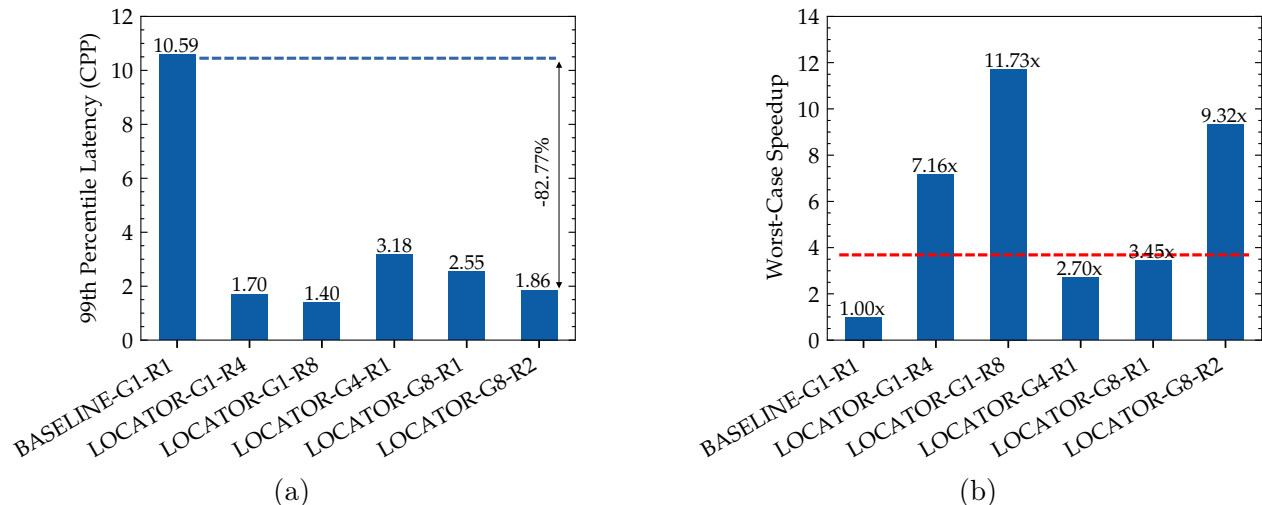


Figure 4.13: (a) 99th Percentile Latency measured in Cycles Per Pixel (CPP) with all the optimizations proposed applied when processing the KITTI dataset. (b) LOCATOR speedup in the worst-case scenario compared to the baseline. The red dotted line indicates the minimum speedup threshold needed to reach our real-time consideration.

100ms or less (that is, 10FPS). We consider this condition to classify the performance as real-time. Therefore, some versions of LOCATOR can process frames satisfying the real-time constraint even in the worst case. In addition, this real-time threshold is exceeded, leaving more slack for other tasks within the localization process.

Besides, figure 4.14 reports the effectiveness of the genetic algorithm (Section 4.2.3) to reduce bank conflicts. The figure shows the number of penalty cycles due to bank conflicts. To compute the extra cycles, we define a lower bound for the latency estimated as the number of accesses of the bank with the most accesses (critical) and assume that the rest of the accesses can be done in parallel. The lower bound is not guaranteed a feasible global optimum, but it allows us to know how much room for improvement could be. Our static scheduling technique (*GA* label in the figure) reduces the penalty cycles due to conflicts by 51.8% and 40.9% for group sizes of 4 and 8 pairs, respectively, as shown in Figure 4.14.

Figure 4.15a shows the speedup achieved by the ORB H-CPU, the H-GPU, and LOCATOR implementations with respect to the *BASELINE-CPU*. All versions of the accelerator employ all the optimizations proposed in this work. Additionally, all systems achieve real-time performance.

The ORB GPU implementation obtains $5.81\times$ speedup. Configurations *LOCATOR-G1-R4* and *LOCATOR-G1-R8* achieve speedups of $4.8\times$ and $8.1\times$ respectively. On the other

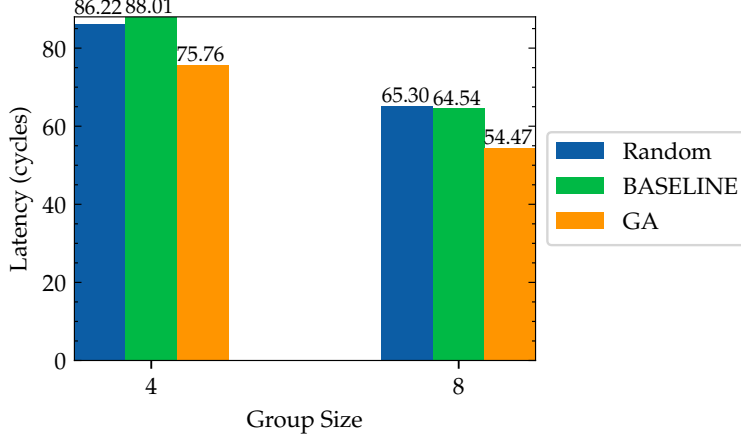


Figure 4.14: Average number of penalty cycles due to bank conflicts when computing an rBRIEF descriptor. The results correspond to the latency incurred when accessing the baseline implementation with and without the GA optimization.

hand, *LOCATOR-G4-R1* and *LOCATOR-G8-R1* deliver $4.82\times$ and $5.66\times$ speedups, respectively. Finally, *LOCATOR-G8-R2* obtains a $16.8\times$, a $8\times$ speedup, and a $1.37\times$ speedup against the A72-like *BASELINE-CPU*, the *H-CPU*, and the *H-GPU*, respectively. The accelerator’s performance is higher as it has a pipeline tailored to the requirements of the ORB feature extraction algorithm. Increasing the group size and rescheduling the pixel pairs for BRIEF computation based on static ordering provides significant benefits, alleviating the need to replicate hardware and its incurred cost. For example, *LOCATOR-G8-R2* is only 1.23% slower than *LOCATOR-G1-R8* with four times fewer replicas. It also obtains $1.66\times$ speedup compared with *LOCATOR-G1-R4* with half replicas.

The accelerator significantly reduces power dissipation, as illustrated in Figure 4.15b. This vast power reduction is due to several reasons. First, the accelerator includes a specifically designed streaming architecture for ORB extraction, exhibiting high throughput and significant data reuse. Second, the improved rBRIEF unit, with the combination of techniques exploited in this work, further improves power dissipation by reducing the required on-chip structures, avoiding conflicts between pairs of pixels, and decreasing the underlying data movements. The power reduction and the performance speedup imply significant energy reductions. *LOCATOR-G8-R2* achieves a $7364\times$, $14597\times$, $9609\times$, and $1.64\times$ average reduction of energy consumption per frame compared with the A72-like *BASELINE-CPU*, the *H-CPU*, the *H-GPU*, and the baseline accelerator, respectively. Moreover, *LOCATOR-G8-R2* consumes 7.7% less energy per frame on average than *LOCATOR-G1-R8*, a more expensive configuration that presents a high replication degree.

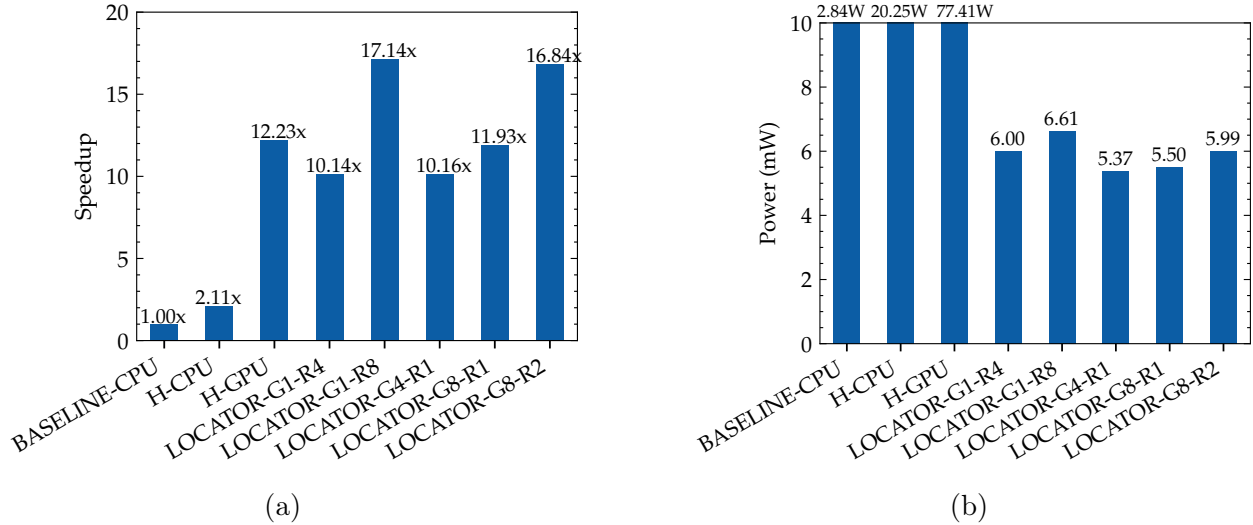


Figure 4.15: (a) speedup achieved by the accelerator compared with the BASELINE-CPU. (b) Power dissipation of ORB FE (Feature Extraction and Matching) across different platforms and the accelerator.

To conclude, the experimental results show the benefits of the proposed architecture and the combination of techniques to optimize the rBRIEF unit. The impact of the new optimizations in energy consumption and area is negligible since the selective replication of ports counteracts their costs, obtaining a more energy-efficient final result. Although the impact on average latency is modest, there is an improvement in tail latency and worst-case scenario latency without needing as high a replication level as in other versions of LOCATOR.

LOCATOR-G8-R2 is the best configuration tested in our experiments since it is only 1.23% slower than *LOCATOR-G1-R8*, the most performant configuration tested, requiring four times fewer replicas. It also achieves significantly lower power and area footprint. The combination of optimizations presented translates into a $1.67\times$ speedup against the *H-CPU* in the overall execution time. The accelerator achieves 9609 \times average reduction of energy consumption per frame compared with the ORB *H-GPU* implementation. Finally, *LOCATOR-G8-R2* reduces 99th percentile latency an 82.77% and obtains a speedup of 9.32 \times in the worst-case scenario compared with the baseline accelerator, achieving real-time even in critical circumstances.

4.4 Conclusions

This chapter described *LOCATOR*: a **L**ow-power **ORB** **aC**celerator for **AuT**onom**O**us **caR**s for feature extraction, a key component of camera-based localization for self-driving cars. We propose a novel solution to implement rBRIEF descriptor computation in hardware that, unlike previous proposals, achieves the same accuracy as reference software implementations, not sacrificing accuracy for the sake of simpler hardware. The pipelined design of the rBRIEF unit evaluates multiple pairs of pixels simultaneously following an order based on static scheduling that minimizes the bank conflicts for any angle. *LOCATOR* hides the conflicts of pair accesses with pipelining and avoids them through a statically managed duplication cache whose cost is mitigated by applying selective replication of ports. Our experimental results show that *LOCATOR* achieves a speedup of $16.8\times$, a $8\times$ speedup, and a $1.37\times$ speedup against the A72-like *BASELINE-CPU*, the *H-CPU*, and the *H-GPU*, respectively.

5

SLIDEX

This chapter presents SLIDEX (SLIDing window EXtension for image processing), a novel high-performance and energy-efficient vector ISA (Instruction Set Architecture) extension with specialized instructions to natively support SWP (Sliding Window Processing) in conventional CPUs and optimize convolutions and stencil operations.

5.1 A Novel Architecture for Sliding Window Processing

As demonstrated in Chapter 4, a specialized ASIC accelerator can surpass a CPU's energy efficiency by several orders of magnitude. However, the rapidly evolving nature of technology can render these specialized solutions quickly outdated. Therefore, in this chapter, we explore more programmable and versatile solutions to accommodate current and unforeseen future algorithms better.

Another strategy to increase efficiency while preserving adaptability involves leveraging Data Level Parallelism (DLP) through the SIMD [45] support available in the CPUs of most embedded and real-time systems that commonly host emerging CV (Continuous Vision)

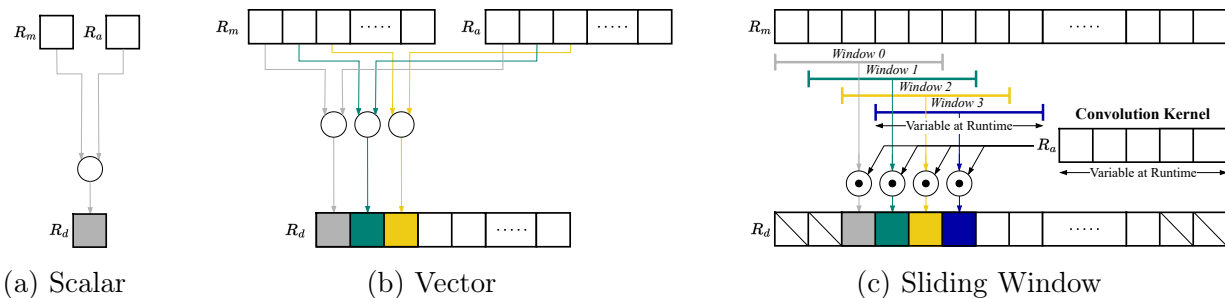


Figure 5.1: Different execution models utilizing registers R_m and R_a for computing output in register R_d . SLIDEX instructions process multiple windows in parallel to compute the neighborhood operation (e.g., convolution) reduced results.

applications. SIMD (Single Instruction, Multiple Data) units offer high performance and energy efficiency by amortizing instruction management costs (fetch, decode, issue) across multiple operations. Hameed et al. [61] estimate that about 94% of a modern CPU’s total energy goes towards supporting instruction supply generality, with only around 6% devoted to ALU (Arithmetic Logic Unit) computations when executing the instructions. Nowadays, most mainstream CPUs feature SIMD Vector Processing Units (VPUs) that support vector instructions such as NEON [13], SVE [140], Intel AVX [71], and RISC-V [130].

Convolutions and stencils are core primitive operations critical for CV applications. Both operations analyze each image pixel, considering a window of neighboring pixels that substantially overlap between adjacent elements. Unfortunately, current SIMD solutions generally manipulate the elements within a vector register independently of each other. While this provides excellent generality, it implies that, for example, for a simple 1D convolution, the programmer needs to load the data, perform element-wise multiplication with the first kernel component, execute a data shift, conduct a multiplication with the subsequent kernel component, accumulate the results, and so on. As a result, there is a notable gap between the available DLP (Data Level Parallelism) and its actual utilization in these crucial image-processing tasks.

Furthermore, an ideal system that could eliminate all energy overhead spent to support the generality of CPUs would be around 16x more energy efficient than the baseline, assuming this overhead is 94% [61]. However, this efficiency level does not match the over 1000x efficiency in typical accelerators. To bridge this gap, the CPU requires instructions with higher semantic content. Moreover, reducing the number of memory and register file accesses, which typically account for about 60% of a standard CPU’s energy use, is also vital to pushing current efficiency limits.

Considering these premises, we introduce SLIDEX, a novel high-performance and energy-efficient vector ISA extension with specialized instructions to natively perform convolutions and stencil operations over vector register employing a new execution model, named SWP, in conventional CPUs. SWP is our proposed SIMD execution model that reinterprets the layout of the vector registers not as a sequence of isolated elements but as a sequence of overlapped windows of elements. Figure 5.1 illustrates the difference of scalar, vector, and SWP execution models. Unlike the traditional vector SIMD paradigm (Figure 5.1b), the SWP model can perform multiple parallel computations sourced from the overlapping, shifted window copies of the elements within a vector register. Figure 5.1c depicts how an SWP instruction can interpret a register as multiple overlapping windows and perform a 1D convolution to the different windows through a single instruction.

To efficiently exploit SWP, we propose a lightweight microarchitecture extension for standard VPUs (Vector Processing Units), primarily involving the modification of the datapath to implement a 2D array of ALUs to exploit the arisen DLP when considering the operand reuse from the overlapping windows. As presented in the following section, SLIDEX supports variable window sizes at runtime without complex interconnections, thus ensuring scalability.

5.2 Implementation

This section describes the architecture extension introduced by SLIDEX to support SWP and a microarchitecture that implements it.

5.2.1 Overview

We engineer SLIDEX as a cost-effective, minimally invasive vector extension for conventional SIMD processors, such as ARM Neon. This design strategically leverages existing resources in commercial cores, such as vector register files and ALUs. Given the widespread presence of SIMD engines in edge and embedded domains, our approach requires minimal integration changes, facilitating industry adoption of SLIDEX. To integrate SLIDEX, we incorporate support for decoding new instructions, along with various control and predicate registers. Additionally, we modify the datapath to include specialized ALUs capable of processing multiple sliding windows and executing various operations per window in parallel, thereby enhancing the DLP exploited by the baseline vector engine.

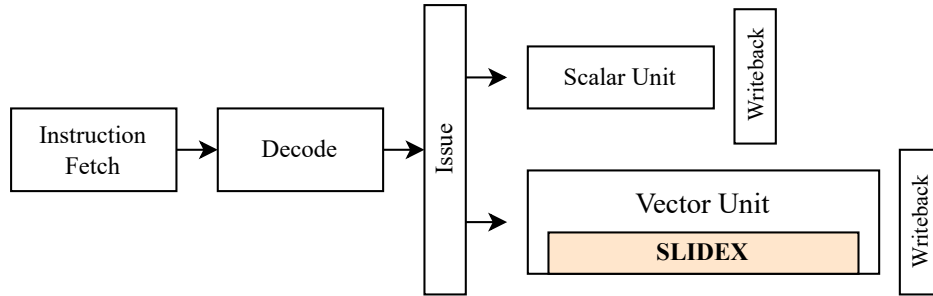


Figure 5.2: A high-level system overview. The SLIDEX unit is a pipeline placed inside the Execution stage of the in-order CPU.

Figure 5.2 shows a high-level diagram of the tightly coupled integration of the SLIDEX augmentation in a conventional in-order CPU pipeline to support SWP, which we detail in the next section.

5.2.2 Sliding Window Processing

Sliding Window Processing (SWP) is a new SIMD processing model with instructions that natively operate on vector registers that store multiple overlapped windows. Exploiting the windows overlap amplifies the potential number of operations per instruction compared with an equivalent vector instruction without increasing the vector length. The local elements of each window are processed in parallel and then reduced and written to the output.

This work presents the canonical form of SWP, 1D SWP, which operates on one-dimensional vector registers. 1D SWP requires fewer changes to conventional vector engines than, for example, 2D SWP. Additionally, 1D SWP is enough to saturate the memory system of our baseline, as Section 5.3 shows.

Moreover, 1D SWP supports 2D operations such as 2D non-separable convolutions by partially accumulating the results of multiple 1D SWP operations (see an example in subsection 5.2.5).

SWP achieves several benefits compared with the traditional scalar and vector processing models, with only a minimal increase in hardware requirements. The key advantages of SWP to push SIMD performance on modern CPUs for image processing tasks are:

- *SIMD instructions with richer semantics*: Programs require fewer SWP instructions than conventional ISAs to implement neighborhood operations, extending SIMD ben-

Table 5.1: SLIDEX architectural visible state.

Identifier	Size	Interpretation
p[0-15]	$VL \times 1bit$	PRF registers to store the predicates.
r_wir	$VL \times 48bits$	Register to store intermediate wide results.
r_wsize	$1 \times 32bits$	Register to store the window size.

efits such as instruction supply reduction. In addition, the richer semantics translate to more exploited DLP.

- *Decrease of reads and alignments of input operands:* Specific hardware can efficiently route window data to processing elements by taking advantage of the overlap between windows. A specific routing for sliding windows translates into fewer register file and memory accesses and fewer instructions to align or organize the data.
- *Less intermediate result writes:* Each instruction performs a partial or total neighborhood operation on multiple windows, reducing or accumulating the results. Other approaches require saving the intermediate results in the register file or memory before the reduction.

SLIDEX SWP ISA extension provides specific hardware to achieve these benefits, increasing CPU performance and energy efficiency. The following sections describe the lightweight changes required to support SWP on existing vector processors.

5.2.3 Architectural State

In addition to the typical architectural state of the vector engine of the CPU, SLIDEX includes a Predicate Register File (PRF), a Wide Intermediate Register (WIR), and a control register that holds the WSIZE (Window Size) value, as shown in Table 5.1. Similar to the baseline vector registers, some of these new specialized registers scale with the VL (Vector Length).

The PRF (registers p0-p15) store predicates that applications can utilize to mask the operations performed by SLIDEX instructions. The WIR (r_wir) register provides additional temporary storage to accumulate intermediate results before reducing the register to fewer bits. This process enables operations to use higher precision formats before reducing the

Table 5.2: SLIDEX new proposed instructions.

Instruction	Description
<i>WCONV</i> Rd, Rn, Rm, Ra	Performs the convolution using the kernel coefficients stored in the vector register <i>Ra</i> over the data stored in the vector register <i>Rm</i> , adds the values from <i>Rn</i> , and places the result in <i>Rd</i> .
<i>WFAST</i> (p0) Rd, Rn, Rm, Ra, #TH	Performs the partial FAST (Features from Accelerated Segment Test) extraction comparing the elements from the register <i>Rm</i> with the candidate corners stored in <i>Ra</i> using the threshold indicated in the immediate value <i>#TH</i> and fuses the partial result with the contents of <i>Rn</i> storing the result in <i>Rd</i> . <i>p0</i> predicates and masks the execution.
<i>WFASTRED</i> Rd, r_wid	Reduce the <i>WFAST</i> state stored in <i>R_wid</i> and places the result in <i>Rd</i> : zero or the score if the pixel is a corner.

results to a byte. Finally, the programmer can actively control the `WSIZE` used by SLIDEX instructions at runtime through the control register (`r_wsize`).

5.2.4 Instruction Set Extension

Table 5.2 presents the new instructions introduced by SLIDEX to leverage Sliding Window Processing for convolutions (e.g., Gaussian filtering) and FAST keypoint detection.

5.2.5 SLIDEX Programability

To facilitate programmability and similarly to the manually optimized libraries provided by most vendors [102, 72], we have created a SLIDEX-optimized library with representative kernels for image processing, such as convolution and FAST functions programmed using the SLIDEX instructions. Additionally, future compilers may support SLIDEX by providing compiler intrinsic functions. We will illustrate the use of the instructions dedicated to convolution and FAST tasks through simplified examples.

The *WCONV* instruction performs 1D convolution. As presented in Figure 5.1c, the instruction processes multiple windows in parallel, applying the dot product with the coefficients of the convolution operator. The elements that appear crossed out at the beginning and end of the register are part of the padding, and their value is not defined. Listing 5.1 shows how to perform a separable 2D convolution using *WCONV*. First, the program appropriately sets the window size (`WSIZE=5`) by loading the correct value on register `r_wsize`. Then, it performs horizontal 1D convolution using one instruction per row. Before con-

```

1 LDR r_wsize, 5 # Set Kernel size
2 VLD {V15}, [R1] # Load horizontal kernel
3 # Horizontal Convolution
4 VLD {V0}, [R0]! # Load first row
5 WCONV V0, VZR, V0, V15
6 WCONV V1, VZR, V1, V15
7 ...
8 # Macro to transpose the image block
9 TRANSPOSE_MATRIX V0, ..., V15
10
11 VLD {V15}, [R2] # Load vertical kernel
12 # Vertical Convolution
13 WCONV V0, VZR, V0, V15
14 WCONV V1, VZR, V1, V15
15 ...
16 # Store the result
17 VST {V0}, [R3]! # Store the first column
18 ...

```

Listing 5.1: 2D Separable Convolution example. *VZR* is the zero register.

```

1 LDR r_wsize, 7 # Window size for FAST
2 VMOV r_wir, #0 # Initialize r_wir
3 VMOV p0, #0b0011100
4 # Perform WFAST operations
5 WFAST (p0) r_wir, r_wir, V1, #10
6 ...
7 WFAST (p1) r_wir, r_wir, V5, #10
8 WFAST (p0) r_wir, r_wir, V6, #10
9 # Reduce the WFAST state
10 WFASTRED V15, r_wir

```

Listing 5.2: FAST feature extraction pseudoassembly example.

ducting the vertical pass, it is necessary to transpose the output of the results. Then, the algorithm repeats the convolution process for the vertical operation.

Non-separable 2D convolutions can be efficiently executed by splitting them into successive 1D convolutions for each kernel row and accumulating them to obtain the final 2D convolution. To implement this strategy with SLIDEX, the programmer can apply a *WCONV* instruction for each kernel row (*WSIZE* in total), generating the final results for an entire image row.

Similar to the non-separable convolution, stencil operations like FAST require multiple *WFAST* instructions to generate the corner classification of the elements of each image block row. In particular, since FAST operates using a *WSIZE*=7 (see Figure 2.7b), the programmer should use seven *WFAST* instructions to process one image row. These *WFAST* instructions check whether the pixels in the Bresenham circle, shown in the figure, are brighter or darker than the center pixel by the corresponding threshold indicated in the immediate field of the instruction. It stores the comparisons in the destination register. In addition, the instruction simultaneously computes the corner score. The score is the sum of absolute differences between the intensity of the candidate and 16 surrounding pixel values. Listing 5.2 shows how the *WFAST* instruction employs the *WIR* register for intermediate results, as it provides enough bits to store the state of the FAST operation. SLIDEX sliding window unit provides predication or masking capabilities that can prevent some window elements from participating in the neighborhood operation. In the case of *WFAST*, the predicate allows selecting the pixels of the Bresenham circumference (e.g., the first predicate will be 0011100X...X corresponding to masking all pixels but the 16, 1, and 2 shown in Figure 2.7b). Lastly, the final *WFAST* instruction (line 10) generates the final state with all the information required to classify the pixels. *WFASTRED* instruction processes the contents of the *WID* register, writing every pixel score to the destination register if the corner satisfies the FAST condition or a zero otherwise.

5.2.6 SLIDEX Execution Unit

In this work, we propose a cost-effective vector extension that implements the SLIDEX instructions to leverage 1D SWP as a proof of concept.

Figure 5.3 presents the main component of our proof of concept design. This design extends the baseline storage composed of the Vector Register File (VRF) with additional elements: a Predicate Register File (PRF), the *WIR*, and control registers (detailed in

Section 5.2.3). The other main component is the SLIDEX Execution unit, whose design we discuss below in this section.

Before discussing the implementation details, we define one important hardware parameter of the microarchitecture. The PWO (Parallel Window Operations) parameter is the maximum number of partial operations of each window that the SLIDEX Execution Unit can perform in parallel each cycle. It is related to the VL parameter since it indicates the maximum number of windows computed in parallel each cycle. Together, VL and PWO embody how SLIDEX allows exploiting 1D SWP and determines the peak performance.

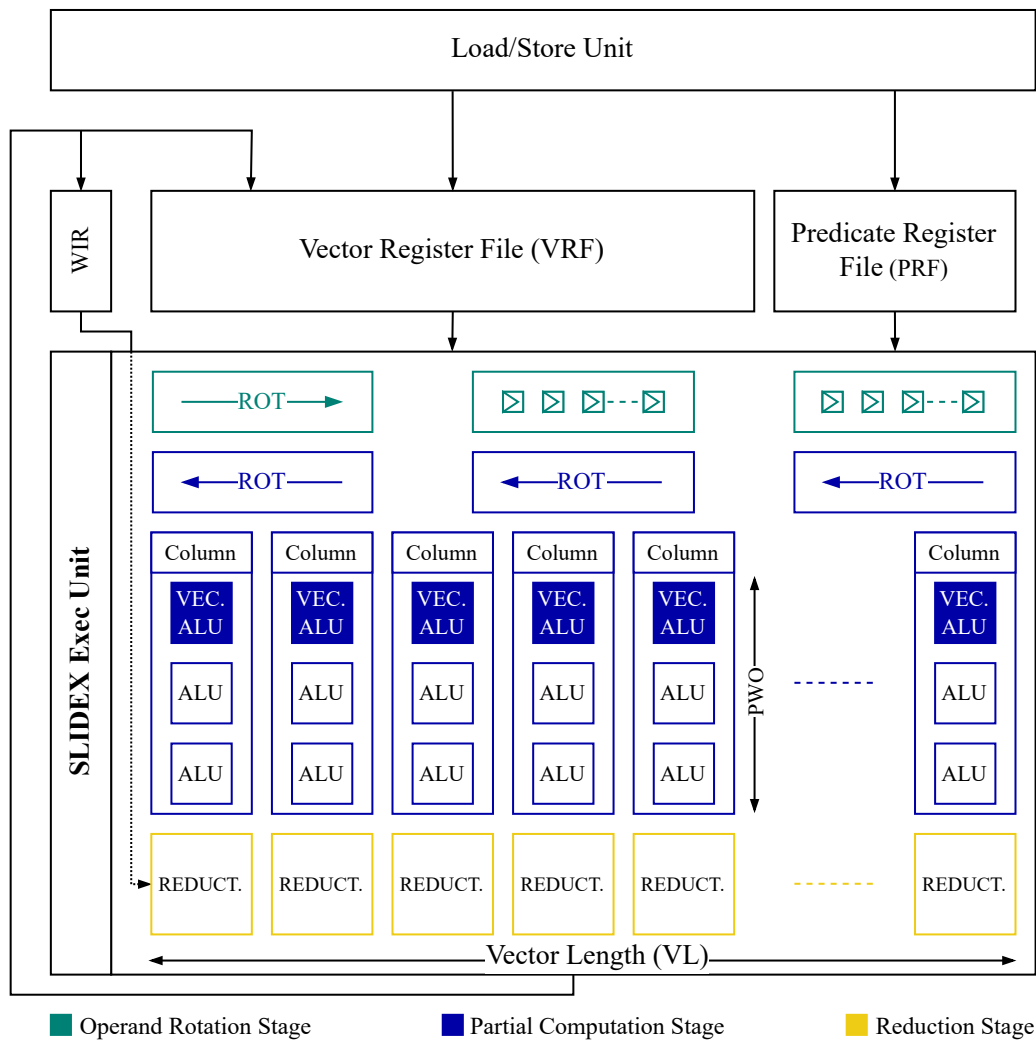


Figure 5.3: General overview of the SLIDEX augmentations showing the state (VRF, PRF) visible to the programmer (Table 5.1) and the SLIDEX Execution unit. ROT modules rotate the input in the direction of the arrow.

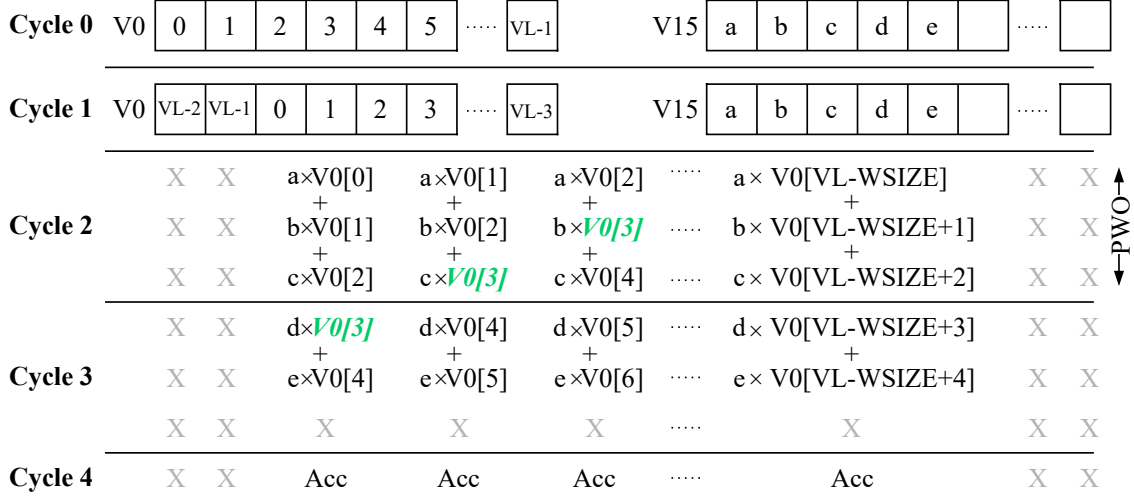


Figure 5.4: *WCONV* V0, VZR, V0, V15 simplified cycle chronogram (WSIZE=5). Note how, for example, the register value V0[3] slides in space across the ALU inputs (Cycle 2) and in time at the cycle change.

In order to implement 1D SWP, the SLIDEX Execution unit has a three-stage pipeline, namely: i) ORS (Operand Rotation Stage), ii) PCS (Partial Computation Stage), and iii) RS (Reduction Stage).

The Operand Rotation stage gathers the instruction operands. Then, it rotates (ROT→) the register Rm (see Table 5.2) that always contains the row of pixels of the image block loaded in the VRF over which the unit applies the neighborhood operations. The rotation is required to align the first element of each window with the corresponding output element for the final result of the window. Thus, the unit shifts the elements of Rm $\frac{WSIZE-1}{2}$ positions to the right. Figure 5.5a shows the fundamental generic component of the variable step shift register used in the ORS. After a first register reading cycle (Rm_i element coming via the corresponding I[i] input), the unit controller determines the maximum step to shift the Rm register to reach the required position in successive ones. ORS holds the rest of the instruction’s operands in input latches. Figure 5.4 depicts a chronogram for *WCONV* V0, VZR, V0, V15 instruction (WSIZE=5). The unit reads V0 in Cycle 0 and rotates it in Cycle 1, aligning the V0[0] (first window element) position with the position of the center pixel of the window (V0[3]).

The Partial Computation Stage performs the local operations on the pixels of the windows. PCS can process a maximum of VL windows and a maximum of PWO operations from each window in parallel. The unit initially routes the Rm (rotated), Ra, and p0 (if any)

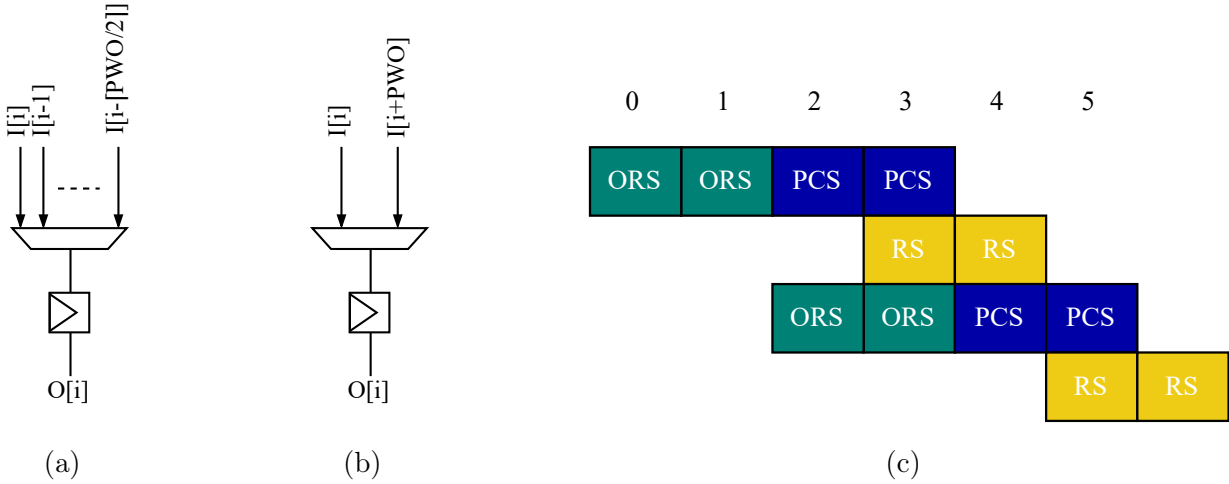


Figure 5.5: (5.5a) general diagram of the variable step shift register element used in ORS. (5.5b) general diagram of the shift register element used in PCS. (5.5c) Pipeline diagram of SLIDEX Exec Unit with $PWO=3$ and $WSIZE=5$. The pipeline can completely hide the ORS latency.

from the ORS through a shift register that can rotate ($\leftarrow ROT$) the instruction operands PWO positions to the left in each cycle. Figure 5.5b illustrates the design of the generic element of said mechanism. $VL \times PWO$ ALUs process the output values of the shift registers. These ALUs appear grouped in columns of PWO elements in Figure 5.6a. Each column processes the pixels of the same window and sends the output results to the next stage. Note that the columns use the already existing ALUs from the baseline vector processor (denoted with solid blue color in Figures 5.3 and 5.6a). Our proposal augments it with simpler ALUs specialized for the typical image processing operations. The columns use a ring interconnection scheme (i.e., the first column connected with the last), each receiving elements i , $i + 1$, and $i + 2$ from register R_m .

Furthermore, each ALU has a multiplexer to control the operand R_a input according to the operation type. For the *WCONV* instruction, the multiplexers of the first ALU of each column select the $R_a[0]$ element from the horizontal bus. For the case of the *WFAST* instruction, the unit selects the $R_a[i]$ coming from the vertical bus. Thus, the difference between the two modes of operation lies in whether the ALUs use the horizontal or vertical bus. In the first case, the three ALUs of all columns (windows) receive the same three elements (e.g., convolution coefficients). In the second, the three ALUs of each column receive the same value (e.g., the corresponding center pixel in *WFAST*). The PCS column interconnection network exploits window overlapping to efficiently route the ALU operands.

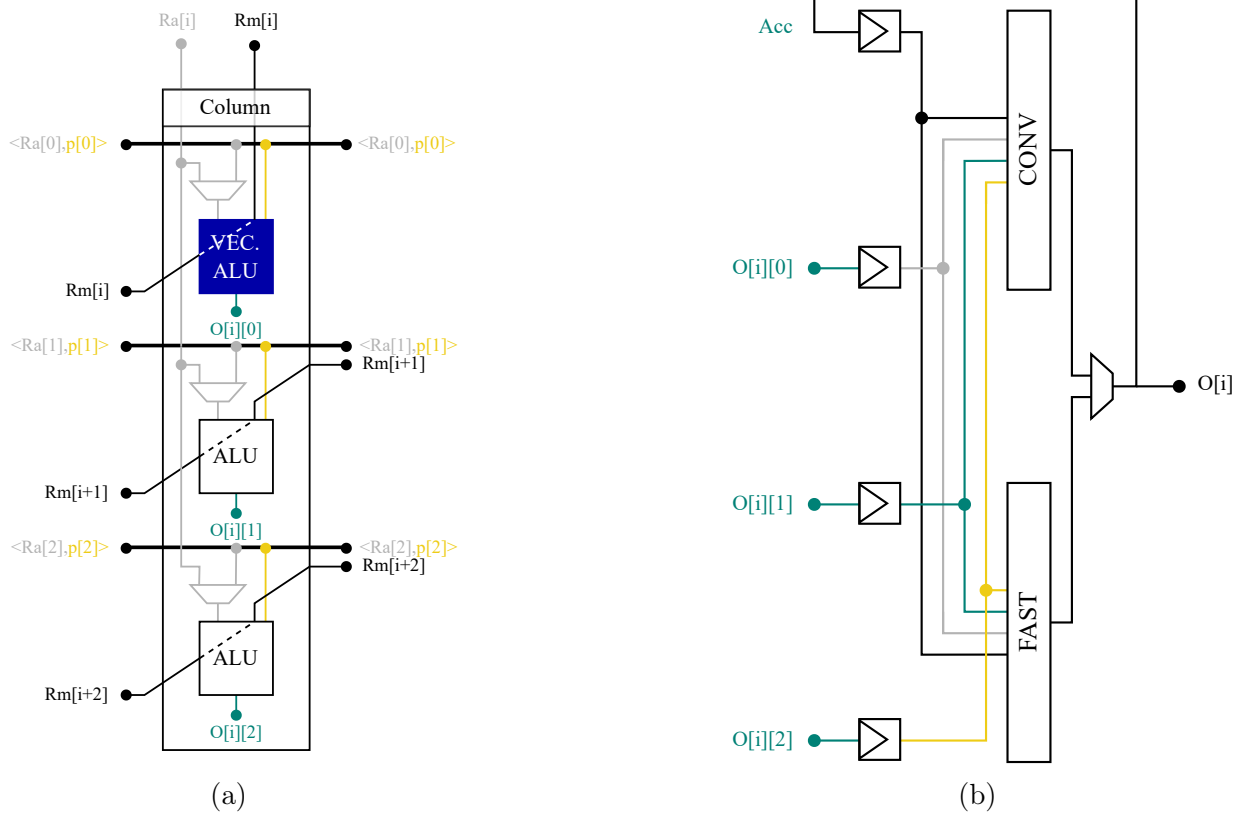


Figure 5.6: (5.6a) SLIDEX column able to perform PWO=3 parallel window operations each cycle. (5.6b) SLIDEX reduction unit.

In Figure 5.4, cycles 2 and 3 show in more detail an example of how this interconnection network routes the window pixels and the convolution operator coefficients to the ALUs in the case of *WCONV* instruction.

The Reduction Stage, shown in Figure 5.6b, reduces each column's output of the ALUs. In the first RS cycle, the unit initializes the internal accumulator (Acc) register with the corresponding Rn value. Acc register has the same number of bits as the r_wir register, allowing intermediate accumulations to have a larger state before an eventual reduction. Furthermore, the unit has Rn operand chaining support, which is especially interesting for 2D window operations such as the one exemplified in Listing 5.2. The stage saves one register read in each intermediate *WFAST* instruction since it can immediately use the correct r_wir value already in the internal accumulator.

The pipeline design described can hide the latency of ORS and RS for any PWO value. The ORS has to shift the register Rm $\frac{WSIZE-1}{2} \sim \frac{WSIZE}{2}$ positions. In contrast, the Partial

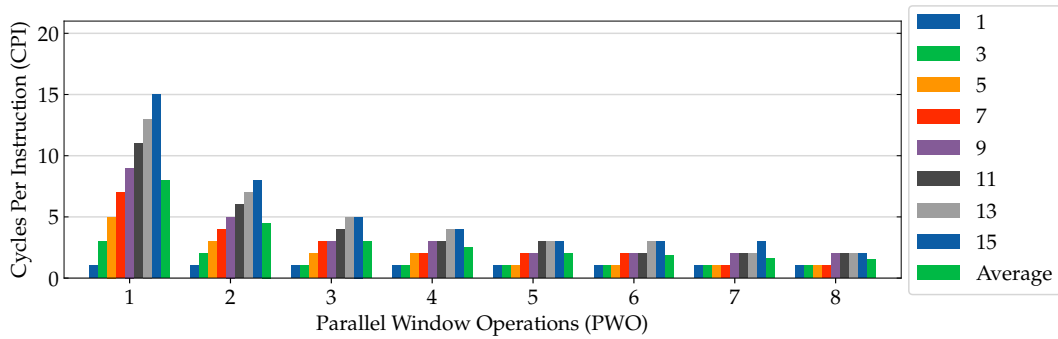
Computation stage has to shift the input operands WSIZE positions, taking $\lceil \frac{WSIZE}{PWO} \rceil$ cycles to complete its operation. With the variable step shift register of our design (Figure 5.5a), ORS can shift with a maximum step of up to $\lceil \frac{PWO}{2} \rceil$ positions taking approximately the same number of cycles as PCS to finalize. RS also overlaps with PCS, implying that the pipeline reaches the throughput imposed by the critical stage PCS. Figure 5.5c illustrates the balanced SLIDEX Execution Unit pipeline diagram for PWO=3 processing a window operation with WSIZE=5.

Finally, we discuss the performance, power, and area tradeoffs associated with varying the VL and PWO parameters. First, augmenting VL implies increasing the number of shift registers, columns, and reduction subunits. All these elements have a fixed cost that depends on PWO and not VL, making the solution scalable. Second, as shown in Figure 5.7a, an increase in PWO improves SLIDEX Execution unit performance for the window sizes (WSIZE) typically used in image processing. However, this leads to a linear increase in the cost of the solution and degrades the unit’s efficiency. Figure 5.7b shows the sensitivity of the ALUs’ utilization to the PWO parameter. The chronogram in Figure 5.4 illustrates how, in Cycle 3, the third ALU of each column does not do any valuable work. Based on our experiments, we decided to use a value of PWO=3 for our proposal. The design achieves 71% of the ideal (CPI=1) gain at the expense of a 16% average loss of utilization w.r.t. to PWO=1.

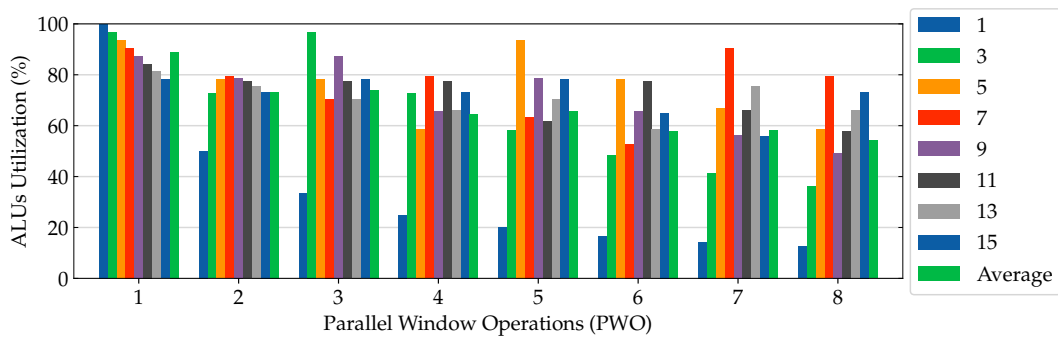
Figure 5.7c shows the effective number of ALU operations per SLIDEX instruction, normalized by the number of operations performed by a vector instruction. For example, on average, SLIDEX-16 performs more than twice as many operations per instruction as a vector instruction. It is worth noting that these operations are also more complex, but we are simplifying the explanation for clarity. The increased number of operations offsets the energy cost of the dominant instruction supply, making ALU underutilization energy inefficiency marginal.

5.3 Experimental Results

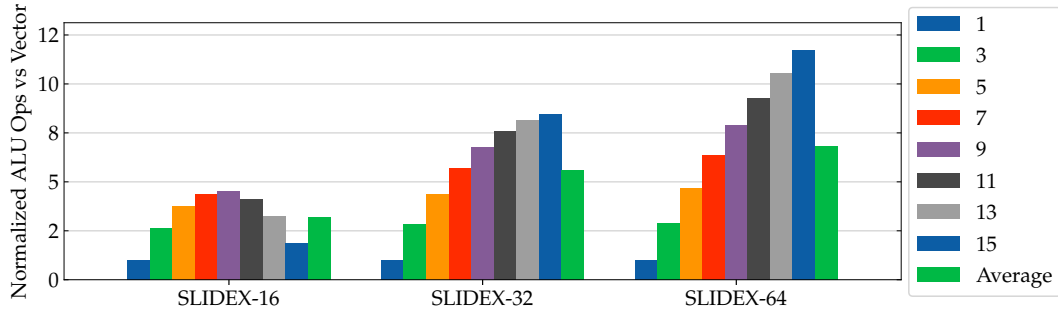
This section evaluates the performance of scalar and vector implementations of the visual localization system’s frontend under study. We then compare these results with SLIDEX under the same baseline conditions, notably using the same memory bandwidth and L/S unit.



(a)



(b)



(c)

Figure 5.7: (5.7a) Cycles Per Instruction (CPI) of SLIDEX Exec. unit for different PWO and WSIZE values. (5.7b) SLIDEX Exec. unit ALU utilization percentage for different PWO values and window sizes (WSIZE) assuming a fixed VL=64. (5.7c) Normalized ALU operations per instruction performed by SLIDEX-16, -32, and -64 solutions for PWO=3 and different WSIZE values relative to the operations per instruction performed by a vector processor with corresponding VL value. The nomenclature SLIDEX-VL specifies the VL parameter of each SLIDEX version.

5.3.1 Baseline Characterization

For our evaluation, we select an A55-like CPU as the baseline. We assess the execution time per frame for various components of our localization engine as introduced in Section 2.2.6. The breakdown of the FE (Feature Extraction and Matching) stage comprises the FAST feature detection (FAST), the Gaussian filtering of pyramid levels (GAUSS), the feature descriptor generation (DESC), and specific feature manipulation tasks (OTHER). Another crucial part of the algorithm involves using the detected features to track the camera (TRACK). Figure 5.8 illustrates the relative execution times of these components, highlighting that FAST and GAUSS together account for more than 40% of the total execution time. This significant proportion underscores their potential to demonstrate the benefits of SLIDEX.

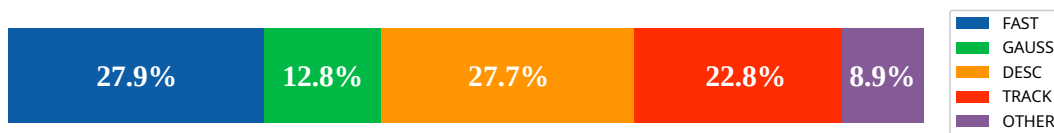


Figure 5.8: Relative end-to-end execution time breakdown of the main ORB-SLAM tasks.

ORB-SLAM relies heavily on the OpenCV library [18] for image manipulation. However, the standard implementation does not fully utilize the small core of our baseline, as the library’s Gaussian convolution and FAST implementations are not optimized for ARM vectorization, and automatic vectorization by the compiler is limited. We have optimized the software by vectorizing the Gaussian convolution and FAST feature extraction, achieving speedups of approximately $\sim 11.1\times$ and $\sim 3.3\times$, respectively. We use this vectorized version as the primary reference for performance and energy comparisons.

We evaluate the performance and energy efficiency of three versions of the visual frontend of ORB-SLAM:

- **OpenCV (CV)**: Original ARM-compiled version of the visual frontend of the application under study. We use OpenCV [18] library for image pre-processing tasks, which lacks optimal vectorization for this platform.
- **Vectorized (VEC)**: A modified version implemented with NEON vector instructions for image processing tasks.
- **Our proposal (SLIDEX-VL)**: A modified version using SLIDEX vector extensions. We denote different VL configurations as SLIDEX-VL.

All versions use the same L/S unit configuration as the baseline’s NEON unit, designed with a PWO of three. Given that the CPU’s NEON instructions support a VL of sixteen, the SLIDEX-16 setup is particularly significant for comparison.

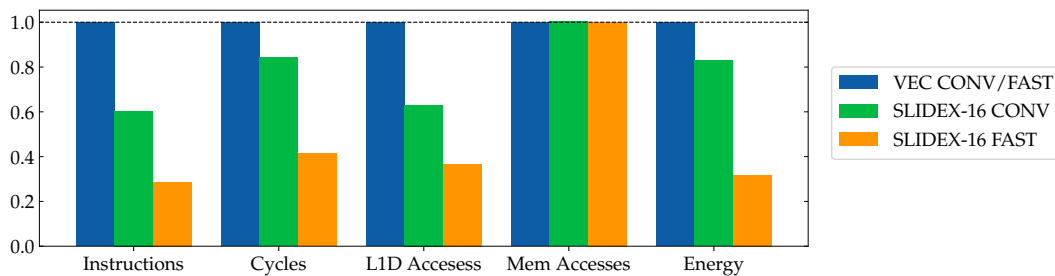
5.3.2 Performance Analysis

Figure 5.9a, Figure 5.9b, and Figure 5.9c present the primary metrics of interest extracted during the execution of the Gaussian convolution and FAST feature extraction. The results correspond with an average across KITTI sequence executions and normalized w.r.t. the vector implementation results. We compare the execution of each part using the vector extensions with the implementations using the SLIDEX extensions.

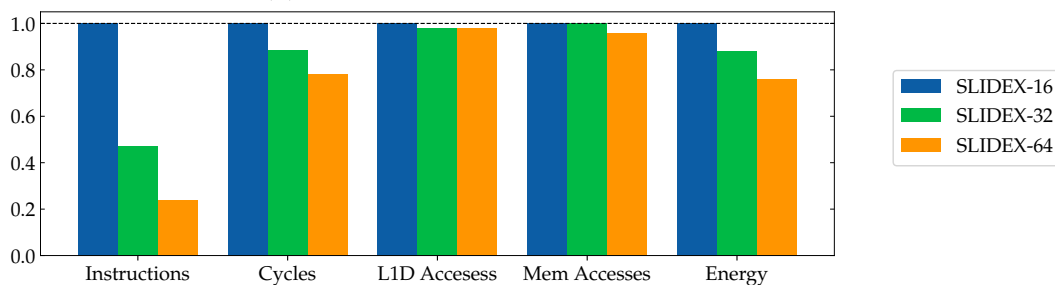
Figure 5.9a illustrates how, in all cases and for both tasks, we can observe a substantial decrease in the number of instructions, cycles, and data cache accesses compared to the vector implementation. There are several explanations for these results. First, NEON exploits a maximum DLP given by the $VL=16$ parameter, whereas SLIDEX can exploit a maximum DLP of $VL \times PWO$, which allows SLIDEX implementation to reduce the number of cycles. Second, SLIDEX exploits this parallelism through efficient operand routing that takes advantage of the inherent overlapping between the elements traversed using the sliding window pattern. Therefore, SLIDEX extensions avoid using the instructions to align, load, or arrange data in the vectorized code, which reduces the number of instructions and cache accesses. For example, the vectorized FAST implementation performs sixteen unaligned vector loads of the pixels of the Bresenhan circumference. Once loaded into the vector register file, the program can perform vector comparisons to classify multiple corners in parallel, requiring additional instructions to compare and obtain the final results. As opposed, SLIDEX FAST implementations do not require the intervention of the cache or any other external mechanism to align the data. Finally, the last key factor that allows SLIDEX extensions to outperform the vectorized case lies in the higher semantics of the instructions that translate to a decrease in execution time and energy. The high semantic value of the instructions comes from expressing 1D SWP and the specialization (*WCONV* and *WFAST*).

Noteworthy, main memory accesses do not change substantially in any cases, indicating that the cache can contain the application’s working set in all the cases since both the vectorized code and the SLIDEX code use the same data.

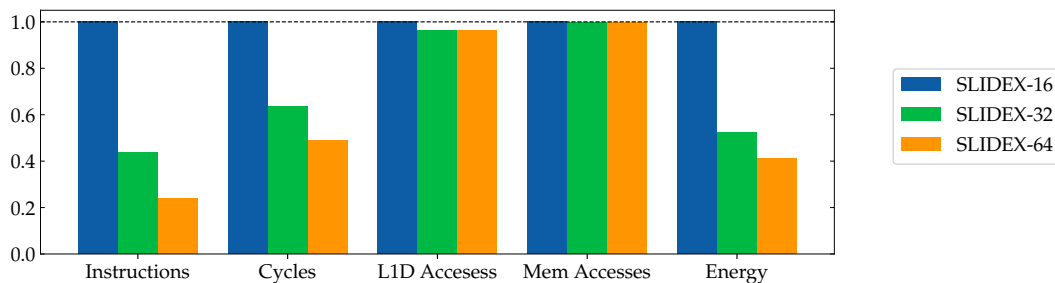
If we compare the difference between convolution and FAST, we notice that the improvement is more significant in the latter case. The fact that SLIDEX allows the classi-



(a) Comparison of vector with SLIDEX-16.



(b) Comparison of SLIDEX flavors for the Gaussian convolution.



(c) Comparison of SLIDEX flavors for the FAST feature extraction.

Figure 5.9: Normalized main hardware metrics extracted during the execution of the Gaussian convolution and FAST feature extraction comparing the behavior of the vectorized version and that of different flavors of SLIDEX.

fication of multiple pixels according to the FAST algorithm with only eight instructions (0.8 instructions per pixel in SLIDEX-16 if we ignore load and store instructions) compared to the tens or hundreds of the vectorized version mitigates the fact that the complexity of the convolution (1D) is less than that of FAST (2D).

If we compare the different versions of SLIDEX looking at Figures 5.9b and 5.9c, we observe that by raising VL, the performance increases, although the rate of improvement stagnates. The improvement comes from increasing the DLP by processing more windows in parallel. In addition, a higher VL decreases the impact of image tile padding overhead, thus increasing utilization and efficiency. A simulation with a perfect memory system (instant access time) shows that SLIDEX-64 obtains more than 4x speedup compared with SLIDEX-16. However, with a realistic memory model, increasing VL incurs diminishing returns in the platform under study. SLIDEX-16 almost saturates the memory system. SLIDEX-32 and SLIDEX-64 units process image rows much faster than the system can provide. Thus, they do not obtain significant performance benefits compared with SLIDEX-16 due to the memory access bottleneck that throttles the CPU’s overall performance.

Furthermore, we have evaluated ORB-SLAM end-to-end. Figure 5.10 shows a box plot depicting the different implementations’ end-to-end effects. The end-to-end speedup is $\sim 1.2\times$ for the VL=16 version of SLIDEX and, following Amdahl’s law, as the VL rises, the rest of the tasks begin to weigh more in the global computation and the overall speedup stagnates. We believe that SLIDEX can be a great alternative to classic vector processing units for image processing, as illustrated by the fact that the speedup obtained for FAST extraction is $2.4\times$ $4.1\times$ and $5.9\times$ for SLIDEX-16, SLIDEX-32 and SLIDEX-64, respectively.

Despite the benefits of increasing the VL, the memory hierarchy cannot sustain the hungry data consumption rate of the SLIDEX-32 and SLIDEX-64 versions. Thus, SLIDEX-16 is the best performance-wise setting on this platform.

5.3.3 Energy, Power and Area Analysis

We have evaluated the cost of SLIDEX unit for the parameters used so far following the methodology introduced in Section 3.3. Table 5.3 summarizes the costs introduced to support it.

SLIDEX-16 requires $0.031mm^2$ increasing the base processor area by only 0.63%. Assuming $2.79mm^2$ area of an industry reference CPU [86], SLIDEX incurs a modest 1.13%

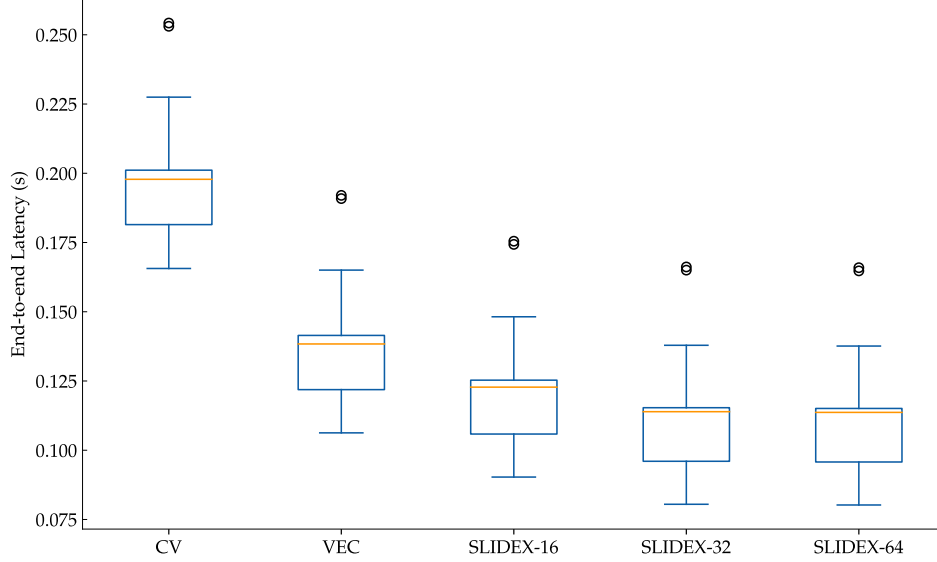


Figure 5.10: End-to-end latency of our localization system for the different systems tested in this work.

area overhead. We achieve a minimal impact on the area by designing it as a unit tailored for images with byte-level register banks and operations for this operator size.

Another vital aspect is power consumption. The increase in power is less than 1%, which helps maintain a low power despite the SLIDEX-16 unit’s high performance. Power dissipation is critical in autonomous driving systems [93] since it directly affects vehicle autonomy and cooling.

Table 5.3: Area and power for the baseline CPU and the different versions of SLIDEX.

Platform	Area (mm ²)	Static Power (W)	Dynamic Power (W)
Baseline CPU	4.87	0.572	0.934
SLIDEX-16	3.11E-02	2.73E-03	4.07E-03
Storage	3.02E-02	2.73E-03	3.98E-03
Exec. Unit	8.71E-04	2,66E-07	9.41E-05
Relative Change	0.63%	0.48%	0.43%

Finally, we analyze the energy consumption results of the SLIDEX units on the KITTI dataset. Figure 5.9c and Figure 5.9b show the relative energy improvement of the different versions of SLIDEX for the convolution and FAST tasks, respectively. The SLIDEX implementations obtain an energy reduction not only by reducing execution time but also by reducing the number of instructions and cache accesses. Figure 5.11 shows the average

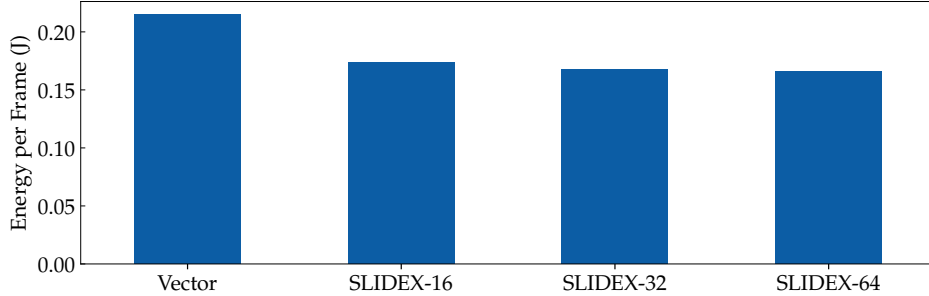


Figure 5.11: End-to-end average energy per frame for the vector and SLIDEX implementations.

end-to-end energy per frame for the vector and SLIDEX implementations. The 19% energy reduction comes from the execution time reduction, the number of instructions and their associated cost, and the reduction in the number of cache accesses. The energy reduction also stagnates with the increase of VL because SLIDEX-16 already saturates most of the benefits attainable by optimizing the Gaussian convolution and the FAST feature extraction in ORB-SLAM on this platform.

Furthermore, the memory access bottleneck and related cost dominate the task under study, causing the increase in VL to generate diminishing benefits. Thus, SLIDEX-16 is the most balanced configuration on this platform energy-wise. The vector implementation yields a PPA (Performance Per Area) of $1.44 \text{ FPS}/\text{mm}^2$ and a PPW (Performance Per Watt) of $4.64 \text{ FPS}/\text{W}$. However, SLIDEX-16 outperforms it with a PPA of $1.79 \text{ FPS}/\text{mm}^2$ and a PPW of $5.74 \text{ FPS}/\text{W}$, showcasing superior performance and energy efficiency.

5.4 Conclusions

In this chapter, we have introduced SLIDEX, a novel approach that supports the SWP execution model in conventional cores in the form of ISA extensions. SWP exploits the inherent data access patterns of the sliding window scheme, a standard traversal method to perform neighborhood operations. SLIDEX exploits the neighborhood window overlap, which provides many benefits compared to traditional scalar and vector execution models, including instructions with richer semantics, a decrease in data reads and alignments, and fewer intermediate result writes employing a novel scalable and flexible microarchitecture. We have demonstrated the feasibility of SLIDEX architecture by proposing special instructions to support 1D SWP targeting visual localization. SLIDEX achieves higher perfor-

mance and energy efficiency than vector implementations and only requires minor changes to support it, making SLIDEX a strong candidate to extend the functionalities of vector architectures in future vision platforms.

6

δ LTA

This chapter introduces δ LTA (δ ont't Look Twice, it's Alright), a novel mechanism that capitalizes on the inherent spatio-temporal similarity in vision streams by empowering the camera to discard redundant image regions before they enter subsequent CV (Continuous Vision) pipeline stages.

6.1 Introduction

Camera movements in CV typically result in smooth transitions, causing consecutive images to exhibit subtle position, angle, and orientation changes. These subtle variations create significant spatio-temporal similarities between frames, which provide an opportunity to optimize processing. The processing efficiency is enhanced because similar image regions tend to yield similar analytical outcomes. To illustrate, Figure 6.1a presents a similarity study that divides each image into a grid of varying-size regions and assesses the percentage of identical regions between consecutive frames. As region sizes increase, the percentage of identical regions decreases significantly. For instance, around 60% of 1x1 regions (i.e., pixels)

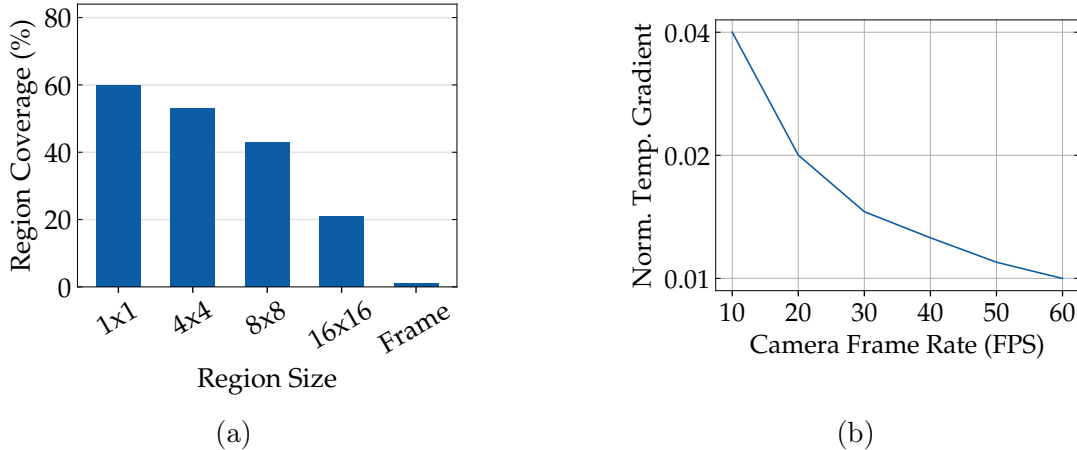


Figure 6.1: (6.1a) Similarity study at different region sizes across adjacent frames for sequence 00 of the KITTI [52] dataset. (6.1b) Reduction in Normalized Temporal Gradient with increasing frame rate (FPS).

are identical, whereas 20% of 16x16 regions share this characteristic. For larger regions, the percentage drops considerably, reaching nearly 0% of whole identical frames.

However, opportunities for computation reuse are much higher if we consider similar regions rather than totally identical ones. Figure 6.2 provides an example using 64x64 pixel regions, demonstrating that many regions exhibit minimal changes during typical camera movements. Several factors contribute to this observation, such as the parallax effect, camera sampling-to-velocity ratio, and low 2D frequency component areas.

The parallax effect causes objects far from the camera to appear to move slowly relative to the camera. Figure 6.2 depicts these areas in blue.

The camera sampling-to-velocity ratio also impacts the frame-to-frame similarity. Figure 6.1b illustrates the strong relationship between an increase in the FPS (Frames Per Second) rate and similarity within a given scene. We assess similarity using a metric called TG (Temporal Gradient), a widely accepted indicator for temporal similarity in video processing [164]. It involves computing the overall absolute difference between the pixel values of two consecutive frames by subtracting one frame from the other and then averaging the resulting pixel values across the entire image. A low TG indicates little change between consecutive frames, implying high similarity. The normalized TG (by the maximum brightness) consistently decreases as the frame rate increases, indicating that higher frame rates lead to higher similarity.

Another property of a CV stream is related to the low 2D frequency areas. These



Figure 6.2: The figure compares two consecutive KITTI frames and highlights regions that exhibit spatio-temporal similarity. Yellow regions represent areas with low 2D frequency and high similarity. The blue regions correspond to distant areas in the scene that move slowly due to parallax.

areas, containing little information, require lower frame rates than regions with more details. Figure 6.2 highlights similar regions in yellow. Despite movement between frames, many of these regions maintain nearly identical values due to the uniformity of their surrounding area.

6.2 Decoupling Camera Sampling from Processing

Traditionally, optimizations in CV have been approached by enhancing individual IPs in isolation. However, as outlined in Section 2.1, the CV pipeline comprises components that primarily operate independently, presenting a significant opportunity for integrating information across the system. This chapter introduces a novel contribution that leverages collaborative processing across the CV pipeline.

δ LTA redefines the CV pipeline by decoupling the frontend camera frame sampling from backend processing. It introduces an innovative CV frontend that utilizes frame-to-frame similarity to discard image regions in the very early stages of the vision pipeline during the frontend camera sampling. Instead of writing the entire frame to the FB (Framebuffer)

every time the camera captures an image, δ LTA frontend selectively updates only the regions of the frame that differ from the previous version and informs the backend which regions are different. This mechanism allows the backend to focus solely on processing the unique and distinctive regions of the image at their rate of change and not always at the arbitrary camera sampling rate.

Compared with current systems, δ LTA reduces the frontend FB updates, the redundant backend computations, and the backend FB accesses needed to produce equivalent final results with decreased end-to-end latency and overall energy consumption.

Another pivotal benefit of decoupling the backend workload from the sampling arises when considering camera frame rate scaling. Frame-to-frame similarity increases as the frame rate grows since the camera has less time to capture variations in the scene between consecutive frames up to a certain point when, for a given scene, more frame rate provides only redundant information. Hence, δ LTA can progressively filter the increasingly similar regions of these frames to limit the backend workload to the actual scene information. Moreover, increasing the frame rate enhances responsiveness. To illustrate this, consider a baseline system limited to processing streams at 20 FPS. Consider also that, assuming a given scene’s particular temporal similarity characteristic, this system with the δ LTA frontend can process 30 FPS instead. While the baseline system processes all image regions at 20 FPS, δ LTA-equipped one can selectively process some regions, the most distinctive that changed, at up to 30 FPS, thanks to freeing computational resources that otherwise would be devoted to processing irrelevant regions.

To ensure all these advantages, δ LTA must efficiently address two notable challenges. The first challenge is enabling the frontend to detect similar regions efficiently without significant latency and energy consumption overheads. To achieve this, δ LTA divides each frame into a grid of equally sized regions and computes a BRIEF (Binary Robust Independent Elementary Features) [22] signature for each region through a specialized hardware accelerator embedded in the ISP (Image Signal Processor) of the frontend. BRIEF signatures require a limited number of pixel computations. They can be efficiently stored in a small buffer in the ISP, as they only occupy a fraction of the original region. To identify similar regions in an incoming frame, the ISP computes the Hamming distance between the BRIEF signatures for each region of the incoming frame and the past frame stored in a buffer. This approach is highly efficient and requires a relatively small accelerator.

The second challenge of δ LTA is to design a versatile frontend interface since CV sys-

tems operate in various potential scenarios. We achieve this flexibility through a hardware-software co-design and collaboration approach. The proposed frontend interface offers the programmer two primary mechanisms to regulate its adaptable behavior: i) a threshold that determines when δ LTA ISP should consider that two regions are similar and ii) a mask that specifies which regions it must update regardless of the similarity measurement. The vision processing software can be modified to take advantage of the new frontend interface using the threshold and mask according to the characteristics and requirements of the CV system, its deployment, and the acceptable approximation levels.

6.3 Implementation

This section presents δ LTA, our novel solution to overcome the limitations of existing CV systems. We discuss the proposal’s core concept and implementation in modern vision frontends, specifically in the ISP, and describe how to harness its potential using software on the backend CPU. Importantly, any backend, such as a GPU or ASIC, can similarly leverage δ LTA.

6.3.1 General Overview

δ LTA optimizes the end-to-end image transmission and processing of the CV pipeline. It decouples camera sampling from processing to avoid unnecessary image data transmissions and redundant computations in the backend. It also enables proactive frontend and backend collaboration.

As explained earlier, modern CV backends have to access and process all the frames the camera generates since they cannot make any assumptions about them. Consequently, the backend’s workload depends on the camera’s frame rate. δ LTA breaks this coupling, enabling the ISP to discard similar regions in consecutive frames by exploiting the spatio-temporal similarity. Similar regions are not transferred to the main memory nor later accessed by the backend. The backend of a system equipped with δ LTA only has to process the distinctive regions of images that change. For the rest, it can reuse cached results or approximate the output using computationally cheaper methods. δ LTA design transforms a system that processes images based on the camera frame rate to one that processes the images based on how they change.

Moreover, current vision pipelines communicate only in one direction. The frontend processes the images without knowing anything about the requirements of the backend application. Consequently, it fails to adapt to them and always operates the same. For this reason, δ LTA enables proactive communication between both parts of the pipeline, employing a richer interface that informs what image regions changed compared to the previously captured images each time the camera samples. Additionally, it allows the backend to adjust its behavior. The backend can communicate with the frontend to indicate a threshold determining if two regions are similar and a mask to indicate which regions must be sent from the ISP to the Framebuffer (FB) regardless of their similarity. This way, the backend can adjust the performance, accuracy, and energy tradeoffs employing specific runtime information of the vision application.

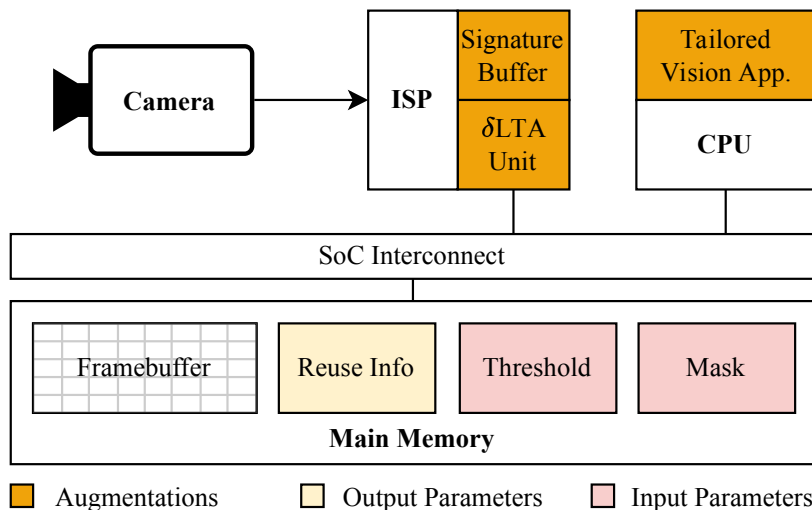


Figure 6.3: General overview of the proposed vision pipeline.

In summary, we enhance the state-of-the-art vision SoCs with a new ISP capable of filtering similar regions and a new interface that allows the backend to exploit this similarity. Figure 6.3 depicts the new extensions and frontend interface parameters. In the remainder of this section, we delve into the details of implementing δ LTA and demonstrate its effectiveness in addressing the challenges of efficient and flexible CV processing.

6.3.2 Frontend

The δ LTA frontend operates by dividing each frame into a grid of equally sized regions, extracting BRIEF [22] descriptors (introduced in Subsection 2.2.6) that are used as

signatures to identify these regions, and comparing incoming frame regions' signatures with the signatures of the corresponding regions in the FB. If the ISP deems a region similar, the ISP discards it, avoiding its transfer to the FB. Conversely, the ISP transfers the distinct regions to the FB.

Furthermore, we extend the frontend interface to enable proactive collaboration. The new ISP exposes the identifiers of regions that have changed since the previous FB version to the backend in a dedicated *Reuse Information* area within the FB (see Figure 6.3). The backend can also guide the ISP's actions by sending parameters via the FB interface, setting the threshold for region similarity detection, and supplying a mask to prevent the ISP from discarding a region, irrespective of the similarity mechanism. This new metadata is vital for facilitating ISP collaboration and allowing the backend to control the ISP's behavior, adjusting latency, energy, and accuracy tradeoffs on demand according to the requirements of the application, and even enabling the ISP to function as the baseline when required.

We integrated a highly efficient new hardware component into the ISP to support this new functionality that operates in parallel with other ISP stages to hide its latency. This design avoids the BRIEF extraction and matching to increase frame processing latency, even in worst-case scenarios where all regions differ or when the user deactivates δ LTA functionality.

The following subsection presents an example of the δ LTA frontend operation. Next, we will discuss the benefits of using BRIEF descriptors, followed by a description of the microarchitecture of the new component of the δ LTA ISP frontend and its integration within the pipeline.

Operation Overview

Figure 6.4 provides a high-level overview of δ LTA frontend operation. First, the backend initializes the system with a fixed threshold and an empty mask, indicating to the frontend that it has to send all regions since this is the first frame. The frontend divides the frames into an equally-sized grid. In our example, the ISP splits the frames into a 3×4 grid, with each *ISP* region having an associated identifier IR_i . Additionally, the color of each region indicates the frame to which it belongs.

The first frame processed in our example is Frame 0. Since it is the first frame, all regions are new, and the ISP must send all of them and their identifiers (Reuse Information)

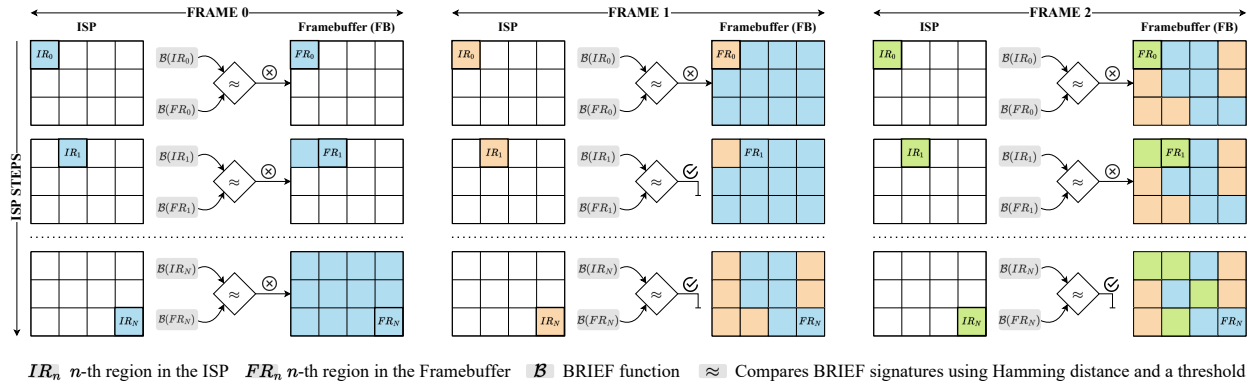


Figure 6.4: Overview of δ LTA technique for discarding redundant regions in the frontend.

to the FB in the main memory. We refer to the *Framebuffer Regions* as FR_i . The ISP computes the BRIEF (represented with \mathcal{B} signatures for each IR_i region and stores them in a Signature Buffer (SB). At the end of this process, the FB contains Frame 0 regions (in blue), and the SB stores signatures of every FR_i .

When Frame 1 arrives, the ISP extracts BRIEF signatures for every region and compares them with the corresponding signature in the SB using their Hamming distance. If the Hamming distance of a region is below the threshold indicated by the backend, the ISP classifies it as similar and does not transfer it to the FB (e.g., region IR_1 and IR_N). Otherwise, it sends the region to the FB, updates its BRIEF signature in the SB, and sends the region identifier to the Reuse Information of the frontend interface (e.g., region IR_0). This way, the backend knows which regions have changed. At the end of this process, the FB contains regions from Frame 0 (in blue) and Frame 1 (in orange).

When Frame 2 arrives, the same process occurs. The δ LTA ISP processes each region, extracting its BRIEF signature, $\mathcal{B}(IR_i)$, and compares it with the one in the SB corresponding to the same region, $\mathcal{B}(FR_i)$. It transfers the distinct regions to the FB and their identifiers to the Reuse Information. Upon finishing Frame 2 processing, the ISP sends only four regions, and the FB contains regions from all three processed frames.

Why BRIEF Descriptors?

We employ BRIEF [22] as a signature for the grid regions in which δ LTA divides each frame to leverage spatio-temporal frame-to-frame similarity on CV streams. Our decision to utilize BRIEF signatures over other methods [99, 16, 134] in δ LTA is motivated by several

key factors:

- **Robustness for δ LTA use-case:** BRIEF signatures can effectively measure similarity between regions under moderate changes like those found in consecutive spatio-temporal similar frames. We do not require scale or rotation invariant features that would raise the hardware complexity and cost.
- **Speed:** BRIEF generation and matching are highly hardware-friendly and computationally efficient, as they only require a limited number of pixel comparisons. The matching consists of computing the Hamming distance between two signatures. Other methods (e.g., SIFT [99]) require costly floating point operations. This efficiency is particularly beneficial in real-time scenarios where speed and energy efficiency are critical.
- **Compact representation:** Storing BRIEF signatures has minimal memory overhead, less than 4% of the image size. This small overhead eliminates the need for the ISP to double buffer past frame contents.

We evaluated the effectiveness of BRIEF signatures to describe the grid regions of frames in CV by conducting experiments to establish the correlation between our method and the Temporal Gradient (TG).

Figure 6.5a and Figure 6.5b display the negative correlation between the Normalized TG and the percentage of similar regions detected with BRIEF for 10 and 30 FPS, respectively. The correlation for the 10 FPS case is -0.785, indicating a strong negative relationship between the change between frames (represented by the normalized TG) and the number of similar regions identified by the BRIEF method. A similar correlation (-0.8) exists for the 30 FPS case. This robust negative correlation demonstrates that δ LTA BRIEF-based approach can effectively identify similar regions between consecutive frames. To further support this, we present in Section 6.4 results showing that our method achieves excellent accuracy when applied to a contemporary localization system.

6.3.3 ISP Implementation

In this subsection, we describe the microarchitecture implementation of the new δ LTA ISP. As shown in Figure 6.3, the new hardware includes a δ LTA unit that computes and

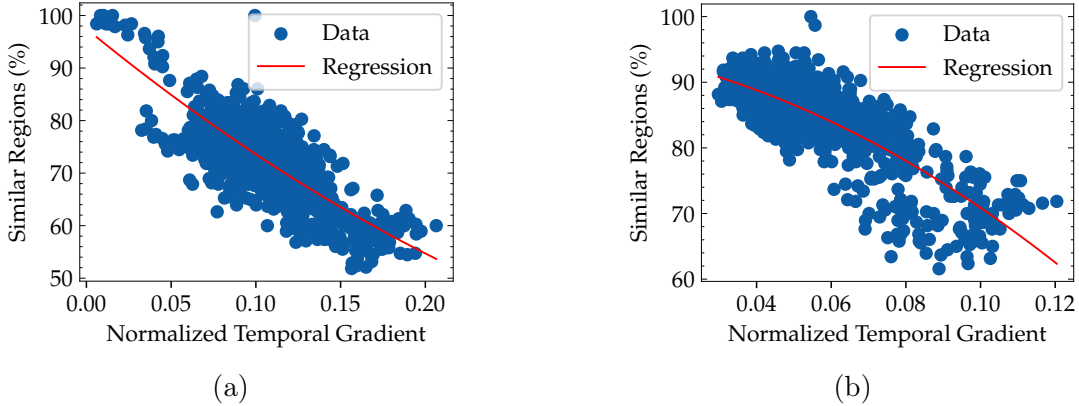


Figure 6.5: Correlation between the normalized Temporal Gradient and the percentage of similar regions obtained with δ LTA similarity detection method at 10 FPS (6.5a) and 30 FPS (6.5b) for the sequence 00 of KITTI [52] benchmark.

matches BRIEF signatures of image regions and a Signature Buffer (SB) that stores the BRIEF descriptor of every image region stored in the FB ($\mathcal{B}(FR_n)$). Every time the ISP transfers a region, it updates the corresponding signature for the region in the SB to maintain coherence between the SB and the FB contents. Our choice of using BRIEF signatures makes the SB cost relatively low. For example, for an FHD (1920x1080 image) that occupies approximately 2MB, the SB requires only \sim 64KB.

δ LTA Unit

Figure 6.6 illustrates the δ LTA unit microarchitecture, designed for efficient and hardware-friendly filtering of similar image regions. It features a streaming architecture capable of processing one pixel per cycle, utilizing a 2D line buffer to access image regions (IR_n) of 32x32-pixel size. State-of-the-art image processing architectures use streaming line buffers because of their efficiency [154, 64, 43, 160].

The δ LTA unit applies a predefined pattern of pixel intensity comparisons to each specific region to generate its BRIEF signature. Each comparator (CMP), shown in Figure 6.6, takes Gaussian-filtered pixels from the region as input to generate one bit of the final signature. The *Gauss* modules efficiently perform the filtering using a 3x3 Gaussian kernel approximation, $\frac{1}{16} \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$, that only requires sums and proper bit-wise wire routing. Filtering of the region data smooths out high-frequency noise and details, improving the robustness of BRIEF.

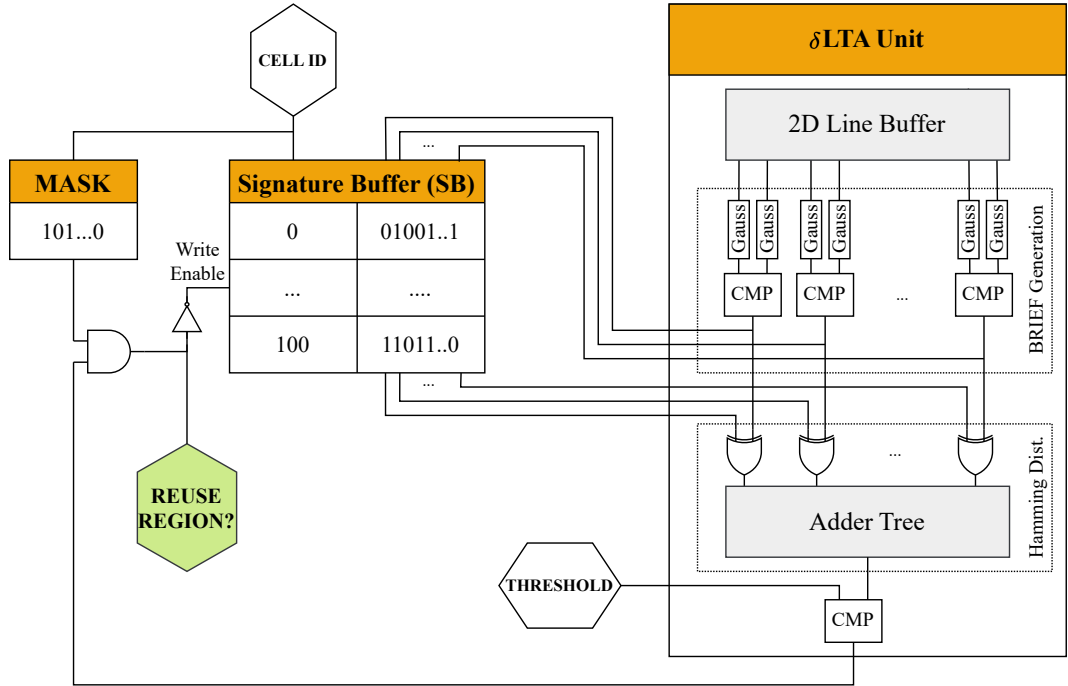


Figure 6.6: Microarchitecture of the new ISP components: δ LTA unit and Signature Buffer.

The unit subsequently compares the newly generated BRIEF signature ($\mathcal{B}(IR_n)$) with the stored SB signature from the FB’s corresponding region ($\mathcal{B}(FR_n)$), calculating the Hamming distance using the sum of bit-wise XOR operations between the two descriptors. Then, it compares the distance with the threshold input parameter for the similarity test and updates the SB if the region is not similar. The mask input parameter determines if the unit updates the region’s SB entry, irrespective of the similarity test.

ISP Pipeline Integration

We consider a baseline ISP that resembles literature [64, 20] and commercially available solutions [106, 9, 8]. An ISP sequencer orchestrates various ISP stages to convert the camera sensor’s raw data to RGB and YUV color spaces (demosaicing) and to apply several image enhancements afterward. As explained in Section 2.1.2, some of these improvements adjust the image’s color balance to ensure accurate color reproduction (color correction), adjust the gamma and apply HHDR (High Dynamic Range) processing to improve contrast and detail in both bright and dark areas (tone mapping) or improve image quality (sharpening), among others [64, 20, 8]. The generated image is temporarily stored in the DMA (Direct Memory Access) buffer to smooth data flow within the ISP pipeline before the DMA

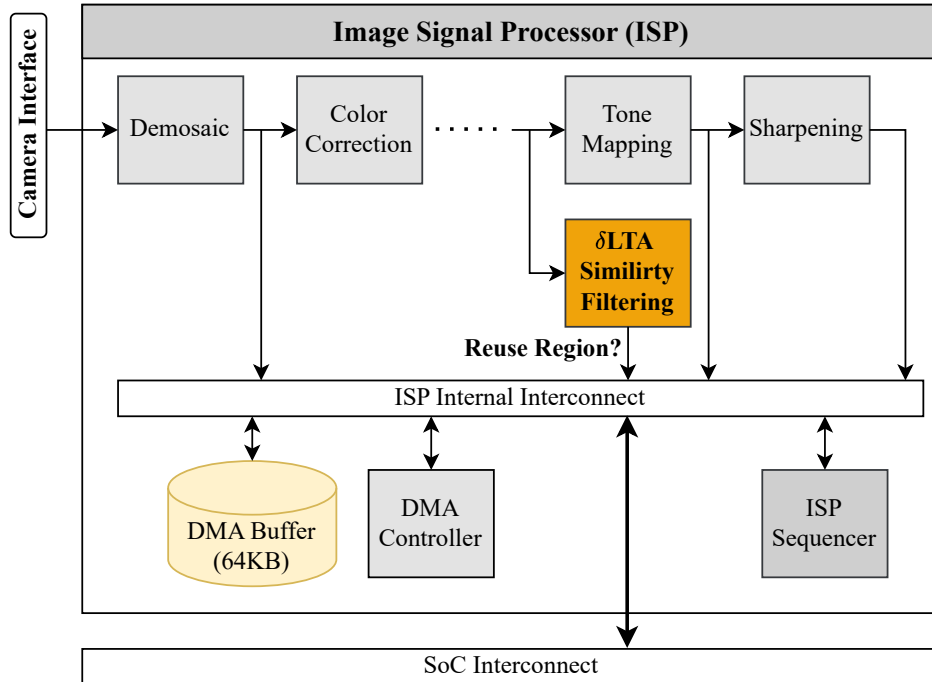


Figure 6.7: δ LTA incorporates a new ISP stage for BRIEF Similarity Filtering, which overlaps with other ISP stages.

controller transfers it to the main memory.

Figure 6.7 illustrates the described traditional ISP pipeline and the integration of the δ LTA unit into it by adding a new δ LTA similarity Filtering stage. This stage generates a reuse signal that the DMA controller uses to avoid sending those regions deemed similar by the δ LTA unit.

This new stage runs in parallel with tone mapping and sharpening stages since it does not rely on the images produced after these stages. As described in the previous subsection, the δ LTA unit utilizes a Gaussian filter to blur the regions, effectively negating the image-enhancing advantages of tone mapping and sharpening stages.

We modeled a cycle-accurate RTL hardware description of the δ LTA similarity Filtering stage and its integration with the two final ISP pipeline stages to demonstrate that the new δ LTA stage latency overlaps with a standard ISP pipeline. We measured that our new stage has a latency of up to 3ms when running at 700 MHz. We confirmed that its latency could overlap even with two simple 5x5 stencil operations. Since the latency of ISP stages in real scenarios is more significant than in our experiments, we can conclude that the δ LTA Similarity Filtering latency can be hidden entirely. This zero-latency overhead

design is particularly advantageous for mission-critical real-time visual applications, where minimizing latency is crucial.

Overall, δ LTA frontend allows for highly efficient and hardware-friendly similarity detection, enabling the system to focus only on processing the unique and distinctive regions of the image that change while avoiding redundant computations and unnecessary data transfers. Next, we show an example of how a visual application running in a CPU backend can benefit from using δ LTA.

6.3.4 Backend

δ LTA frontend is generic and flexible since it makes no assumptions about the application. It divides the image into regions and classifies them based on visual change. On the other hand, the backend can exploit more specialized insights from the application and adapt the requirements at runtime, proactively communicating with the frontend through the versatility of our proposed FB interface.

A vision system can exploit δ LTA implementing a background service similar to the ones proposed in prior works [91, 165, 59], but in a more efficient manner. As explained in Subsection 1.3.5, these previous works leverage frame-to-frame similarity by using image contents as a key for a SC (Software Cache) that temporarily memoizes the associated output results for the regions (e.g., CNN partial convolution results). These methods must access the FB and process the whole image to generate a key (e.g., a descriptor or feature as an identifier) and perform cache lookups for every incoming image. This approach induces an unavoidable energy and latency overhead that reduces the gains for cache hits and increases the worst-case latency since they require reading and processing the entire image for cache lookup. This cost rises with the camera resolution and frame rate increases; hence, the CPU and memory transmission bandwidth limit the benefits of these methods. By leveraging the δ LTA frontend interface, it is possible to mitigate these limitations significantly.

We illustrate the benefits of δ LTA through a crucial emerging application, a state-of-the-art visual localization technique called ORB-SLAM [23, 113] that typically runs on CPUs [50]. We aim to leverage δ LTA to reduce latency and energy consumption without impacting localization accuracy.

To achieve this, we modified the ORB-SLAM codebase to include a SC to store the ORB features and all its associated metadata (e.g., BRIEF descriptors) that the algorithm

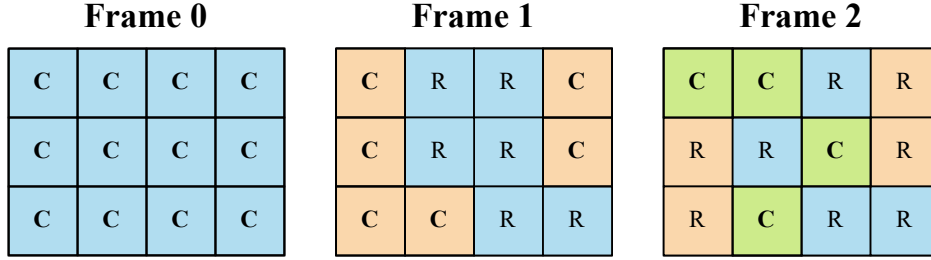


Figure 6.8: Illustration of how the backend CPU reuses the computations for the redundant regions (R) and only processes the non-similar regions (C).

uses as the basis for its entire execution. ORB Feature extraction represents more than 60% of execution time for SLAM and up to 90% for localization [50] as we also reported in Figure 1.2. By leveraging δ LTA, the modified algorithm only processes the regions that change across consecutive frames while ignoring the remaining ones, as detailed in the following subsections. In Section 6.4, we demonstrate how the resulting modified version of ORB-SLAM significantly reduces computation and memory requirements while exhibiting the same accuracy.

Operation Overview

The backend initiates the camera pipeline by setting a threshold and an empty blank mask to compute all regions for the first frame. Figure 6.8 shows the final FB contents seen by the CPU after the δ LTA frontend operation, as discussed in Subsection 6.3.2.

When the δ LTA frontend finishes processing Frame 0, the backend starts processing it. First, the backend reads the Reuse information, which for Frame 0 indicates that all the regions changed from the last frame (represented with a C in the regions of the figure) since it is the first frame. Based on this information, the backend processes each region, writing the extracted ORB features in a sSC that stores these results for each FR_n .

Upon Frame 1 arrival, the backend repeats the same process. It reads the Reuse Information that informs which FR_n regions changed from the past frame. For these regions, the localization algorithm extracts ORB features as usual, and like in the previous frame, it stores the results in the SC. The system can reuse the features extracted from the last frame for the rest of the regions, labeled with an R in the figure. The backend does not have access to the pixel values for similar regions, only to the Reuse information. The rest of the algorithm continues the processing, employing the ORB features stored in the cache

to localize the camera.

Next, the backend processes Frame 2. It reads the Reuse information and recomputes only the regions that changed from the past FB version. For similar regions, the SC contains the values from either Frame 0 or Frame 1. Similar to what happens with the frontend, where the ISP SB is kept coherent to hold the BRIEF signatures of the regions of the FB, the backend also stores the results (e.g., ORB features) of each region of the FB.

Threshold

A crucial element of our proposal is its adaptability to various application requirements. The threshold parameter, a value between 0 and 1, dictates the intensity with which the frontend discards regions. This threshold can be set on a per-frame basis following different strategies, for example:

- **Static Thresholding:** This approach involves using a fixed threshold throughout the entire execution of the application without any alterations.
- **Utilizing Application-specific Insights:** Many characteristics unique to specific applications can offer valuable information to modify the threshold. For instance, in our case study, the number of matched ORB features or unsuccessful re-localization attempts could be employed to fine-tune the threshold to be more aggressive in regions that have a minor impact on the localization outcome.
- **Multisensor Augmentation:** Additional sensor data from the external environment can be harnessed to adjust the threshold. For example, in some AD systems, the Inertial Measurement Unit (IMU) supplies crucial information on vehicle acceleration [124, 116, 147, 168, 171]. The system can leverage this information to dynamically modify the threshold, lowering it when the vehicle accelerates, turns, or executes unusual maneuvers where heightened accuracy is essential.

For dynamic approaches like the last two above, the system requires a threshold control mechanism receiving feedback from camera pose error estimations. The backend can estimate this error every N frames by conducting two parallel independent pose estimations using the full and δ LTA-filtered frames and comparing the corresponding poses. Our proposal supports this process by masking all the regions and forcing the frontend to send them. Note that the δ LTA frontend still updates the Reuse Information.

Our experiments use conservative static thresholding to illustrate a lower bound of the potential benefits.

Masking Regions

The other mechanism to control δ LTA behavior is a mask. The mask is a binary string where each bit, m_i , tells the ISP if the region IR_i must be sent to the FB regardless of the similarity with previous regions. If the application knows that a region will likely contain important information, it may conservatively decide to recompute it irrespective of the similarity test. Moreover, masking all the regions implies that the frontend operates under baseline conditions, transmitting all regions to the FB. Deactivating δ LTA functionality proves beneficial in conservative situations or where backward compatibility is necessary.

This masking mechanism can be leveraged by drawing insights from the particular visual application. For example, the bounding boxes of objects detected by CNN models exhibit spatial locality in consecutive frames. If the model detected a pedestrian in a frame, it would likely be in the same image region in the next frame. The backend can exploit this mechanism by masking the regions where objects were detected in the precedent frames to force the frontend to send these regions to the FB.

Likewise, in our case study, ORB features in a frame are expected to be in nearby positions in the next frame. Since these features determine the final outcome of the algorithm, the backend could decide to mask the regions that contain a large number of features to force their computation while using a more aggressive threshold for the remaining ones to achieve more significant savings without impacting accuracy.

6.4 Experimental Results

This section evaluates the effectiveness and efficiency of δ LTA, comparing it against the state-of-the-art SC mechanisms, detailed in Subsection 1.3.5 (i.e. Deepmon [69]). We use the ARM Cortex A72, described in Chapter 3, as our baseline for these comparisons. The assessment involves three versions of our localization system:

- **Baseline:** Original version of ORB-SLAM compiled for ARM.
- **Software Caching (SC):** A modified ORB-SLAM version supporting a SC mechanism that computes BRIEF signatures of image regions and reuses previously cached

ORB image features based on the similarity of the regions. This mechanism performs cache lookups on the backend CPU, and hence, it needs to read all the image regions from the main memory.

- **δ LTA**: A modified version of ORB-SLAM that incorporates the δ LTA camera interface. This modification enables the reuse of computations from previously processed similar regions, avoiding access to such regions and eliminating software cache lookups.

In our localization experiments, we utilize a static threshold established before execution. We have adapted ORB-SLAM to turn off the SC and δ LTA during system initialization, relocalization, or tracking loss. Furthermore, we conservatively mask regions that contained at least one feature in the preceding frame.

The terms SC-TH and δ LTA-TH denote the use of the SC and δ LTA schemes, respectively, with a specified threshold value (TH).

6.4.1 Region Coverage and Accuracy Analysis

Figure 6.9a presents the average percentage of regions our method deems similar while the localization application processes the KITTI dataset. The lower the threshold, the higher the percentage of similar regions. This increased region coverage not only enhances the system’s efficiency but also enables it to process more relevant data in real time.

On the other hand, it is crucial to maintain the accuracy of the application. We assessed the localization accuracy compared to the Baseline using the APE (Absolute Pose Error). APE is a key performance metric for localization algorithms, measuring the difference between the ground truth pose and the estimated pose at each time step. High localization accuracy is essential, particularly for AD systems that require centimeter-level precision. Consequently, we report the maximum APE as our central accuracy metric since a single incorrect critical estimation can significantly impact the user experience. Figure 6.9b presents all KITTI sequences’ maximum APE. We can see that our method introduces no error for threshold values up to 0.7. For 0.7, there are no errors except for sequence 02, which incurs a minor error of half a centimeter in the worst case. Higher thresholds begin to incur some non-negligible errors, so we conclude that 0.7 is the most appropriate threshold and is the one adopted by our experiments unless otherwise indicated.

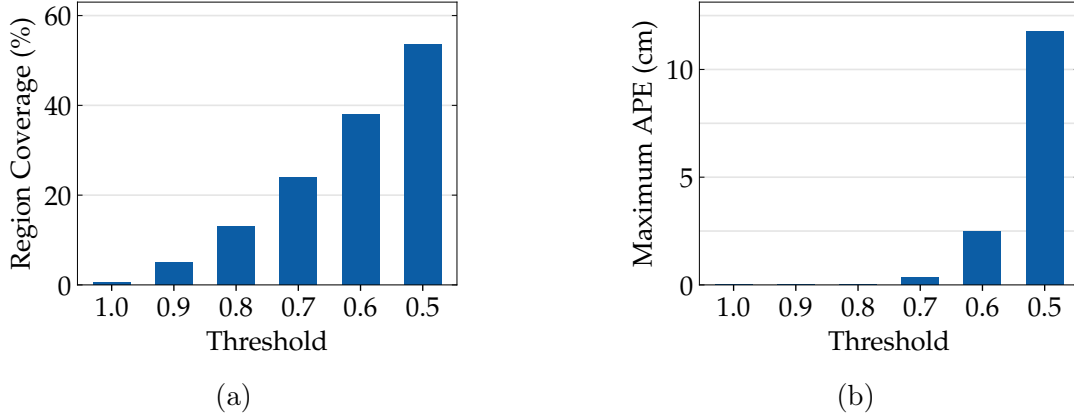


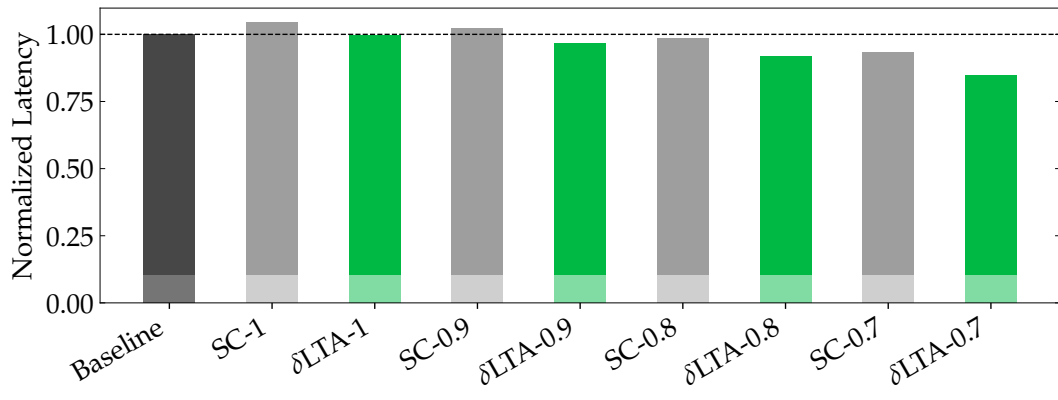
Figure 6.9: (6.9a) Percentage of similar regions. (6.9b) Maximum Absolute Pose Error (APE) for different thresholds.

6.4.2 Latency Analysis

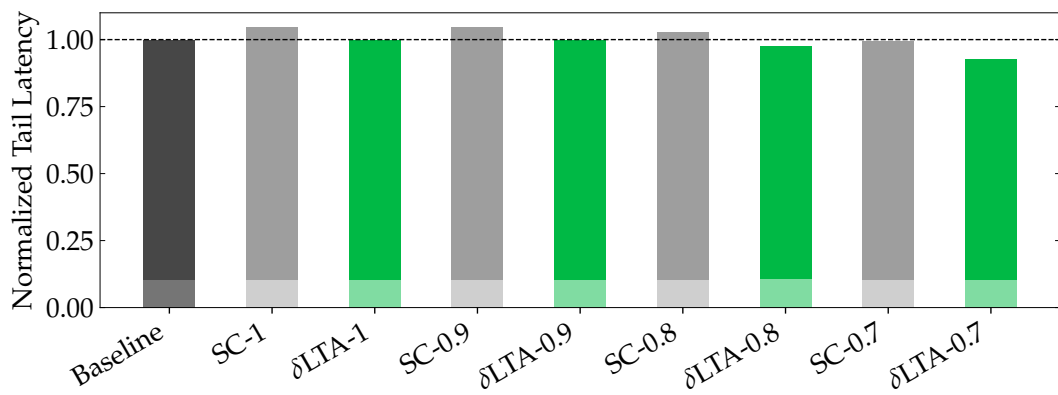
To evaluate the performance of our proposed solution, we examine the end-to-end latency and the 99th percentile tail latency of the CV pipeline, excluding the image sensor. Tail latency is paramount in the context of AD systems since they have extreme constraints regarding response time. Figure 6.10a illustrates the normalized average latency, comparing the Baseline, SC, and δ LTA. Figure 6.10b depicts the tail latency. To measure the overheads of the different approaches, we also evaluate both δ LTA and SC with a threshold of 1, representing a scenario where minimal region reuse occurs (approximately 0.53%, when the camera stops). In this case, the benefits offered by these schemes are negligible.

Our experiments led to several conclusions. SC’s benefits are minor in average latency and practically null in tail latency when not harming it (SC-1, 0.9, and 0.8). The overhead of cache maintenance and lookups offsets the reuse benefits, leading to a negative impact on the worst-case response of the system, which in turn can introduce safety concerns in mission-critical scenarios such as AD.

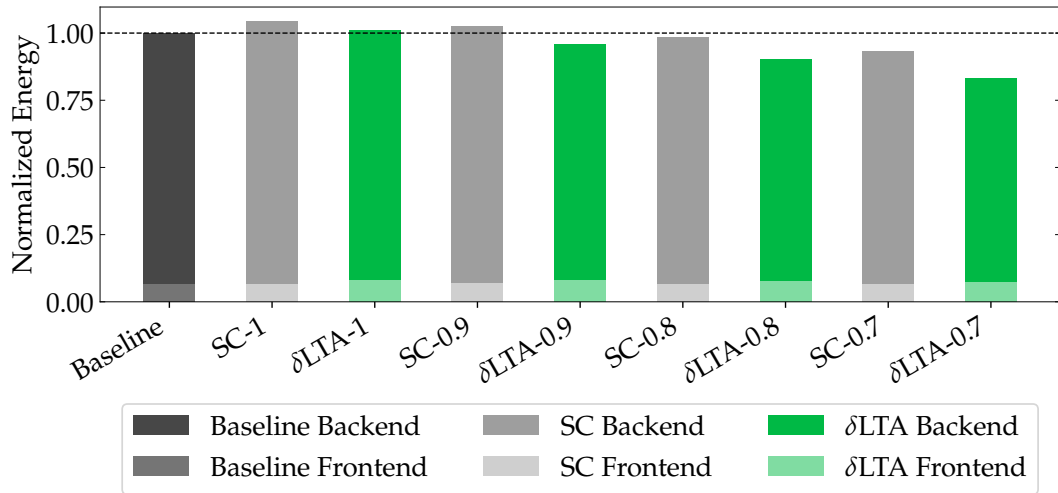
On the other hand, δ LTA introduces no overhead since other ISP tasks entirely hide its latency. Our proposal greatly outperforms SC mechanisms in both average and tail latency reduction. It achieves an average latency reduction of 15.22% and a tail latency reduction of 7.2% over the Baseline, as well as an average latency reduction of 9.15% and a tail latency reduction of 7% over SC.



(a)



(b)



(c)

Figure 6.10: Comparison of Normalized Latencies (6.10a), Normalized Tail Latencies (6.10b), and Normalized Energy (6.10c) for Baseline, Software Caching (SC), and δ LTA.

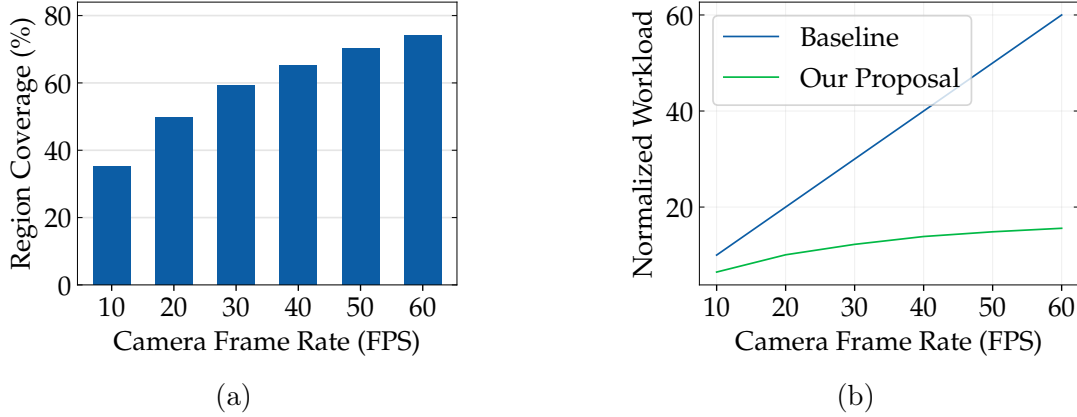


Figure 6.11: (6.11a) Percentage of region coverage for different camera rates. (6.11b) Amount of computations required when increasing frame rate.

6.4.3 Energy Analysis

Figure 6.10c compares energy consumption among the Baseline, SC, and δ LTA configurations. δ LTA achieves a 17% reduction in energy consumption compared to the Baseline and a 11% reduction compared to the SC scheme. These results emphasize the advantages of decoupling camera sampling from processing. The additional energy savings of δ LTA come from fewer ISP and CPU FB accesses and a more energy-efficient implementation of similarity detection.

6.4.4 Frame Rate Scaling Analysis

In this section, we examine a central advantage of δ LTA when processing higher camera frame rates, something to expect in future systems. Figure 6.11a illustrates how frame-to-frame similarity in a given scene increases as the frame rate ascends (without masking regions). Increasing the frame rate reduces the time between samples, leading to fewer changes and, thus, more similarity.

We study the backend workload response of our proposal when scaling up the frame rate and comparing it to the Baseline. We assume ideal conditions, meaning no ISP, main memory, or backend CPU bottlenecks, allowing the system to process arbitrary amounts of pixels. Figure 6.11b shows the results of this sensitivity experiment.

In the Baseline system without δ LTA, the backend must access all image pixels, resulting in an increased workload as the camera frame rate grows (see blue line in Figure 6.11b).

While a SC mechanism can assist the system in skipping some computations to exploit the similarity increase, it does not eliminate the need for processing the images. In more realistic scenarios, the inherent overhead of the SC would eventually become dominant, hindering the system’s ability to function and even resolve SC hits in real time. Thus, the Baseline backend workload depends on the camera frame rate, and it not only constrains the system’s real-time responsiveness but also misses potential opportunities for energy savings.

In contrast, the δ LTA-equipped system discards redundant image regions based on similarity, which for a given scene becomes more frequent as the frame rate increases. Beyond a certain point, more samples do not provide more relevant information about the scene; therefore, the number of distinct regions the backend needs to process eventually stagnates (see green line in Figure 6.11b). As demonstrated, our proposal enables CV SoC to break the backend workload dependency from camera sampling, allowing it to adjust the processing rate based on the amount of information in the captured scene.

6.4.5 Area and Power Analysis

We assess the area and power costs of δ LTA using the approach described in Section 3.4, considering that the δ LTA unit supports up to Full HD resolution (standard nowadays).

δ LTA introduces a minor area overhead of $0.122mm^2$ due to the ISP augmentations. The main area overheads come from the $\sim 64KB$ for the Signature Buffer to store BRIEF descriptors and the $\sim 60KB$ for the 2D line buffer. This area overhead is negligible compared to a typical SoC area, such as the $100mm^2$ of a Mediatek A72 cluster [105], the $88mm^2$ of Apple A14 [2], or the $350mm^2$ of Nvidia Xavier[117]. δ LTA dissipates 50mW mainly due to the 2D Line Buffer that needs to read and write several pixels during ISP processing.

6.5 Conclusions

In this chapter, we have presented δ LTA (**δ ont’t Look Twice, it’s Alright**), a novel architecture that decouples the camera frame rate from the backend processing rate by augmenting the camera with the ability to identify redundant image regions and notify the backend of the specific image areas that exhibit substantial changes compared to the preceding ones. This new frontend capability allows the application to reuse previously computed results for the redundant regions.

Our evaluation shows that δ LTA can significantly reduce the data sent downstream while maintaining high accuracy. Our approach reduces the application latency by 15.22% and its energy consumption by 17%, significantly improving the performance of CV pipelines even in worst-case scenarios. We also show that δ LTA enables systems to process at higher sampling frame rates by focusing solely on processing the valuable visual information, which enhances general responsiveness. Consequently, solutions like δ LTA are notably promising for emerging critical applications such as AD (Autonomous Driving) or XR (Extended Reality). Finally, our work also exemplifies the potential for optimizations in modern vision pipelines, including the ISP, when carefully considering its holistic end-to-end operation perspective.

7

IRIS

This chapter presents IRIS (Image Region ISP-Software Prioritization), our second contribution targetting inter-stage CV pipeline collaboration. IRIS leverages byproducts already generated and currently discarded in the ISP (Image Signal Processor) to forge a metric useful to detect relevant image regions, optimizing available computation resources accordingly.

7.1 Unleashing ISP-Software Cooperation

We make three observations regarding modern CV SoCs:

1. The ISP is a highly efficient and specialized component that performs several fundamental image processing operations whose intermediate results it routinely discards after processing each frame coming from the camera.
2. The backend operates without prior knowledge of the image, and it often requires processing the whole image in a predetermined order (e.g., raster-scan) to either pre-

process the image to detect the most distinctive image regions or directly post-process it to generate the application’s high-level semantic outcome. However, not all regions in the captured images contain relevant information.

3. The frontend and backend operations are isolated since these two SoC (System-on-a-Chip) components have no synergistic collaborations between their image manipulation tasks.

We propose to unleash ISP-software collaboration through the use of our novel proposal IRIS. IRIS enables the frontend to expose a priority map of the image regions based on their prominence, using already computed operations along the ISP pipeline. The backend can adopt an iterative or incremental image processing approach where the first iterations of the algorithms begin processing the most salient regions, yielding relevant insights before the baseline system. IRIS approach potentially reduces latency and significantly cuts energy consumption by sidestepping the non-essential regions (e.g., the sky or the road in AV).

Given that our optimization focus is non-functional—targeting latency and energy reductions with minimal application accuracy impact— it is imperative for our IRIS SoC augmentation design to avoid critical latency overheads, which could dramatically impact real-time applications. We constrain our design to only leveraging existing imaging computations of modern ISPs and solely augmenting them with lightweight extensions to account for each region’s metric.

Modern ISP pipelines implement multiple image enhancement algorithms. We focus on two of them, namely, edge enhancement and motion estimation. Edge enhancement improves the perceived sharpness of the final image by accentuating the acutance around the image edges, which implies that the ISP implements edge detection capabilities. Motion estimation determines how pixels move between consecutive frames by associating a motion vector between each pixel or block of pixels with its past position. High-end ISPs use these motion vectors at various stages, such as TD (Temporal Denoising) [80] or video stabilization [103, 125, 174].

Our ISP augmentation divides the image into equally sized regions, forming a grid. It computes an importance metric for each region to form a PPM (Priority Processing Map) that it exposes to the rest of the SoC as new metadata in the framebuffer. The PPM combines the information from the EDM (Edge Density Map) and the MM (Motion Map). IRIS implementation assigns a score to each cell based on its edge density to form the EDM. For

the MM, it calculates each region’s motion vector magnitude. The backend then consumes this information to adjust its processing order, ensuring prompt processing of the more salient critical regions. Our evaluations indicate that while software-only-based solutions introduce significant overheads, our IRIS design seamlessly integrates with the ISP, further amortizing its cost while guaranteeing minor ISP latency overhead. For a 1080p frame, the newPPM metadata occupies around 8KB, which is relatively small compared with the 6MB size of the frame.

IRIS provides a flexiblePPM calculation by combining the EDM and MM in a manner that the backend can control to suit the vision application’s needs best. Generally, image-processing applications benefit from prioritizing regions with higher fundamental edge features. However, different applications and scenarios profit differently from motion exploitation. For example, suppose the camera is static, such as in video surveillance. In that case, motion is a strong indicator of moving objects, and the CV applications can leverage MM to accelerate object detection and tracking. Other applications would benefit from ignoring moving areas and focusing only on stationary ones, such as localization and mapping [145, 172, 5]. If the camera is moving, common in AD or AR, the perceived relative motion accounts partly for the camera’s movement, and due to the parallax effect, closer scene areas move faster than farther ones. An agent configuring our novel ISP to prioritize regions with more motion would benefit from the early processing of close scene areas, which can benefit collision avoidance or localization.

7.2 Implementation

This section presents IRIS, our novel solution to unleash ISP-software cooperation and optimize existing CV systems. We discuss the proposal’s core concept and implementation in modern vision frontends (ISP) and describe how to harness its potential using co-designed software on the backend CPU.

7.2.1 General Overview

This work highlights a fundamental limitation in modern CV systems: the backend typically indiscriminately processes all image areas, lacking the context that other IP components of the vision pipeline, such as ISPs, might offer. This context could provide valuable information on which regions to prioritize computational resources and where to downscale

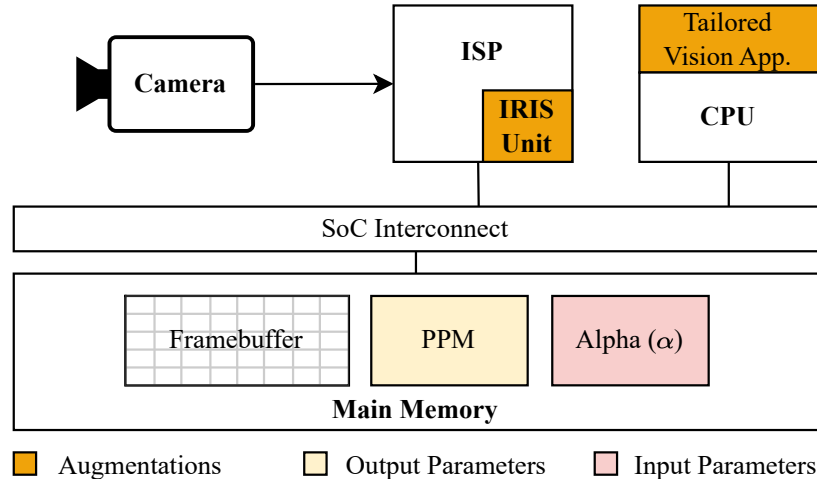


Figure 7.1: General overview of the proposed vision pipeline.

efforts, such as low-detail background zones or the sky. The backend could pre-process an entire image and produce equivalent or even more advanced contextual information to prioritize regions of the image. However, this process would come with a high cost in real-time applications, and such pre-processing costs may outweigh its potential benefits.

IRIS overcomes these challenges by repurposing specific on-the-flight ISP image analyses to compute and expose new image region relevance information. Our proposal allows the backend to reuse ISP internal computations through ISP-software co-design, harnessing application-specific insights to exploit these new SoC-level synergies. Furthermore, as future ISPs evolve and integrate more sophisticated features, such as AI capabilities, we expect IRIS’s collaborative approach to become vital for efficiency.

The backend of a system equipped with IRIS can adopt incremental or iterative image processing mechanisms cost-effectively by capitalizing on the new ISP-provided information since it does not require new computations to analyze the image. Critically, our hardware augmentations of the ISP do not increment its latency, incurring negligible end-to-end overheads even in the worst-case scenario.

Overall, the IRIS scheme enhances real-time CV processing efficiency, reducing ineffective backend framebuffer memory accesses and computations while amortizing further the cost of standard SoC components within the ISP. Figure 7.1 depicts the new extensions and frontend interface parameters we will explain in the subsequent sections.

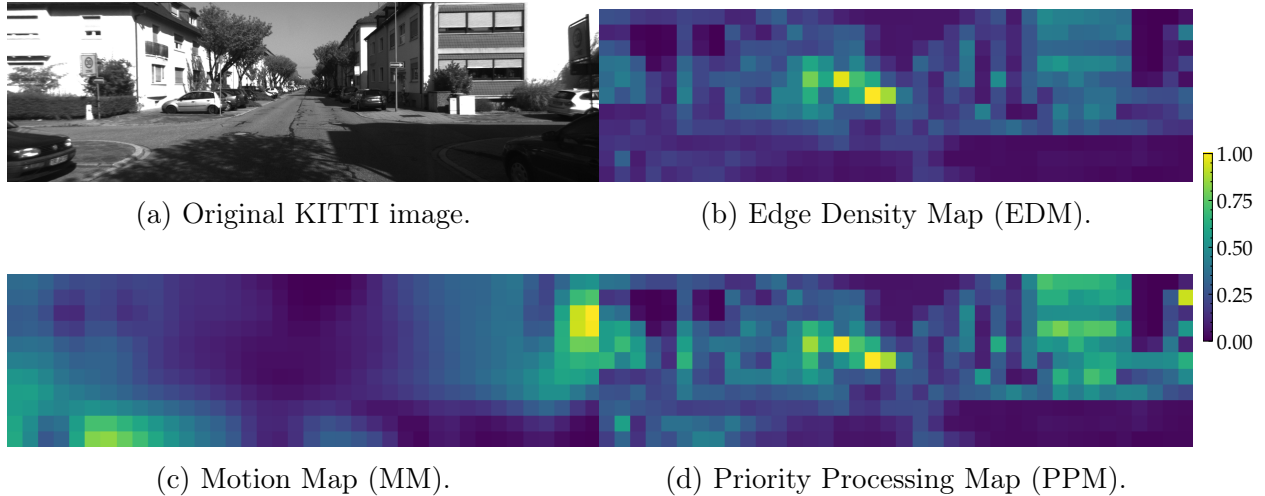


Figure 7.2: Visualization of IRIS prioritization maps. (7.2a) The foundational KITTI image. (7.2b) The EDM highlights areas with dense attributes such as building details and cars. (7.2c) The MM, illustrating forward camera motion, approximates scene depth. (7.2d) The PPM combines EDM and MM, prioritizing regions with features and closer to the camera (compared with the EDM).

7.2.2 Augmenting the Vision Frontend

We propose a lightweight ISP augmentation to empower the frontend with the ability to divide the incoming images into a grid of equally sized regions, collect early fundamental visual insights from internal ISP pipeline stages for each region of the grid to form the Priority Processing Map (PPM), and share this information to the rest of the SoC as new metadata in the framebuffer.

The PPM is a key feature of our design, effectively merging crucial image information from spatial and temporal domains. From the spatial domain, we use edge information to compute an Edge Density Map (EDM). Edges define shapes and objects, providing structure and making them critical for recognizing regions of interest in the images. From the temporal domain, we use motion information to calculate a Motion Map (MM). Motion captures change and dynamics in the environment. Combining these maps in the PPM enriches the saliency scoring mechanism to reflect each particular image region’s distinctiveness and importance.

Our design computes the EDM and MM by reusing the byproducts from the EE (Edge Enhancement) and ME (Motion Estimation) performed within the TD stages of the ISP, which we introduced in Subsection 2.1.2. IRIS also employs the existing ISP SoC-level

communication mechanisms to expose the PPM in the framebuffer. These decisions provide high flexibility for potential use cases of the PPM in the backend with minimal costs.

Figure 7.2 provides an example of a camera image featuring a car moving through a city (Figure 7.2b) alongside the maps generated by our approach, which we describe below.

Edge Density Map (EDM)

The EDM is a representation that assigns a score to each region in an image based on its edge density. Our technique calculates the edge density in the ISP as the number of edges per unit of area in a specific image region normalized by the maximum possible density.

Edges are fundamental, well-known image features. Discretizing the image into regions to assess its edge density allows us to inform the backend about the distinctiveness of image zones with a potential abundance of objects, details, and regions of interest in a practical and manageable form. Figure 7.2b depicts a visualization of the EDM of the image in Figure 7.2a. In this example, the EDM highlights the center regions as the most distinctive ones, corresponding with several cars, trees, and details of the surrounding buildings. On the other hand, the regions of the sky, some parts of the road, or textureless walls have the lowest scores.

Motion Map (MM)

The MM is a representation that assigns a score to each region in an image based on its perceived 2D motion. Our technique calculates the perceived motion of an image region by computing the magnitude of the associated Motion Vector (MV) generated in the ME stage of the ISP. We opportunistically design the MM grid aligned with the macroblock structure of the Block Matching algorithm (discussed in Subsection 2.1.2) so that there is a unique MV at the center of each MM region.

Measuring perceived 2D motion is crucial in identifying relevant image regions in dynamic scenes. In surveillance systems with a static camera, the perceived motion helps recognize the regions of interest that require careful processing. However, when the camera is moving, as in AD and AR, all the components of the scene, including static areas, appear to be in motion relative to the camera. This motion is more pronounced for objects closer to the camera than those further away due to the parallax effect, allowing the MM to approximate each image region's relative depth. Figure 7.2c provides a visualization of the MM of the

image in Figure 7.2a. In this example, the car with the onboard camera capturing the original image is moving forward. The MM captures a basic structure of the scene, with higher scores assigned to the areas closer to the car.

Priority Processing Map (PPM)

We formulate the PPM to integrate salient image characteristics from both the EDM and the MM in an image divided into an $N \times M$ grid of regions. The following equation describes how to calculate the PPM score for the (i, j) grid region:

$$PPM_{ij} = EDM_{ij} + \alpha \times EDM_{ij} \times MM_{ij} \quad (7.1)$$

Here, EDM_{ij} and MM_{ij} denote the edge density and motion scores for the region (i, j) , respectively, while α is a scaling factor determining the relative negative, null, or positive influence of the motion information. Our design encodes each PPM_{ij} in a single byte, resulting in a metadata size of only 8KB for a 1080p frame, which is small compared to the frame size of 6MB.

Figure 7.2d provides a visualization of the PPM of the image in Figure 7.2a using $\alpha = 1$. In this example, EDM accentuates areas with rich and distinct spatial information. In contrast, the MM emphasizes regions nearer to the camera, distributing high PPM scores across the image.

7.2.3 Architectural Support

In this subsection, we describe the architecture augmentations to support the generation of the PPM in the new IRIS frontend. As shown in Figure 7.1, the new hardware includes an IRIS unit to process the internal edge and motion image information already generated by current ISPs and forge the PPM. Every time the ISP transfers a frame, it also updates the PPM metadata to provide the backend with the priority score of each region in the Framebuffer (FB).

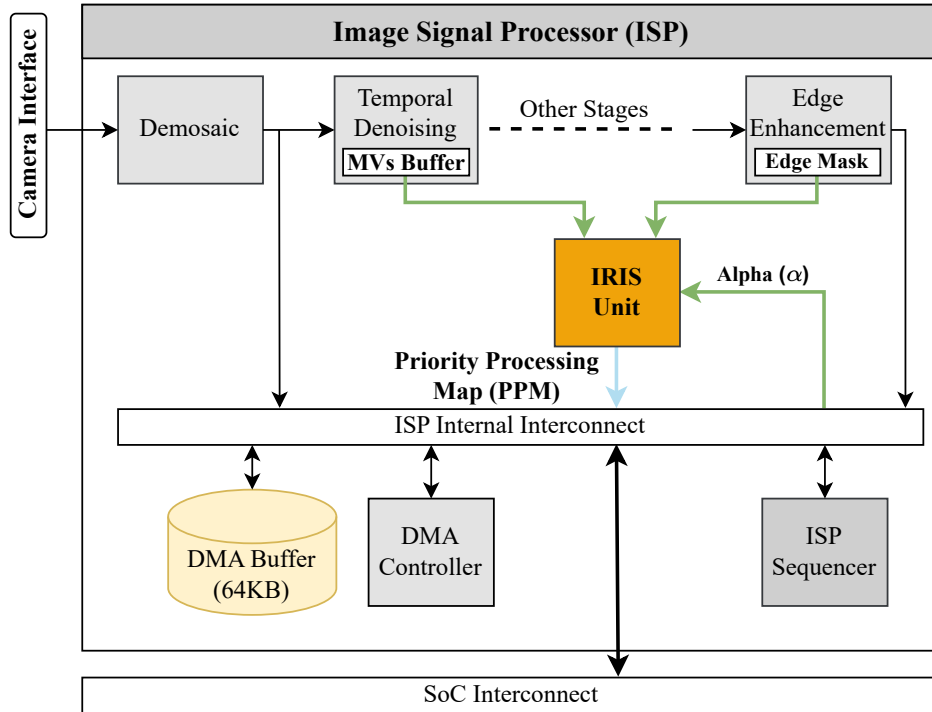


Figure 7.3: Integration of the new IRIS unit in the ISP to ensure zero-latency overhead.

ISP Pipeline Integration

We consider a baseline ISP that resembles literature [64, 20] and commercially available solutions [106, 9, 8] with support for TD and ME. Figure 7.3 illustrates the described standard ISP pipeline and the integration of the IRIS unit running in a parallel datapath. The ISP sequencer orchestrates the ISP stages to process the camera sensor’s raw data, generate the RGB and YUV color spaces (demaicing), and apply several image enhancements (e.g., white-balancing and tone mapping). The generated image is temporarily stored in the DMA buffer to smooth data flow within the ISP pipeline before the DMA controller transfers it to the main memory. The new IRIS unit interfaces with the rest of the ISP by accessing the EE edge mask, the MVs buffer, the alpha parameter from the framebuffer, and storing the PPM scores to the ISP’s DMA buffer.

The standard EE stage in modern ISPs employs unsharp masking, as explained in more detail in Subsection 2.1.2. In hardware, this method involves operating a 2D line buffer to process the raster-scan stream of image pixels from the previous stage and generate a Gaussian-filtered version of it. This filtered image is subtracted from the unfiltered one to create the edge mask. We modified the EE stage to expose this mask directly to the IRIS

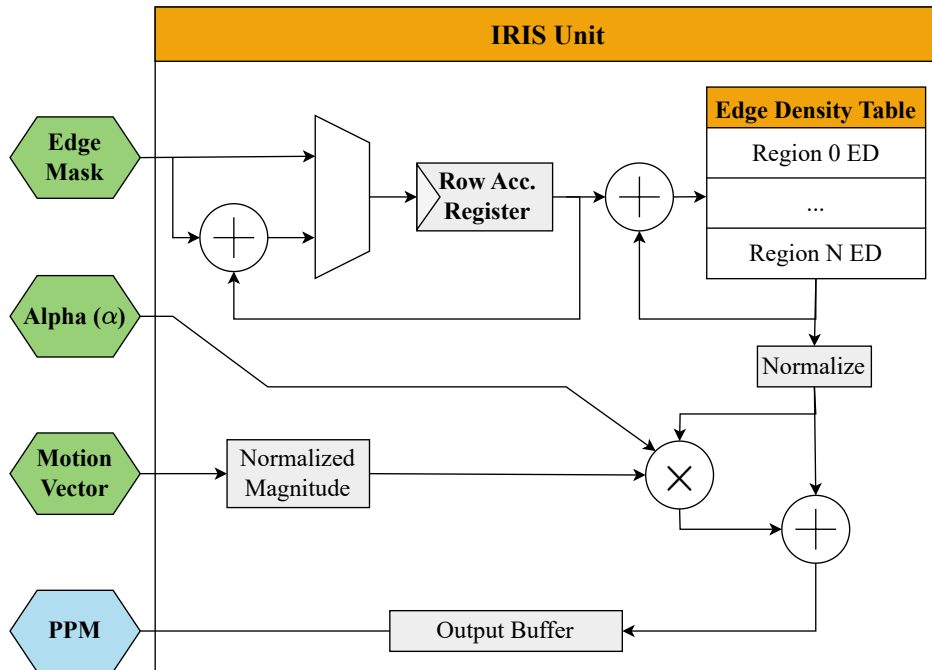


Figure 7.4: Architecture of the IRIS unit to compute coarse-grained saliency maps in real-time.

unit.

The ISP’s ME stage has a specialized on-chip buffer to store MVs. We propose adding a read port to support MV requests from the IRIS unit. Furthermore, we modeled a cycle-accurate RTL hardware description of the IRIS unit and its integration with the EE and ME to demonstrate that the new IRIS processing overlaps with the ISP pipeline and never stalls it.

IRIS Unit

A block diagram of the IRIS unit is shown in Figure 7.4. It features a simple microarchitecture to keep track of each image region’s EDM and MM scores. The unit only requires synchronization to correctly account for each EDM region’s corresponding edge mask values. Our design features a Row Accumulator Register that aggregates the edge values generated in the EE stage to compute the sum of each row of each region of the EDM. The Edge Density Table temporarily holds the partial EDM values for each region, allowing the accumulation of the region edge row sums. This accumulation process is repeated for each image row until completion, after which the process restarts for the next row of EDM regions.

In the last image row required to compute the EDM, the unit generates a final normalized EDM score every l cycles, where l is the length of the side of the image regions in which EDM divides the image. During these cycles, the unit retrieves the corresponding motion vector from the TD stage, calculates its magnitude, and computes the final PPM score. The unit transfers this score to an output buffer and, ultimately, to the DMA infrastructure through the internal ISP interconnect.

7.2.4 Augmenting the Vision Backend

The vision backend collaborates with the new IRIS frontend by accessing the PPM metadata and adjusting each region’s perceived motion’s contribution through the α parameter. Our design provides fundamental image information in real-time using a generic format, a map ranking image regions, that makes no assumptions about the application. Therefore, the backend is responsible for exploiting more specialized insights from the application to adapt its behavior based on it.

A viable approach to leverage IRIS scheme consists of tailoring the vision application to perform iterative image processing where the algorithm processes regions in steps based on their distinctiveness, as the PPM quantifies. This approach enables the backend to decide at what step to stop based on criteria such as the onset of diminishing returns. We illustrate the benefits of such an approach in a crucial emerging application, a state-of-the-art visual localization technique called ORB-SLAM [23, 113] that typically runs on CPUs [50].

We modified the ORB-SLAM codebase to include an iterative ORB Feature Extraction (FE) algorithm that does not require processing all the image regions, avoiding processing the ineffectual and the least significant ones. As a result, the backend can significantly reduce computation, memory, and energy requirements while providing the same accuracy. The upcoming subsections will focus on reformulating the FE (Feature Extraction and Matching) stage, a critical bottleneck in localization algorithms, in an iterative form.

Harnessing the PPM Information

This section elaborates on the localization algorithm-specific insights to exploit the PPM information. We continue to address the primary bottlenecks in our localization engine, focusing specifically on optimizing feature extraction and matching processes. We substantiate our discussion with more empirical evidence in Section 7.3.

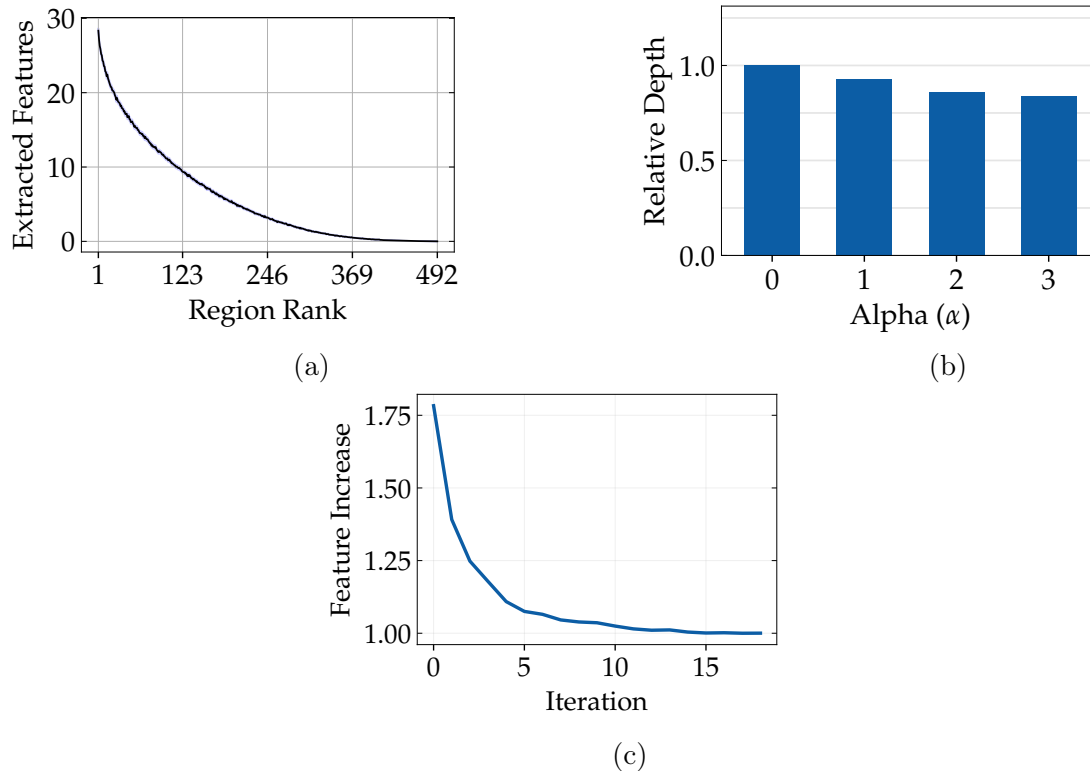


Figure 7.5: Analysis of the FE stage and the characteristics of our iterative solution to perform it.

Although only four correspondences are mathematically necessary for uniquely determining the camera’s pose, practical systems seek a more significant number to counteract the potential inaccuracies and noise of the observed 2D positions of the extracted features on the image. ORB-SLAM extracts thousands of ORB features and selects the most prominent ones (up to 2000 for the images of KITTI) to narrow down the computational complexity required in later processing stages. Our observations reveal that most ORB features concentrate on a small range of image regions (80% of features cluster in just 35% of regions). Thus, prioritizing them offers the greatest return on computational investment. As discussed in the Introduction Chapter, considerable image regions lack features (around 40% on average), rendering them ineffective for localization. Figure 1.3, previously introduced, shows a histogram that details the frequency distribution of regions based on the number of ORB features extracted from each KITTI sequence.

Another vital insight into pose estimation is that points closer to the camera play a crucial role for two reasons. First, the regions closer to the camera capture objects and scene attributes with greater scale and details, improving feature extraction and matching

robustness and accuracy. Second, due to the parallax effect, scene areas close to the camera exhibit more noticeable apparent motion, providing higher certainty in triangulating the pose. Figure 7.5b shows the role of MM in favoring regions proximal to the camera. It examines the average relative distance (depth) of detected ORB features to the camera within the top 50% of regions ranked by the PPM for different values of α compared to the case $\alpha = 0$.

On the other hand, processing regions of images capturing dynamic and moving objects unrelated to the static scene is unnecessary for localization purposes [145, 172, 5]. Besides, the localization algorithm naturally generates a trajectory prediction, implying that the systems could infer camera movements, enabling it to discern, for instance, whether a car is stationary, moving forward, or turning based on the video stream. We can leverage this ability to adjust the system’s α value based on the mentioned application insights. Setting α to 1 when the camera moves directs the MM to focus on proximal regions. In contrast, when the camera is stationary, setting α to -1 de-prioritizes regions with dynamic objects irrelevant to the static scene.

Moreover, we empirically evaluate the ability of the PPM metadata to rank image regions consistently. Figure 7.5a illustrates the average number of features extracted per region (y-axis) for each position of the PPM ranking (x-axis) on sequence 0 of the KITTI benchmark, showing a clear global trend that correlates PPM with the application level features.

In conclusion, the presented analyses demonstrated PPM’s effectiveness in precisely estimating regions with a high density of ORB features and closer to the camera. We next present the design of our iterative FE algorithm.

Iterative Feature Extraction

As detailed in Algorithm 1, our iterative algorithm processes regions in batches (for instance, 5% in each iteration) based on their PPM scores. It extracts features from each batch during each iteration and calculates the MFI (Marginal Feature Increase). The MFI is the percentage increase in new features obtained in the most recent iteration. The programmer can establish two parameters: N_{min} , the minimal conservative number of regions to process, and T_{MFI} , a threshold for each iteration MFI to indicate when further processing yields diminishing returns and thus stops processing. Figure 7.5c shows the MFI progression during the iterative processing of a typical frame. By setting an MFI threshold of 3% (

Algorithm 1 Iterative Feature Extraction

```
1: Define  $N_{min}, T_{MFI}$ 
2: Initialize  $Processed \leftarrow \emptyset, LastFeatureCount \leftarrow 0$ 
3: while not all regions processed do
4:    $Selected \leftarrow$  Top 5% PPM regions from the unprocessed pool
5:   Extract features from  $Selected$  regions
6:    $FeatureCount \leftarrow$  Count of features in  $Selected$ 
7:    $Processed \leftarrow Processed \cup Selected$ 
8:   if  $Processed > N_{min}$  then
9:      $MFI \leftarrow \frac{FeatureCount - LastFeatureCount}{LastFeatureCount}$ 
10:     $LastFeatureCount \leftarrow FeatureCount$ 
11:    if  $MFI < T_{MFI}$  then
12:      Break
13:    end if
14:  end if
15: end while
16: Continue pose estimation with extracted features
```

$T_{MFI} = 1.03$), the algorithm stops when an iteration adds fewer than 3% new features. In the example, it ceases processing after 65% of the regions, successfully retrieving more than 90% of baseline-identical features.

The advantages of this strategy are twofold. First, the system can skip processing areas with low information, typically with little to no significant features. Second, it enables the system to strategically focus on processing the most informative from the rest of the regions, stopping further processing based on a diminishing returns criterion. This criterion acknowledges that while processing each region incurs a consistent cost, the benefits, represented by the potential number of features in each region, increasingly decrease for areas with lower PPM scores.

Figure 7.6 qualitatively illustrates the benefits of our approach (detailed quantitative evaluation is provided in Section 7.3) by comparing the baseline approach where the system processes every image region always, a software-only version where the backend generates an equivalent PPM (MG box in the figure) to inform the iterative FE, and IRIS, where the ISP provides this information. Our approach reduces computations and memory accesses to the frame buffer, aiming to get the best accuracy for the computation resources spent processing the image.

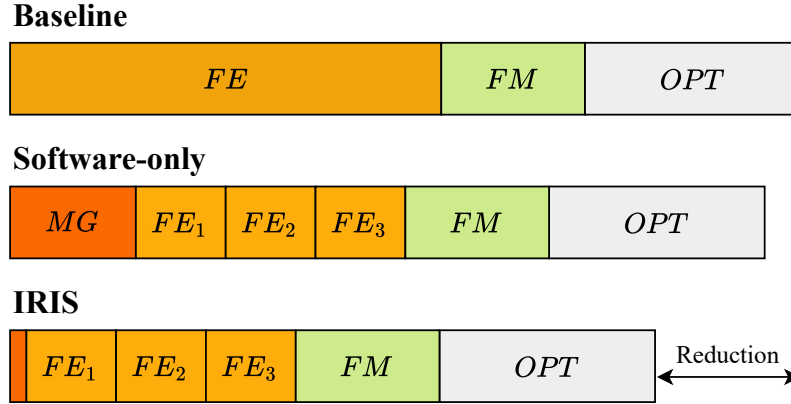


Figure 7.6: Comparison of baseline, software-only, and IRIS (not at scale).

7.3 Experimental Results

This section evaluates the proposed ISP-software collaboration approach, contrasting it with a purely software-based method that produces an analogous image region ranking.

In our study, we statically set the MFI threshold (T_{MFI}) that guides the early termination of the iterative feature extraction algorithm while also ensuring processing a minimum of 30% of image regions (N_{min}). Additionally, we adjusted ORB-SLAM to fall back to the baseline’s feature extraction during system initialization, relocalization, and tracking loss, ensuring the system’s robustness.

We compare the performance, energy efficiency, and accuracy of three versions of this application running in our baseline mobile ARM Cortex A72 CPU:

- **Baseline:** Original ORB-SLAM version optimized for ARM architecture.
- **Software-only (SO):** A modified version of ORB-SLAM that implements our iterative feature extraction algorithm and pre-processes the entire image to generate an equivalent PPM metric to guide it.
- **IRIS:** A enhanced version of ORB-SLAM that incorporates the IRIS frontend interface. This version delegates the PPM generation to the ISP, which it produces at a minimal cost by amortizing existing computations.

For clarity, we use the terms IRIS-TH and SO-TH to denote our IRIS scheme or its software-only counterpart with a static MFI threshold (T_{MFI}) value of TH, respectively. To

evaluate the performance and robustness of ORB-SLAM, we employed the entire suite of sequences from the KITTI dataset, replicating the original assessment methodology used in its validation [112].

7.3.1 Region and Feature Coverage Analyses

We conducted detailed analyses on image region and feature coverage to evaluate the effectiveness of IRIS to capitalize on existing ISP computations to forge PPM’s new metadata and rank distinct regions for our localization system. Our goal was to determine the required region coverage to extract a specific fraction of features comparable to the baseline and, conversely, the proportion of regions needed to process a target percentage of identical features.

Figure 7.7a depicts how our ranking method (IRIS) identifies a certain percentage of features identical to the baseline (x-axis) using an average percentage of regions (y-axis) in comparison to random sampling (RAND) and a lower bound (MIN) across the entire KITTI dataset. Random (RAND) sampling necessitates processing a proportion of regions almost equal to the desired percentage of identical features. In contrast, IRIS PPM metadata effectively identifies the most distinctive regions. For instance, to extract 80% of the original features, our method requires processing only about 55% of the image regions on average. Similarly, to recover 90% of the features, it needs to process just 69% of the regions. We computed the lower bound (MIN), extracting all the features from the frame, sorting the regions by feature count, and calculating the minimum regions required to attain each target feature percentage. IRIS showcases its effectiveness in selective region processing as it is close to the lower bound, only requiring processing 20% extra regions to obtain 80% of features compared with the lower bound.

We now focus on the coverage achieved by our proposed iterative feature extraction method elaborated in Section 7.2.4. This method, one of the potential options to harness IRIS metadata, selectively bypasses ineffective regions and empowers the programmer to make informed decisions, balancing computational costs with approximate results. It leverages the insights gained from PPM metadata and its application-specific properties demonstrated in the preceding figure. Figure 7.7b illustrates the average percentage of processed regions and the recovered identical features for a subset of representative MFI thresholds. Notably, as the MFI threshold increases, resulting in more aggressive region reduction, feature coverage decreases slower. Lower IRIS-ranked regions typically contain fewer features, underscoring the

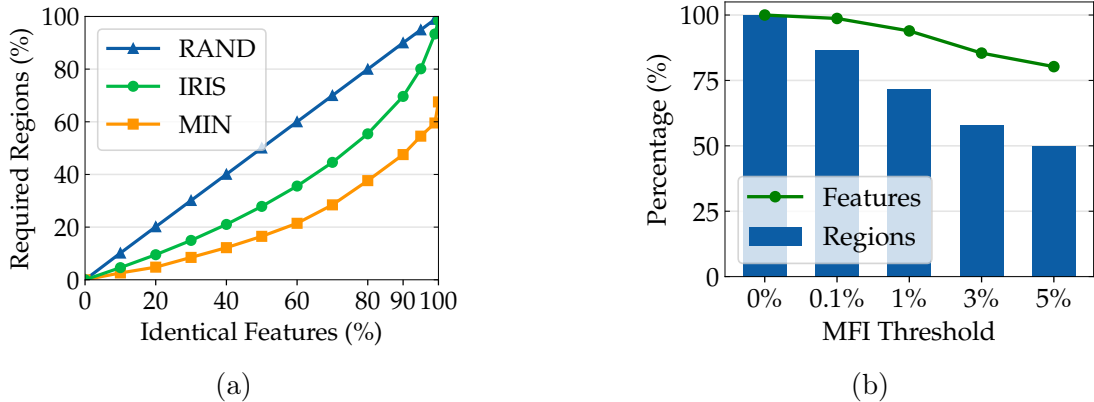


Figure 7.7: Sensitivity analyses of image region and feature coverage. Figure 7.7a contrasts IRIS with random sampling and the ideal minimum. Figure 7.7b evaluates their response for different values of MFI threshold during the iterative FE execution.

efficacy of our technique in prioritizing distinctive regions rich in features and deprioritizing those with lower feature counts.

7.3.2 Accuracy Analysis

Our analysis of localization accuracy against the baseline utilizes the Absolute Pose Error (APE), a crucial metric for assessing localization algorithms. APE quantifies the discrepancy between the ground truth pose and the estimated pose at each timestep. High localization accuracy is vital, especially for AD systems requiring centimeter-level precision and only using onboard sensors. This precision surpasses that of external commercial technologies like GPS. Therefore, we adopted a conservative methodology, employing median and maximum APE metrics for our accuracy evaluation. Accounting for the maximum APE ensures we cover the worst-case scenario on the dataset.

Figure 7.8 displays the maximum and median APE across the evaluated KITTI sequences. Our approach maintains an error below 4mm for MFI thresholds within the 0 to 3% range. As thresholds increase beyond this range, noticeable errors emerge, suggesting the need for advanced methods like dynamic MFI thresholding and the exploitation of more specific insights. Consequently, we determined that an MFI threshold of 3% is optimal for our experiments, and it has been adopted as the standard unless specified otherwise.

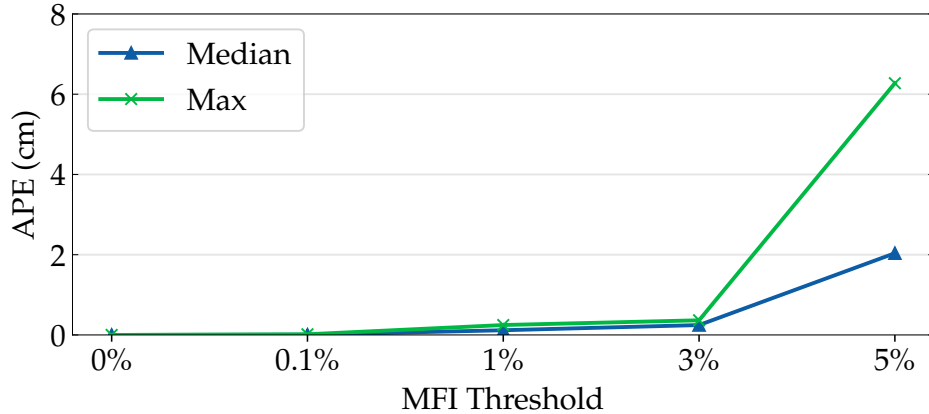


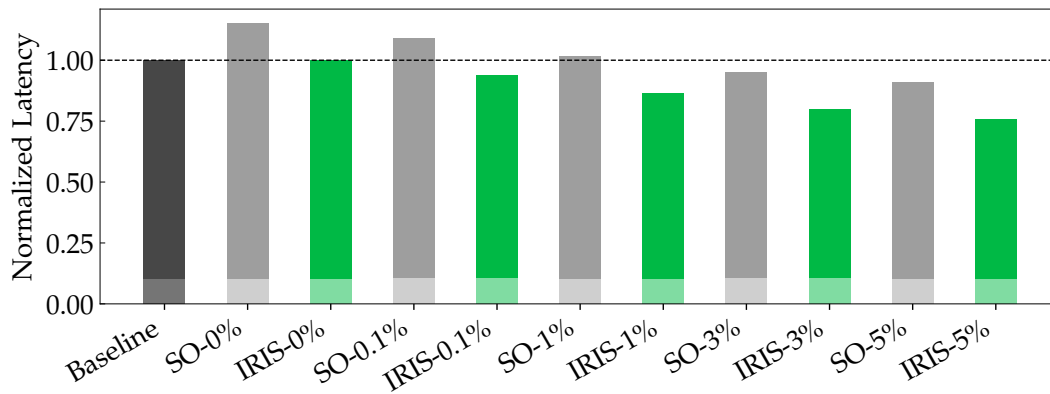
Figure 7.8: Maximum Absolute Pose Error (APE) for different MFI thresholds.

7.3.3 Latency Analysis

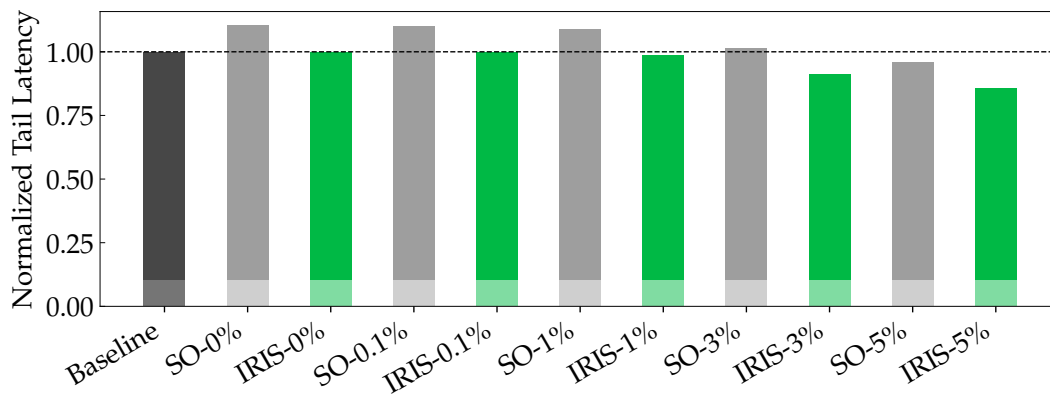
Our main performance evaluation of our proposed solution focuses on tail and end-to-end latency within the CV pipeline, excluding the image sensor. Tail latency is particularly critical in CV systems used in AD, where response times are subject to stringent constraints. Figure 7.9a presents the normalized average latency, comparing our method, IRIS, with the baseline and SO configurations. Moreover, Figure 7.9b showcases the 99th tail latency. To evaluate the overheads of the different approaches, we also assess IRIS and SO using an MFI threshold of 0%. This scenario is functionally equivalent to the baseline, with the additional cost of generating the ranking.

The results show that the SO’s benefits are minor in average latency and practically null in tail latency when not harming it (SO-0.1%, 1%, and 3%). The overhead of computing the PPM for every frame offsets the potential benefits, forcing the programmer to utilize an aggressive MFI threshold to realize any benefit. This finding may explain why experts rarely implement such software-only optimizations in vision algorithms like ORB-SLAM and motivates the adoption of our novel CV architecture.

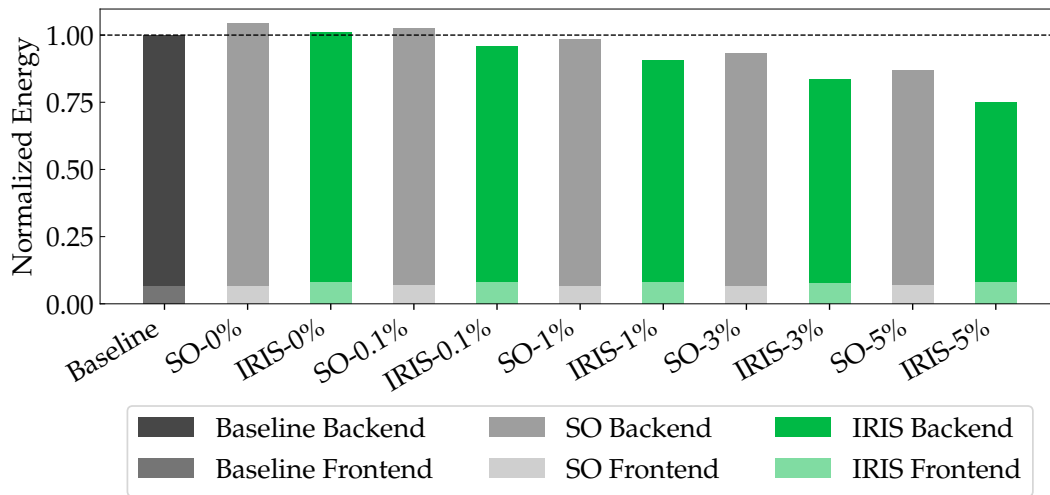
Conversely, IRIS significantly surpasses the SO implementation in reducing average and tail latency. It reduces the average latency by 20% and in tail latency by 9% compared to the baseline. Moreover, it reduces average latency by 16% and tail latency by 10% relative to SO. IRIS surpasses its contenders since it adds minimal overhead even in the worst-case scenario, mainly for reading the PPM (a few KBs) and the control to implement the iterative feature extraction. Standard, intermediate ISP computations generate the PPM, and concurrent ISP tasks effectively hide its processing time. Consequently, IRIS enables



(a)



(b)



(c)

Figure 7.9: Comparison of Normalized Latencies (7.9a), Normalized Tail Latencies (7.9b), and Normalized Energy (7.9c) for Baseline, Software-Only (SO), and IRIS.

strategies for selectively processing image regions, greatly benefiting real-time critical CV systems such as the one evaluated in this work.

7.3.4 Energy Analysis

Figure 7.9c illustrates the energy consumption comparison among the Baseline, SO, and IRIS configurations. IRIS demonstrates a significant reduction in energy use, achieving 16% less than the Baseline and 10% less than the SO scheme. These results emphasize the advantages of unleashing the synergies between SoC components in the vision pipeline. The additional energy savings of IRIS come from fewer CPU framebuffer accesses and the minimal cost of computing the PPM that IRIS generates by repurposing the existing internal byproducts of an already highly specialized energy-efficient unit for image processing, the ISP.

7.3.5 Area and Power Analysis

We assess the area and power costs of IRIS using the approach described in Section 3.4, considering that the IRIS unit supports up to Full HD resolution, a current standard. IRIS incurs a minimal area overhead of $0.017mm^2$, primarily due to the new read port to the MVs (Motion Vectors) on-chip buffer and the enhancements in the ISP. This increase in area is insignificant when compared to the areas of typical SoCs, such as the $100mm^2$ of a Mediatek A72 cluster, the $88mm^2$ of the Apple A14, or the $350mm^2$ of Nvidia Xavier. Additionally, IRIS dissipates 6.2mW due to the new SRAM accesses and the final PPM metric computation that requires few operations to obtain the motion vector gradients and edge density.

7.4 Conclusions

Continuous Vision systems face significant challenges regarding latency and energy consumption. To overcome them, we propose IRIS (Image Region ISP-Software Prioritization), a novel co-designed CV architecture that empowers its frontend to expose two critical fundamental image metrics internally generated in modern ISPs for its ordinary processing: edge density and motion intensity. Our contribution opens the door to more efficient backend processing by actively leveraging the new information unleashed by the IRIS frontend.

As a case study, we demonstrate how to harness the new IRIS image ranking metadata to prioritize the processing of image regions through an iterative feature extraction algorithm. Our evaluation shows that our proposal can significantly reduce backend computations while maintaining high accuracy with simple, lightweight hardware augmentations. Consequently, IRIS's efficiency and flexibility make it a significant leap forward in CV SoC design.

8

Conclusions

This chapter summarizes the main contributions of this thesis and outlines some promising open research areas for future work.

8.1 Conclusions

In this thesis, we have effectively addressed the critical challenges of latency and energy consumption in mobile CV (Continuous Vision) systems, presenting a suite of comprehensive optimizations that advance the boundaries of performance and energy efficiency in this domain. Our journey began with a thorough analysis of the visual localization engine, where we pinpointed feature extraction as the primary bottleneck. Addressing this, we introduced LOCATOR (Low-power ORB aCcelerator for AuTonomOus caRs), an innovative hardware accelerator that significantly enhances the detection and processing of ORB (Oriented FAST and Rotated BRIEF) features, outperforming existing energy efficiency and speed solutions.

Building on this, we recognized the need for more versatile and programmable so-

lutions in mobile CV systems. Our response was SLIDEX (SLIDing window EXtension for image processing), a novel vector ISA (Instruction Set Architecture) extension designed to optimize SWP (Sliding Window Processing). SLIDEX dramatically increases DLP (Data Level Parallelism) and reduces data movement, showcasing remarkable improvements in vital image processing tasks.

We then broadened our perspective to SoC-level optimizations with δ LTA (*don't Look Twice, it's Alright*), a strategy that capitalizes on the inherent properties of vision input streams to reduce redundant processing. By enabling the frontend's ISP (Image Signal Processor) to selectively filter ineffectual regions and the backend to focus on distinct image regions, δ LTA considerably reduces unnecessary computations and memory accesses, leading to significant latency and energy efficiency gains.

Finally, we presented IRIS (Image Region ISP-Software Prioritization), a novel technique that leverages computation byproducts from the frontend's ISP to optimize CV processing. By prioritizing image regions based on their detail and motion, IRIS enables more efficient backend processing and avoids processing unimportant parts of the images.

8.2 Open-Research Areas

Building on the insights gained from this thesis, we spotlight three forward-looking research areas that promise to elevate the capabilities and efficiency of mobile CV systems through architectural innovations and advanced computational strategies:

1. **Leveraging Heterogeneous Sensory Inputs:** Investigating architectures that exploit heterogeneous properties—such as multi-camera systems, multi-resolution, multi-frame rate capabilities, and multi-modal sensors, including event cameras—represents an interesting research direction. Event cameras, which capture pixel-level changes in intensity instead of static frames, offer a dynamic and efficient way to process visual information. By integrating these with traditional sensors in a multi-modal setup, systems can harness the unique advantages of each sensor type, optimizing computational demands, accuracy, and energy efficiency. Research in this area could focus on developing algorithms and hardware that dynamically balance and fuse data from these diverse sources, maximizing the system's performance while minimizing energy consumption.

2. **Advanced Frontend and Backend Collaboration:** Another promising path lies in further exploring innovative collaboration models between the frontend and backend, particularly utilizing advanced machine learning techniques. This direction involves enhancing the frontend's capability to conduct more advanced preprocessing and interpretation tasks, significantly offloading the computational burden from the backend. Employing machine learning techniques to process and transmit higher-level semantic information can streamline the overall data processing pipeline, reducing latency and energy usage while maintaining or even enhancing system performance.
3. **In-Sensor Computing:** A transformative approach involves moving computations closer to, or directly within, the sensor itself, an advancement known as in-sensor computing. This method addresses and alleviates the communication bottleneck between sensors and processing units. By integrating processing capabilities directly within the sensor, data can be preprocessed at the source, dramatically reducing the volume of data that needs to be transmitted and processed downstream. This shift promises significant reductions in energy consumption and latency and opens up new possibilities for real-time processing and responsiveness in mobile CV applications.

Bibliography

- [1] *A survey of techniques for energy efficient on-chip communication | Proceedings of the 40th annual Design Automation Conference*. URL: <https://dl.acm.org/doi/abs/10.1145/775832.776059> (visited on 04/02/2024).
- [2] *A14 Bionic - Apple - WikiChip*. en. URL: <https://en.wikichip.org/wiki/apple/ax/a14> (visited on 04/28/2023).
- [3] *About Face ID advanced technology*. en. URL: <https://support.apple.com/en-us/102381> (visited on 04/15/2024).
- [4] *Adding mainline Arm Frame Buffer Compression support for Rockchip*. en. URL: <https://www.collabora.com/news-and-blog/blog/2020/04/08/adding-mainline-arm-frame-buffer-compression-support-for-rockchip/> (visited on 07/08/2023).
- [5] Yong-bao Ai et al. “Visual SLAM in dynamic environments based on object detection”. In: *Defence Technology* 17.5 (Oct. 2021), pp. 1712–1721. ISSN: 2214-9147. DOI: [10.1016/j.dt.2020.09.012](https://doi.org/10.1016/j.dt.2020.09.012). URL: <https://www.sciencedirect.com/science/article/pii/S2214914720304402> (visited on 11/22/2023).
- [6] Jorge Albericio et al. “Cnvlutin: Ineffectual-Neuron-Free Deep Neural Network Computing”. In: *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. ISSN: 1063-6897. June 2016, pp. 1–13. DOI: [10.1109/ISCA.2016.11](https://doi.org/10.1109/ISCA.2016.11).
- [7] Stefano Aldegheri et al. “Data Flow ORB-SLAM for Real-time Performance on Embedded GPU Boards”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. ISSN: 2153-0866. Nov. 2019, pp. 5370–5375. DOI: [10.1109/IROS40897.2019.8967814](https://doi.org/10.1109/IROS40897.2019.8967814). URL: <https://ieeexplore.ieee.org/document/8967814> (visited on 03/26/2024).

- [8] *AP0100CS*. URL: <https://www.onsemi.com/products/sensors/image-signal-processors-isps/ap0100cs> (visited on 04/26/2023).
- [9] *AP0201AT*. URL: <https://www.onsemi.com/products/sensors/image-signal-processors-isps/ap0201at> (visited on 04/26/2023).
- [10] *Apple unveils M3, M3 Pro, and M3 Max, the most advanced chips for a personal computer*. ur-CM. URL: <https://www.apple.com/cm/newsroom/2023/10/apple-unveils-m3-m3-pro-and-m3-max-the-most-advanced-chips-for-a-personal-computer/> (visited on 04/15/2024).
- [11] *Apple Vision Pro*. en-US. URL: <https://www.apple.com/apple-vision-pro/> (visited on 11/29/2023).
- [12] *AR & VR - Worldwide | Statista Market Forecast*. en. URL: <https://www.statista.com/outlook/amo/ar-vr/worldwide> (visited on 03/31/2024).
- [13] ARM. *Neon Programmer's Guide*. 2013.
- [14] *Autopilot and Full Self-Driving Capability | Tesla Support*. en. URL: <https://www.tesla.com/support/autopilot> (visited on 03/31/2024).
- [15] Brendan Barry et al. "Always-on vision processing unit for mobile applications". In: *IEEE Micro* 35.2 (2015). Publisher: IEEE, pp. 56–66.
- [16] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. "SURF: Speeded Up Robust Features". en. In: *Computer Vision – ECCV 2006*. Ed. by Aleš Leonardis, Horst Bischof, and Axel Pinz. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, pp. 404–417. ISBN: 978-3-540-33833-8. DOI: [10.1007/11744023_32](https://doi.org/10.1007/11744023_32).
- [17] M. Bigas et al. "Review of CMOS image sensors". In: *Microelectronics Journal* 37.5 (May 2006), pp. 433–451. ISSN: 0026-2692. DOI: [10.1016/j.mejo.2005.07.002](https://doi.org/10.1016/j.mejo.2005.07.002). URL: <https://www.sciencedirect.com/science/article/pii/S0026269205002764> (visited on 04/04/2024).
- [18] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).
- [19] A. Buades, B. Coll, and J. M. Morel. "A Review of Image Denoising Algorithms, with a New One". In: *Multiscale Modeling & Simulation* 4.2 (Jan. 2005). Publisher: Society for Industrial and Applied Mathematics, pp. 490–530. ISSN: 1540-3459. DOI: [10.1137/040616024](https://doi.org/10.1137/040616024). URL: <https://epubs.siam.org/doi/abs/10.1137/040616024> (visited on 04/15/2024).

- [20] Mark Buckler, Suren Jayasuriya, and Adrian Sampson. “Reconfiguring the Imaging Pipeline for Computer Vision”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. ISSN: 2380-7504. Oct. 2017, pp. 975–984. DOI: [10.1109/ICCV.2017.111](https://doi.org/10.1109/ICCV.2017.111).
- [21] Mark Buckler et al. “EVA²: Exploiting Temporal Redundancy in Live Computer Vision”. In: *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. ISSN: 2575-713X. June 2018, pp. 533–546. DOI: [10.1109/ISCA.2018.00051](https://doi.org/10.1109/ISCA.2018.00051).
- [22] Michael Calonder et al. “Brief: Binary robust independent elementary features”. In: *European conference on computer vision*. Springer, 2010, pp. 778–792.
- [23] Carlos Campos et al. “ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial, and Multimap SLAM”. In: *IEEE Transactions on Robotics* 37.6 (Dec. 2021). Conference Name: IEEE Transactions on Robotics, pp. 1874–1890. ISSN: 1941-0468. DOI: [10.1109/TR0.2021.3075644](https://doi.org/10.1109/TR0.2021.3075644).
- [24] Lukas Cavigelli, Philippe Degen, and Luca Benini. “CBinfer: Change-Based Inference for Convolutional Neural Networks on Video Data”. In: *Proceedings of the 11th International Conference on Distributed Smart Cameras*. ICDSC 2017. New York, NY, USA: Association for Computing Machinery, Sept. 2017, pp. 1–8. ISBN: 978-1-4503-5487-5. DOI: [10.1145/3131885.3131906](https://doi.org/10.1145/3131885.3131906). URL: <https://dl.acm.org/doi/10.1145/3131885.3131906> (visited on 04/11/2023).
- [25] Nanchini Chandramoorthy et al. “Exploring architectural heterogeneity in intelligent vision systems”. In: *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2015, pp. 1–12.
- [26] Dimitris Chatzopoulos et al. “Mobile Augmented Reality Survey: From Where We Are to Where We Go”. In: *IEEE Access* 5 (2017). Conference Name: IEEE Access, pp. 6917–6950. ISSN: 2169-3536. DOI: [10.1109/ACCESS.2017.2698164](https://doi.org/10.1109/ACCESS.2017.2698164).
- [27] Yu-Hsin Chen et al. “14.5 Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks”. In: *2016 IEEE International Solid-State Circuits Conference (ISSCC)*. ISSN: 2376-8606. Jan. 2016, pp. 262–263. DOI: [10.1109/ISSCC.2016.7418007](https://doi.org/10.1109/ISSCC.2016.7418007).

- [28] Jun Cheng et al. “A review of visual SLAM methods for autonomous driving vehicles”. en. In: *Engineering Applications of Artificial Intelligence* 114 (Sept. 2022), p. 104992. ISSN: 0952-1976. DOI: [10.1016/j.engappai.2022.104992](https://doi.org/10.1016/j.engappai.2022.104992). URL: <https://www.sciencedirect.com/science/article/pii/S0952197622001853> (visited on 02/16/2023).
- [29] Jaehyuk Choi et al. “An Energy/Illumination-Adaptive CMOS Image Sensor With Reconfigurable Modes of Operations”. In: *IEEE Journal of Solid-State Circuits* 50.6 (June 2015). Conference Name: IEEE Journal of Solid-State Circuits, pp. 1438–1450. ISSN: 1558-173X. DOI: [10.1109/JSSC.2015.2420678](https://doi.org/10.1109/JSSC.2015.2420678). URL: <https://ieeexplore.ieee.org/abstract/document/7095624> (visited on 04/02/2024).
- [30] Yoojin Choi, Mostafa El-Khamy, and Jungwon Lee. “Universal Deep Neural Network Compression”. In: *IEEE Journal of Selected Topics in Signal Processing* 14.4 (May 2020). Conference Name: IEEE Journal of Selected Topics in Signal Processing, pp. 715–726. ISSN: 1941-0484. DOI: [10.1109/JSTSP.2020.2975903](https://doi.org/10.1109/JSTSP.2020.2975903).
- [31] Lucian Codrescu et al. “Hexagon DSP: An architecture optimized for mobile multimedia and communications”. In: *IEEE Micro* 34.2 (2014). Publisher: IEEE, pp. 34–43.
- [32] H. Collewijn and J. B. Smeets. “Early components of the human vestibulo-ocular response to head rotation: latency and gain”. eng. In: *Journal of Neurophysiology* 84.1 (July 2000), pp. 376–389. ISSN: 0022-3077. DOI: [10.1152/jn.2000.84.1.376](https://doi.org/10.1152/jn.2000.84.1.376).
- [33] Jamie Cross et al. “Using Extended Reality in Flight Simulators: A Literature Review”. In: *IEEE Transactions on Visualization and Computer Graphics* 29.9 (Sept. 2023). Conference Name: IEEE Transactions on Visualization and Computer Graphics, pp. 3961–3975. ISSN: 1941-0506. DOI: [10.1109/TVCG.2022.3173921](https://doi.org/10.1109/TVCG.2022.3173921). URL: <https://ieeexplore.ieee.org/abstract/document/9772329> (visited on 04/01/2024).
- [34] *Cruise Self Driving Cars | Autonomous Vehicles | Driverless Rides & Delivery*. en. URL: <https://getcruise.com/> (visited on 03/31/2024).
- [35] Igor Cvišić, Ivan Marković, and Ivan Petrović. “SOFT2: Stereo Visual Odometry for Road Vehicles Based on a Point-to-Epipolar-Line Metric”. In: *IEEE Transactions on Robotics* (2022). Publisher: IEEE.

- [36] Kostadin Dabov et al. “Image Denoising by Sparse 3-D Transform-Domain Collaborative Filtering”. In: *IEEE Transactions on Image Processing* 16.8 (Aug. 2007). Conference Name: IEEE Transactions on Image Processing, pp. 2080–2095. ISSN: 1941-0042. DOI: [10.1109/TIP.2007.901238](https://doi.org/10.1109/TIP.2007.901238). URL: <https://ieeexplore.ieee.org/abstract/document/4271520> (visited on 04/15/2024).
- [37] Klaus David and Alexander Flach. “CAR-2-X and Pedestrian Safety”. In: *IEEE Vehicular Technology Magazine* 5.1 (Mar. 2010). Conference Name: IEEE Vehicular Technology Magazine, pp. 70–76. ISSN: 1556-6080. DOI: [10.1109/MVT.2009.935536](https://doi.org/10.1109/MVT.2009.935536).
- [38] Guang Deng. “A Generalized Unsharp Masking Algorithm”. In: *IEEE Transactions on Image Processing* 20.5 (May 2011). Conference Name: IEEE Transactions on Image Processing, pp. 1249–1261. ISSN: 1941-0042. DOI: [10.1109/TIP.2010.2092441](https://doi.org/10.1109/TIP.2010.2092441). URL: <https://ieeexplore.ieee.org/abstract/document/5635330> (visited on 11/22/2023).
- [39] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. “SuperPoint: Self-Supervised Interest Point Detection and Description”. In: 2018, pp. 224–236. URL: https://openaccess.thecvf.com/content_cvpr_2018_workshops/w9/html/DeTone_SuperPoint_Self-Supervised_Interest_CVPR_2018_paper.html (visited on 02/23/2024).
- [40] Natalia Dużmańska, Paweł Strojny, and Agnieszka Strojny. “Can Simulator Sickness Be Avoided? A Review on Temporal Aspects of Simulator Sickness”. en. In: *Frontiers in Psychology* 9 (2018). Publisher: Frontiers Media SA. DOI: [10.3389/fpsyg.2018.02132](https://doi.org/10.3389/fpsyg.2018.02132). URL: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6232264/> (visited on 04/01/2024).
- [41] A. El Gamal and H. Eltoukhy. “CMOS image sensors”. In: *IEEE Circuits and Devices Magazine* 21.3 (May 2005). Conference Name: IEEE Circuits and Devices Magazine, pp. 6–20. ISSN: 1558-1888. DOI: [10.1109/MCD.2005.1438751](https://doi.org/10.1109/MCD.2005.1438751). URL: <https://ieeexplore.ieee.org/abstract/document/1438751> (visited on 04/08/2024).
- [42] *Energy characterization and optimization of image sensing toward continuous mobile vision | Proceeding of the 11th annual international conference on Mobile systems, applications, and services*. URL: <https://dl.acm.org/doi/abs/10.1145/2462456.2464448> (visited on 04/02/2024).

- [43] Weikang Fang et al. “FPGA-based ORB feature extraction for real-time visual SLAM”. In: *2017 International Conference on Field Programmable Technology (ICFPT)*. IEEE, 2017, pp. 275–278.
- [44] Yu Feng, Paul Whatmough, and Yuhao Zhu. “ASV: Accelerated Stereo Vision System”. In: *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO ’52. New York, NY, USA: Association for Computing Machinery, Oct. 2019, pp. 643–656. ISBN: 978-1-4503-6938-1. DOI: [10.1145/3352460.3358253](https://doi.org/10.1145/3352460.3358253). URL: <https://doi.org/10.1145/3352460.3358253> (visited on 02/16/2023).
- [45] Michael J Flynn. “Very high-speed computing systems”. In: *Proceedings of the IEEE* 54.12 (1966). Publisher: IEEE, pp. 1901–1909.
- [46] Denis Fortun, Patrick Bouthemy, and Charles Kervrann. “Optical flow modeling and computation: A survey”. In: *Computer Vision and Image Understanding*. Image Understanding for Real-world Distributed Video Networks 134 (May 2015), pp. 1–21. ISSN: 1077-3142. DOI: [10.1016/j.cviu.2015.02.008](https://doi.org/10.1016/j.cviu.2015.02.008). URL: <https://www.sciencedirect.com/science/article/pii/S1077314215000429> (visited on 11/22/2023).
- [47] Jorge Fuentes-Pacheco, José Ruiz-Ascencio, and Juan Manuel Rendón-Mancha. “Visual simultaneous localization and mapping: a survey”. In: *Artificial Intelligence Review* 43.1 (2015). Publisher: Springer, pp. 55–81.
- [48] Hamed Kiani Galoogahi et al. “Need for Speed: A Benchmark for Higher Frame Rate Object Tracking”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. ISSN: 2380-7504. Oct. 2017, pp. 1134–1143. DOI: [10.1109/ICCV.2017.128](https://doi.org/10.1109/ICCV.2017.128).
- [49] Dorian Galvez-López and Juan D. Tardos. “Bags of Binary Words for Fast Place Recognition in Image Sequences”. In: *IEEE Transactions on Robotics* 28.5 (Oct. 2012). Conference Name: IEEE Transactions on Robotics, pp. 1188–1197. ISSN: 1941-0468. DOI: [10.1109/TR0.2012.2197158](https://doi.org/10.1109/TR0.2012.2197158).
- [50] Yiming Gan et al. “Eudoxus: Characterizing and accelerating localization in autonomous machines industry track paper”. In: *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 827–840.
- [51] Yiming Gan et al. “Low-Latency Proactive Continuous Vision”. In: *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*. PACT ’20. New York, NY, USA: Association for Computing Machinery, Sept. 2020, pp. 329–342. ISBN: 978-1-4503-8075-1. DOI: [10.1145/3410463.3414650](https://doi.org/10.1145/3410463.3414650). URL: <https://doi.org/10.1145/3410463.3414650> (visited on 02/16/2023).

- [52] Andreas Geiger, Philip Lenz, and Raquel Urtasun. “Are we ready for autonomous driving? the kitti vision benchmark suite”. In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2012, pp. 3354–3361.
- [53] *Getting the Most Out of Delta Color Compression*. en-GB. URL: <https://gpuopen.com/learn/dcc-overview/> (visited on 07/08/2023).
- [54] Amir Gholami et al. *A Survey of Quantization Methods for Efficient Neural Network Inference*. arXiv:2103.13630 [cs]. June 2021. DOI: [10.48550/arXiv.2103.13630](https://doi.org/10.48550/arXiv.2103.13630). URL: <http://arxiv.org/abs/2103.13630> (visited on 04/12/2023).
- [55] *Global autonomous car market size*. en. URL: <https://www.statista.com/statistics/428692/projected-size-of-global-autonomous-vehicle-market-by-vehicle-type/> (visited on 03/31/2024).
- [56] David E Goldberg, Robert Lingle, et al. “Alleles, loci, and the traveling salesman problem”. In: *Proceedings of an international conference on genetic algorithms and their applications*. Vol. 154. Lawrence Erlbaum, Hillsdale, NJ, 1985, pp. 154–159.
- [57] *Google Pixel Phones*. es. URL: <https://store.google.com/es/category/phones?hl=es> (visited on 04/15/2024).
- [58] Michael Grupp. *evo: Python package for the evaluation of odometry and SLAM*. 2017. URL: <https://github.com/MichaelGrupp/evo>.
- [59] Peizhen Guo and Wenjun Hu. “Potluck: Cross-Application Approximate Deduplication for Computation-Intensive Mobile Applications”. In: *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS ’18. New York, NY, USA: Association for Computing Machinery, Mar. 2018, pp. 271–284. ISBN: 978-1-4503-4911-6. DOI: [10.1145/3173162.3173185](https://doi.org/10.1145/3173162.3173185). URL: <https://doi.org/10.1145/3173162.3173185> (visited on 02/14/2023).
- [60] Ayah Hamad and Bochen Jia. “How Virtual Reality Technology Has Changed Our Lives: An Overview of the Current and Potential Applications and Limitations”. eng. In: *International Journal of Environmental Research and Public Health* 19.18 (Sept. 2022), p. 11278. ISSN: 1660-4601. DOI: [10.3390/ijerph191811278](https://doi.org/10.3390/ijerph191811278).

- [61] Rehan Hameed et al. “Understanding sources of inefficiency in general-purpose chips”. In: *Proceedings of the 37th annual international symposium on Computer architecture*. ISCA '10. New York, NY, USA: Association for Computing Machinery, June 2010, pp. 37–47. ISBN: 978-1-4503-0053-7. DOI: [10.1145/1815961.1815968](https://doi.org/10.1145/1815961.1815968). URL: <https://dl.acm.org/doi/10.1145/1815961.1815968> (visited on 04/14/2023).
- [62] Song Han, Huizi Mao, and William J. Dally. “Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding”. In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2016. URL: <http://arxiv.org/abs/1510.00149> (visited on 04/12/2023).
- [63] Patrick Hansen et al. “ISP4ML: The Role of Image Signal Processing in Efficient Deep Learning Vision Systems”. In: *2020 25th International Conference on Pattern Recognition (ICPR)*. ISSN: 1051-4651. Jan. 2021, pp. 2438–2445. DOI: [10.1109/ICPR48806.2021.9411985](https://doi.org/10.1109/ICPR48806.2021.9411985). URL: <https://ieeexplore.ieee.org/document/9411985> (visited on 03/26/2024).
- [64] James Hegarty et al. “Darkroom: compiling high-level image processing code into hardware pipelines”. In: *ACM Transactions on Graphics* 33.4 (July 2014), 144:1–144:11. ISSN: 0730-0301. DOI: [10.1145/2601097.2601174](https://doi.org/10.1145/2601097.2601174). URL: <https://dl.acm.org/doi/10.1145/2601097.2601174> (visited on 04/26/2023).
- [65] James Hegarty et al. “Rigel: flexible multi-rate image processing hardware”. In: *ACM Transactions on Graphics* 35.4 (July 2016), 85:1–85:11. ISSN: 0730-0301. DOI: [10.1145/2897824.2925892](https://doi.org/10.1145/2897824.2925892). URL: <https://doi.org/10.1145/2897824.2925892> (visited on 04/15/2024).
- [66] R. Ho, K.W. Mai, and M.A. Horowitz. “The future of wires”. In: *Proceedings of the IEEE* 89.4 (Apr. 2001). Conference Name: Proceedings of the IEEE, pp. 490–504. ISSN: 1558-2256. DOI: [10.1109/5.920580](https://doi.org/10.1109/5.920580). URL: <https://ieeexplore.ieee.org/abstract/document/920580> (visited on 04/02/2024).
- [67] Andrew Howard et al. “Searching for MobileNetV3”. English. In: IEEE Computer Society, Oct. 2019, pp. 1314–1324. ISBN: 978-1-72814-803-8. DOI: [10.1109/ICCV.2019.00140](https://doi.org/10.1109/ICCV.2019.00140). URL: <https://www.computer.org/csdl/proceedings-article/iccv/2019/480300b314/1hV1GG4j720> (visited on 04/12/2023).

- [68] Yu-Shun Hsiao et al. “Zhuyi: perception processing rate estimation for safety in autonomous vehicles”. In: *Proceedings of the 59th ACM/IEEE Design Automation Conference*. DAC '22. New York, NY, USA: Association for Computing Machinery, Aug. 2022, pp. 289–294. ISBN: 978-1-4503-9142-9. DOI: [10.1145/3489517.3530445](https://doi.org/10.1145/3489517.3530445). URL: <https://dl.acm.org/doi/10.1145/3489517.3530445> (visited on 04/01/2024).
- [69] Loc N. Huynh, Youngki Lee, and Rajesh Krishna Balan. “DeepMon: Mobile GPU-based Deep Learning Framework for Continuous Vision Applications”. In: *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*. MobiSys '17. New York, NY, USA: Association for Computing Machinery, June 2017, pp. 82–95. ISBN: 978-1-4503-4928-4. DOI: [10.1145/3081333.3081360](https://doi.org/10.1145/3081333.3081360). URL: <https://dl.acm.org/doi/10.1145/3081333.3081360> (visited on 04/11/2023).
- [70] *IMGIC Technology*. en-GB. URL: <https://www.imaginationtech.com/resources/imgic-technology/> (visited on 07/08/2023).
- [71] Intel. *Instruction Set Extensions and Future Features Programming Reference*. 2021.
- [72] *Intel® Integrated Performance Primitives*. en. URL: <https://www.intel.com/content/www/us/en/developer/tools/oneapi/ipp.html> (visited on 08/10/2023).
- [73] *Intel® Movidius™ Vision Processing Units (VPUs)*. en. URL: <https://www.intel.com/content/www/us/en/products/details/processors/movidius-vpu.html> (visited on 04/15/2024).
- [74] *Interconnect-power dissipation in a microprocessor | Proceedings of the 2004 international workshop on System level interconnect prediction*. URL: <https://dl.acm.org/doi/abs/10.1145/966747.966750> (visited on 04/02/2024).
- [75] Idaku Ishii et al. “2000 fps real-time vision system with high-frame-rate video recording”. In: *2010 IEEE International Conference on Robotics and Automation*. ISSN: 1050-4729. May 2010, pp. 1536–1541. DOI: [10.1109/ROBOT.2010.5509731](https://doi.org/10.1109/ROBOT.2010.5509731).
- [76] Shubham Jain et al. “Compensated-DNN: energy efficient low-precision deep neural networks by compensating quantization errors”. In: *Proceedings of the 55th Annual Design Automation Conference*. DAC '18. New York, NY, USA: Association for Computing Machinery, June 2018, pp. 1–6. ISBN: 978-1-4503-5700-5. DOI: [10.1145/3195970.3196012](https://doi.org/10.1145/3195970.3196012). URL: <https://dl.acm.org/doi/10.1145/3195970.3196012> (visited on 04/12/2023).

- [77] M. Jakubowski and G. Pastuszak. “Block-based motion estimation algorithms — a survey”. en. In: *Opto-Electronics Review* 21.1 (Mar. 2013), pp. 86–102. ISSN: 1896-3757. DOI: [10.2478/s11772-013-0071-0](https://doi.org/10.2478/s11772-013-0071-0). URL: <https://doi.org/10.2478/s11772-013-0071-0> (visited on 11/22/2023).
- [78] Suren Jayasuriya et al. “Software-Defined Imaging: A Survey”. In: *Proceedings of the IEEE* 111.5 (May 2023). Conference Name: Proceedings of the IEEE, pp. 445–464. ISSN: 1558-2256. DOI: [10.1109/JPROC.2023.3266736](https://ieeexplore.ieee.org/abstract/document/10109744). URL: <https://ieeexplore.ieee.org/abstract/document/10109744> (visited on 03/26/2024).
- [79] Geonhwa Jeong et al. “RASA: Efficient Register-Aware Systolic Array Matrix Engine for CPU”. In: *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 253–258.
- [80] Hui Ji et al. “Robust video denoising using low rank matrix completion”. In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. ISSN: 1063-6919. June 2010, pp. 1791–1798. DOI: [10.1109/CVPR.2010.5539849](https://ieeexplore.ieee.org/document/5539849). URL: <https://ieeexplore.ieee.org/document/5539849> (visited on 11/22/2023).
- [81] Xuan Jing and Lap-Pui Chau. “An efficient three-step search algorithm for block motion estimation”. In: *IEEE Transactions on Multimedia* 6.3 (June 2004). Conference Name: IEEE Transactions on Multimedia, pp. 435–438. ISSN: 1941-0077. DOI: [10.1109/TMM.2004.827517](https://ieeexplore.ieee.org/abstract/document/1298816). URL: <https://ieeexplore.ieee.org/abstract/document/1298816> (visited on 11/22/2023).
- [82] Norman P. Jouppi et al. “In-Datcenter Performance Analysis of a Tensor Processing Unit”. In: *Proceedings of the 44th Annual International Symposium on Computer Architecture*. ISCA '17. New York, NY, USA: Association for Computing Machinery, June 2017, pp. 1–12. ISBN: 978-1-4503-4892-8. DOI: [10.1145/3079856.3080246](https://dl.acm.org/doi/10.1145/3079856.3080246). URL: <https://dl.acm.org/doi/10.1145/3079856.3080246> (visited on 04/12/2023).
- [83] Iman Abaspor Kazerouni et al. “A Survey of State-of-the-Art on Visual SLAM”. In: *Expert Systems with Applications* (2022). Publisher: Elsevier, p. 117734.
- [84] Venkatesh Kodukula et al. “Rhythmic pixel regions: multi-resolution visual sensing system towards high-precision visual computing at low power”. In: *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '21. New York, NY, USA: Association for Computing Machinery, Apr. 2021, pp. 573–586. ISBN: 978-1-4503-8317-2. DOI:

- [10.1145/3445814.3446737](https://doi.org/10.1145/3445814.3446737). URL: <https://dl.acm.org/doi/10.1145/3445814.3446737> (visited on 03/26/2024).
- [85] Srivatsan Krishnan et al. “Automatic Domain-Specific SoC Design for Autonomous Unmanned Aerial Vehicles”. In: *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Oct. 2022, pp. 300–317. DOI: [10.1109/MICRO56248.2022.00033](https://doi.org/10.1109/MICRO56248.2022.00033).
- [86] *Kryo*. en. Page Version ID: 1168941353. Aug. 2023. URL: <https://en.wikipedia.org/w/index.php?title=Kryo&oldid=1168941353> (visited on 08/10/2023).
- [87] Siew-Kei Lam et al. “Area-Time Efficient Streaming Architecture for FAST and BRIEF Detector”. In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 66.2 (2018). Publisher: IEEE, pp. 282–286.
- [88] Steven M. LaValle et al. “Head tracking for the Oculus Rift”. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. ISSN: 1050-4729. May 2014, pp. 187–194. DOI: [10.1109/ICRA.2014.6906608](https://doi.org/10.1109/ICRA.2014.6906608). URL: <https://ieeexplore.ieee.org/document/6906608> (visited on 04/01/2024).
- [89] Shang Li et al. “DRAMsim3: A Cycle-Accurate, Thermal-Capable DRAM Simulator”. In: *IEEE Computer Architecture Letters* 19.2 (July 2020). Conference Name: IEEE Computer Architecture Letters, pp. 106–109. ISSN: 1556-6064. DOI: [10.1109/LCA.2020.2973991](https://doi.org/10.1109/LCA.2020.2973991).
- [90] Sheng Li et al. “McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures”. In: *Proceedings of the 42nd annual ieee/acm international symposium on microarchitecture*. 2009, pp. 469–480.
- [91] Robert LiKamWa and Lin Zhong. “Starfish: Efficient Concurrency Support for Computer Vision Applications”. In: *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*. MobiSys ’15. New York, NY, USA: Association for Computing Machinery, May 2015, pp. 213–226. ISBN: 978-1-4503-3494-5. DOI: [10.1145/2742647.2742663](https://doi.org/10.1145/2742647.2742663). URL: <https://dl.acm.org/doi/10.1145/2742647.2742663> (visited on 04/12/2023).
- [92] Kyusam Lim et al. “A multi-lane MIPI CSI receiver for mobile camera applications”. In: *IEEE Transactions on Consumer Electronics* 56.3 (Aug. 2010). Conference Name: IEEE Transactions on Consumer Electronics, pp. 1185–1190. ISSN: 1558-4127. DOI: [10.1109/TCE.2010.5606244](https://doi.org/10.1109/TCE.2010.5606244).

- [93] Shih-Chieh Lin et al. “The Architectural Implications of Autonomous Driving”. In: *ACM SIGPLAN Notices* 53.2 (Mar. 2018). Publisher: Association for Computing Machinery (ACM), pp. 751–766. DOI: [10.1145/3296957.3173191](https://doi.org/10.1145/3296957.3173191). URL: <https://doi.org/10.1145/3296957.3173191>.
- [94] Liangkai Liu et al. “Computing Systems for Autonomous Driving: State-of-the-Art and Challenges”. In: *IEEE Internet of Things Journal* PP (Dec. 2020), pp. 1–1. DOI: [10.1109/JIOT.2020.3043716](https://doi.org/10.1109/JIOT.2020.3043716).
- [95] Runze Liu et al. “ESLAM: An energy-efficient accelerator for real-time ORB-SLAM on FPGA platform”. In: *Proceedings of the 56th Annual Design Automation Conference 2019*. 2019, pp. 1–6.
- [96] Weizhuang Liu et al. “Archytas: A Framework for Synthesizing and Dynamically Optimizing Accelerators for Robotic Localization”. In: *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO ’21. New York, NY, USA: Association for Computing Machinery, Oct. 2021, pp. 479–493. ISBN: 978-1-4503-8557-2. DOI: [10.1145/3466752.3480077](https://doi.org/10.1145/3466752.3480077). URL: <https://doi.org/10.1145/3466752.3480077> (visited on 02/16/2023).
- [97] Z. Liu et al. “Ultra-low-power image signal processor for smart camera applications”. en. In: *Electronics Letters* 51.22 (2015). _eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1049/el.2015.1924> pp. 1778–1780. ISSN: 1350-911X. DOI: [10.1049/el.2015.1924](https://doi.org/10.1049/el.2015.1924). URL: <https://onlinelibrary.wiley.com/doi/abs/10.1049/el.2015.1924> (visited on 04/15/2024).
- [98] Derek Lockhart, Gary Zibrat, and Christopher Batten. “PyMTL: A Unified Framework for Vertically Integrated Computer Architecture Research”. In: *47th IEEE/ACM Int’l Symp. on Microarchitecture (MICRO)*. Dec. 2014, pp. 280–292. DOI: [10.1109/MICRO.2014.50](https://doi.org/10.1109/MICRO.2014.50).
- [99] D.G. Lowe. “Object recognition from local scale-invariant features”. In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. Vol. 2. Sept. 1999, 1150–1157 vol.2. DOI: [10.1109/ICCV.1999.790410](https://doi.org/10.1109/ICCV.1999.790410).
- [100] Jason Lowe-Power et al. “The gem5 simulator: Version 20.0+”. In: *arXiv preprint arXiv:2007.03152* (2020).
- [101] Arm Ltd. *Arm Frame Buffer Compression – Arm®*. en. URL: <https://www.arm.com/technologies/graphics-technologies/arm-frame-buffer-compression> (visited on 07/08/2023).

- [102] Arm Ltd. *Compute Library – Arm®*. en. URL: <https://www.arm.com/technologies/compute-library> (visited on 08/10/2023).
- [103] Arm Ltd. *Mali-C71AE*. en. URL: <https://www.arm.com/products/silicon-ip-multimedia/image-signal-processor/mali-c71ae> (visited on 11/22/2023).
- [104] Mostafa Mahmoud, Kevin Siu, and Andreas Moshovos. “Diffy: a Déjà vu-Free Differential Deep Neural Network Accelerator”. In: *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Oct. 2018, pp. 134–147. DOI: [10.1109/MICRO.2018.00020](https://doi.org/10.1109/MICRO.2018.00020).
- [105] Hugh T Mair et al. “4.3 A 20nm 2.5GHz ultra-low-power tri-cluster CPU subsystem with adaptive power allocation for optimal mobile SoC performance”. In: *2016 IEEE International Solid-State Circuits Conference (ISSCC)*. ISSN: 2376-8606. Jan. 2016, pp. 76–77. DOI: [10.1109/ISSCC.2016.7417914](https://doi.org/10.1109/ISSCC.2016.7417914).
- [106] *Mali-C32*. URL: <https://developer.arm.com/Processors/Mali-C32> (visited on 04/23/2023).
- [107] Krishna T. Malladi et al. “Towards energy-proportional datacenter memory with mobile DRAM”. In: *ACM SIGARCH Computer Architecture News* 40.3 (June 2012), pp. 37–48. ISSN: 0163-5964. DOI: [10.1145/2366231.2337164](https://doi.org/10.1145/2366231.2337164). URL: <https://doi.org/10.1145/2366231.2337164> (visited on 04/02/2024).
- [108] Giosué Cataldo Marinó et al. “Deep neural networks compression: A comparative survey and choice recommendations”. en. In: *Neurocomputing* 520 (Feb. 2023), pp. 152–170. ISSN: 0925-2312. DOI: [10.1016/j.neucom.2022.11.072](https://doi.org/10.1016/j.neucom.2022.11.072). URL: <https://www.sciencedirect.com/science/article/pii/S0925231222014643> (visited on 04/12/2023).
- [109] J. D. Meindl et al. “Interconnect opportunities for gigascale integration”. In: *IBM Journal of Research and Development* 46.2.3 (Mar. 2002). Conference Name: IBM Journal of Research and Development, pp. 245–263. ISSN: 0018-8646. DOI: [10.1147/rd.462.0245](https://doi.org/10.1147/rd.462.0245). URL: <https://ieeexplore.ieee.org/abstract/document/5388992> (visited on 04/02/2024).
- [110] *Meta Quest 3: New mixed reality VR headset – Shop now | Meta Store*. en. URL: <https://www.meta.com/us/en/quest/quest-3/> (visited on 03/31/2024).
- [111] *Mobileye Solutions | From Driver Assistance to Self-Driving*. en. URL: <https://www.mobileye.com/solutions/> (visited on 03/31/2024).

- [112] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. “ORB-SLAM: a versatile and accurate monocular SLAM system”. In: *IEEE transactions on robotics* 31.5 (2015). Publisher: IEEE, pp. 1147–1163.
- [113] Raul Mur-Artal and Juan D. Tardos. “ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras”. In: *IEEE Transactions on Robotics* 33.5 (Oct. 2017). Publisher: Institute of Electrical and Electronics Engineers (IEEE), pp. 1255–1262. DOI: [10.1109/tro.2017.2705103](https://doi.org/10.1109/tro.2017.2705103). URL: <https://doi.org/10.1109/5C%2Ftro.2017.2705103>.
- [114] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P Jouppi. “CACTI 6.0: A tool to model large caches”. In: *HP laboratories* 27 (2009), p. 28.
- [115] Kazuko Nishimura et al. “An 8K4K-resolution 60fps 450ke-saturation-signal organic-photoconductive-film global-shutter CMOS image sensor with in-pixel noise canceller”. In: *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*. ISSN: 2376-8606. Feb. 2018, pp. 82–84. DOI: [10.1109/ISSCC.2018.8310194](https://doi.org/10.1109/ISSCC.2018.8310194).
- [116] Gabriel Nützi et al. “Fusion of IMU and vision for absolute scale estimation in monocular SLAM”. In: *Journal of intelligent & robotic systems* 61.1-4 (2011). Publisher: Springer, pp. 287–299.
- [117] *NVIDIA’s xavier system-on-chip, HotChips 30*. 2018. URL: https://old.hotchips.org/hc30/1conf/1.12_Nvidia_XavierHotchips2018Final_814.pdf.
- [118] Dhinakaran Pandiyan and Carole-Jean Wu. “Quantifying the energy cost of data movement for emerging smart phone workloads on mobile platforms”. In: *2014 IEEE International Symposium on Workload Characterization (IISWC)*. Oct. 2014, pp. 171–180. DOI: [10.1109/IISWC.2014.6983056](https://doi.org/10.1109/IISWC.2014.6983056). URL: <https://ieeexplore.ieee.org/abstract/document/6983056> (visited on 04/02/2024).
- [119] Angshuman Parashar et al. “SCNN: An Accelerator for Compressed-sparse Convolutional Neural Networks”. In: *ACM SIGARCH Computer Architecture News* 45.2 (June 2017), pp. 27–40. ISSN: 0163-5964. DOI: [10.1145/3140659.3080254](https://doi.org/10.1145/3140659.3080254). URL: <https://dl.acm.org/doi/10.1145/3140659.3080254> (visited on 04/12/2023).
- [120] Jun-Seok Park, Hyo-Eun Kim, and Lee-Sup Kim. “A 182 mW 94.3 f/s in Full HD Pattern-Matching Based Image Recognition Accelerator for an Embedded Vision System in 0.13-um CMOS Technology”. In: *IEEE transactions on circuits and systems for video technology* 23.5 (2012). Publisher: IEEE, pp. 832–845.

- [121] Think Hung Pham, Phong Tran, and Siew-Kei Lam. “High-Throughput and Area-Optimized Architecture for rBRIEF Feature Extraction”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27.4 (2018). Publisher: IEEE, pp. 747–756.
- [122] Thammathip Piumsomboon et al. “CoVAR: a collaborative virtual and augmented reality system for remote collaboration”. In: *SIGGRAPH Asia 2017 Emerging Technologies*. SA ’17. New York, NY, USA: Association for Computing Machinery, Nov. 2017, pp. 1–2. ISBN: 978-1-4503-5404-2. DOI: [10.1145/3132818.3132822](https://doi.org/10.1145/3132818.3132822). URL: <https://doi.org/10.1145/3132818.3132822> (visited on 03/31/2024).
- [123] Wajahat Qadeer et al. “Convolution engine: balancing efficiency & flexibility in specialized computing”. In: *Proceedings of the 40th Annual International Symposium on Computer Architecture*. ISCA ’13. New York, NY, USA: Association for Computing Machinery, June 2013, pp. 24–35. ISBN: 978-1-4503-2079-5. DOI: [10.1145/2485922.2485925](https://doi.org/10.1145/2485922.2485925). URL: <https://doi.org/10.1145/2485922.2485925> (visited on 02/14/2023).
- [124] Tong Qin, Peiliang Li, and Shaojie Shen. “VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator”. In: *IEEE Transactions on Robotics* 34.4 (Aug. 2018). Conference Name: IEEE Transactions on Robotics, pp. 1004–1020. ISSN: 1941-0468. DOI: [10.1109/TR0.2018.2853729](https://doi.org/10.1109/TR0.2018.2853729).
- [125] *Qualcomm pioneers new active depth sensing module | Qualcomm*. en. URL: <https://www.qualcomm.com/news/onq/2017/08/qualcomm-pioneers-new-active-depth-sensing-module> (visited on 11/22/2023).
- [126] Jonathan Ragan-Kelley et al. “Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines”. In: *ACM SIGPLAN Notices* 48.6 (June 2013), pp. 519–530. ISSN: 0362-1340. DOI: [10.1145/2499370.2462176](https://doi.org/10.1145/2499370.2462176). URL: <https://dl.acm.org/doi/10.1145/2499370.2462176> (visited on 07/07/2023).
- [127] Mohammad Rastegari et al. “XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks”. en. In: *Computer Vision – ECCV 2016*. Ed. by Bastian Leibe et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, pp. 525–542. ISBN: 978-3-319-46493-0. DOI: [10.1007/978-3-319-46493-0_32](https://doi.org/10.1007/978-3-319-46493-0_32).

- [128] Jason Redgrave et al. “Pixel visual core: Google’s fully programmable image vision and AI processor for mobile devices”. In: *Proc. IEEE hot chips Symp.(HCS)*. 2018, pp. 1–18.
- [129] Marc Riera, Jose-Maria Arnau, and Antonio Gonzalez. “Computation Reuse in DNNs by Exploiting Input Similarity”. In: *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. ISSN: 2575-713X. June 2018, pp. 57–68. DOI: [10.1109/ISCA.2018.00016](https://doi.org/10.1109/ISCA.2018.00016).
- [130] RISC-V. *Working draft of the proposed risc-v v vector extension*. URL: <https://github.com/riscv/riscv-v-spec>.
- [131] Paul L Rosin. “Measuring corner properties”. In: *Computer Vision and Image Understanding* 73.2 (1999). Publisher: Elsevier, pp. 291–307.
- [132] Edward Rosten and Tom Drummond. “Fusing points and lines for high performance tracking.” In: *ICCV*. Vol. 2. Citeseer, 2005, pp. 1508–1515.
- [133] Edward Rosten and Tom Drummond. “Machine learning for high-speed corner detection”. In: *European conference on computer vision*. Springer, 2006, pp. 430–443.
- [134] Ethan Rublee et al. “ORB: An efficient alternative to SIFT or SURF”. In: *2011 International conference on computer vision*. Ieee, 2011, pp. 2564–2571.
- [135] *Self-Driving Car Technology for a Reliable Ride - Waymo Driver*. URL: <https://waymo.com/intl/es/waymo-driver/> (visited on 03/31/2024).
- [136] *Self-Driving Cars Technology & Solutions | NVIDIA Automotive*. en-us. URL: <https://www.nvidia.com/en-us/self-driving-cars/> (visited on 03/31/2024).
- [137] Franyell Silva et al. “E-PUR: an energy-efficient processing unit for recurrent neural networks”. In: *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*. PACT ’18. New York, NY, USA: Association for Computing Machinery, Nov. 2018, pp. 1–12. ISBN: 978-1-4503-5986-3. DOI: [10.1145/3243176.3243184](https://doi.org/10.1145/3243176.3243184). URL: <https://dl.acm.org/doi/10.1145/3243176.3243184> (visited on 04/12/2023).
- [138] Manuel Silva and Luís Teixeira. “eXtended Reality (XR) Experiences in Museums for Cultural Heritage: A Systematic Review”. en. In: *Intelligent Technologies for Interactive Entertainment*. Ed. by Zhihan Lv and Houbing Song. Cham: Springer International Publishing, 2022, pp. 58–79. ISBN: 978-3-030-99188-3. DOI: [10.1007/978-3-030-99188-3_5](https://doi.org/10.1007/978-3-030-99188-3_5).

- [139] *Standalone-Pre/Post Processor*. en. URL: <https://www.chipsnmedia.com/mapi> (visited on 07/08/2023).
- [140] Nigel Stephens et al. “The ARM scalable vector extension”. In: *IEEE micro* 37.2 (2017). Publisher: IEEE, pp. 26–39.
- [141] Aaron Stillmaker and Bevan Baas. “Scaling equations for the accurate prediction of CMOS device performance from 180 nm to 7 nm”. In: *Integration* 58 (June 2017), pp. 74–81. ISSN: 0167-9260. DOI: [10.1016/j.vlsi.2017.02.002](https://doi.org/10.1016/j.vlsi.2017.02.002). URL: <https://www.sciencedirect.com/science/article/pii/S0167926017300755> (visited on 04/15/2024).
- [142] James E Stine et al. “FreePDK: An open-source variation-aware design kit”. In: *2007 IEEE international conference on Microelectronic Systems Education (MSE'07)*. IEEE, 2007, pp. 173–174.
- [143] Rongdi Sun et al. “A 42fps full-hd orb feature extraction accelerator with reduced memory overhead”. In: *2017 International Conference on Field Programmable Technology (ICFPT)*. IEEE, 2017, pp. 183–190.
- [144] Rongdi Sun et al. “A Flexible and Efficient Real-Time ORB-Based Full-HD Image Feature Extraction Accelerator”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (2019). Publisher: IEEE.
- [145] Yuxiang Sun, Ming Liu, and Max Q. -H. Meng. “Improving RGB-D SLAM in dynamic environments: A motion removal approach”. In: *Robotics and Autonomous Systems* 89 (Mar. 2017), pp. 110–122. ISSN: 0921-8890. DOI: [10.1016/j.robot.2016.11.012](https://doi.org/10.1016/j.robot.2016.11.012). URL: <https://www.sciencedirect.com/science/article/pii/S0921889015302232> (visited on 11/22/2023).
- [146] Synopsys. *Synopsys Suite*. 2022. URL: <https://www.synopsys.com/>.
- [147] Jie Tang et al. “ π -SoC: Heterogeneous SoC Architecture for Visual Inertial SLAM Applications”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2018, pp. 8302–8307. DOI: [10.1109/IROS.2018.8594181](https://doi.org/10.1109/IROS.2018.8594181).
- [148] Raúl Taranco, José-Maria Arnau, and Antonio González. “A Low-Power Hardware Accelerator for ORB Feature Extraction in Self-Driving Cars”. In: *2021 IEEE 33rd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. ISSN: 2643-3001. Belo Horizonte, Brazil, Oct. 2021, pp. 11–21.

- DOI: [10.1109/SBAC-PAD53543.2021.00013](https://doi.org/10.1109/SBAC-PAD53543.2021.00013). URL: <https://ieeexplore.ieee.org/document/9651662>.
- [149] Raúl Taranco, José-Maria Arnau, and Antonio González. “LOCATOR: Low-power ORB accelerator for autonomous cars”. In: *Journal of Parallel and Distributed Computing* 174 (Apr. 2023), pp. 32–45. ISSN: 0743-7315. DOI: [10.1016/j.jpdc.2022.12.005](https://doi.org/10.1016/j.jpdc.2022.12.005). URL: <https://www.sciencedirect.com/science/article/pii/S0743731522002507>.
- [150] Raúl Taranco, José-Maria Arnau, and Antonio González. “SLIDEX: A Novel Architecture for Sliding Window Processing”. In: *Proceedings of the 38th ACM International Conference on Supercomputing*. ICS '24. Kyoto, Japan: Association for Computing Machinery, June 2024. ISBN: 979-8-4007-0610-3/24/06. DOI: [10.1145/3650200.3656613](https://doi.org/10.1145/3650200.3656613).
- [151] Raúl Taranco, José-María Arnau, and Antonio González. “ δ LTA: Decoupling Camera Sampling from Processing to Avoid Redundant Computations in the Vision Pipeline”. In: *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO '23. Toronto, Canada: Association for Computing Machinery, Dec. 2023, pp. 1029–1043. ISBN: 9798400703294. DOI: [10.1145/3613424.3614261](https://doi.org/10.1145/3613424.3614261). URL: <https://dl.acm.org/doi/10.1145/3613424.3614261>.
- [152] Raúl Taranco, José-María Arnau, and Antonio González. “SLIDEX: Sliding Window Extension for Image Processing”. In: *2023 32nd International Conference on Parallel Architectures and Compilation Techniques (PACT)*. Vienna, Austria, Oct. 2023, pp. 332–334. DOI: [10.1109/PACT58117.2023.00039](https://doi.org/10.1109/PACT58117.2023.00039). URL: <https://ieeexplore.ieee.org/document/10364589?signout=success>.
- [153] Bill Triggs et al. “Bundle adjustment — A modern synthesis”. In: *International workshop on vision algorithms*. Springer, 1999, pp. 298–372.
- [154] Nisarg Ujjainkar, Jingwen Leng, and Yuhao Zhu. “ImaGen: A General Framework for Generating Memory- and Power-Efficient Image Processing Accelerators”. In: *Proceedings of the 50th Annual International Symposium on Computer Architecture*. ISCA '23. New York, NY, USA: Association for Computing Machinery, June 2023, pp. 1–13. ISBN: 9798400700958. DOI: [10.1145/3579371.3589076](https://doi.org/10.1145/3579371.3589076). URL: <https://doi.org/10.1145/3579371.3589076> (visited on 04/02/2024).
- [155] Artem Vasilyev et al. “Evaluating programmable architectures for imaging and vision applications”. In: *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–13.

- [156] Vibhakar Vemulapati and Deming Chen. “FSLAM: an Efficient and Accurate SLAM Accelerator on SoC FPGAs”. In: *2022 International Conference on Field-Programmable Technology (ICFPT)*. Dec. 2022, pp. 1–9. DOI: [10.1109/ICFPT56656.2022.9974562](https://doi.org/10.1109/ICFPT56656.2022.9974562). URL: <https://ieeexplore.ieee.org/document/9974562> (visited on 03/26/2024).
- [157] Thomas Vogelsang. “Understanding the Energy Consumption of Dynamic Random Access Memories”. In: *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*. ISSN: 2379-3155. Dec. 2010, pp. 363–374. DOI: [10.1109/MICRO.2010.42](https://doi.org/10.1109/MICRO.2010.42). URL: <https://ieeexplore.ieee.org/abstract/document/5695550> (visited on 04/02/2024).
- [158] Yevgen Voronenko and Markus Püschel. “Multiplierless multiple constant multiplication”. In: *ACM Transactions on Algorithms (TALG)* 3.2 (2007). Publisher: ACM New York, NY, USA, 11–es.
- [159] Sen Wang et al. “DeepVO: Towards end-to-end visual odometry with deep Recurrent Convolutional Neural Networks”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. May 2017, pp. 2043–2050. DOI: [10.1109/ICRA.2017.7989236](https://doi.org/10.1109/ICRA.2017.7989236). URL: <https://ieeexplore.ieee.org/abstract/document/7989236> (visited on 04/02/2024).
- [160] Josh Weberruss et al. “FPGA acceleration of multilevel ORB feature extraction for computer vision”. In: *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2017, pp. 1–8.
- [161] *What is an NPU? And why is it key to unlocking on-device generative AI?* en. URL: <https://www.qualcomm.com/news/onq/2024/02/what-is-an-npu-and-why-is-it-key-to-unlocking-on-device-generative-ai> (visited on 04/15/2024).
- [162] *What Your DRAM Power Models Are Not Telling You: Lessons from a Detailed Experimental Study: Proceedings of the ACM on Measurement and Analysis of Computing Systems: Vol 2, No 3*. URL: <https://dl.acm.org/doi/abs/10.1145/3224419> (visited on 04/02/2024).
- [163] Clifford Wolf. *Yosys Open SYnthesis Suite*. URL: <http://www.clifford.at/yosys/>.
- [164] Junfei Xiao et al. “Learning From Temporal Gradient for Semi-Supervised Action Recognition”. en. In: 2022, pp. 3252–3262. URL: https://openaccess.thecvf.com/content/CVPR2022/html/Xiao_Learning_From_Temporal_Gradient_for_Semi-Supervised_Action_Recognition_CVPR_2022_paper.html (visited on 04/24/2023).

- [165] Mengwei Xu et al. “DeepCache: Principled Cache for Mobile Deep Vision”. In: *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*. MobiCom '18. New York, NY, USA: Association for Computing Machinery, Oct. 2018, pp. 129–144. ISBN: 978-1-4503-5903-0. DOI: [10.1145/3241539.3241563](https://doi.org/10.1145/3241539.3241563). URL: <https://dl.acm.org/doi/10.1145/3241539.3241563> (visited on 04/11/2023).
- [166] Takayuki Yamashita and Kohji Mitani. “8K Extremely-High-Resolution Camera Systems”. In: *Proceedings of the IEEE* 101.1 (Jan. 2013). Conference Name: Proceedings of the IEEE, pp. 74–88. ISSN: 1558-2256. DOI: [10.1109/JPROC.2012.2217371](https://doi.org/10.1109/JPROC.2012.2217371).
- [167] Ziyu Ying et al. “Exploiting Frame Similarity for Efficient Inference on Edge Devices”. In: *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*. ISSN: 2575-8411. July 2022, pp. 1073–1084. DOI: [10.1109/ICDCS54860.2022.00107](https://doi.org/10.1109/ICDCS54860.2022.00107).
- [168] Khalid Yousif, Alireza Bab-Hadiashar, and Reza Hoseinnezhad. “An Overview to Visual Odometry and Visual SLAM: Applications to Mobile Robotics”. en. In: *Intelligent Industrial Systems* 1.4 (Dec. 2015), pp. 289–311. ISSN: 2199-854X. DOI: [10.1007/s40903-015-0032-7](https://doi.org/10.1007/s40903-015-0032-7). URL: <https://doi.org/10.1007/s40903-015-0032-7> (visited on 04/12/2023).
- [169] Bo Yu et al. “Building the Computing System for Autonomous Micromobility Vehicles: Design Constraints and Architectural Optimizations”. In: *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Oct. 2020, pp. 1067–1081. DOI: [10.1109/MICRO50266.2020.00089](https://doi.org/10.1109/MICRO50266.2020.00089).
- [170] Ekim Yurtsever et al. “A Survey of Autonomous Driving: Common Practices and Emerging Technologies”. In: *IEEE Access* 8 (2020). Conference Name: IEEE Access, pp. 58443–58469. ISSN: 2169-3536. DOI: [10.1109/ACCESS.2020.2983149](https://doi.org/10.1109/ACCESS.2020.2983149).
- [171] Zhe Zhang et al. “PIRVIS: An Advanced Visual-Inertial SLAM System with Flexible Sensor Fusion and Hardware Co-Design”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. ISSN: 2577-087X. May 2018, pp. 3826–3832. DOI: [10.1109/ICRA.2018.8460672](https://doi.org/10.1109/ICRA.2018.8460672).
- [172] Fangwei Zhong et al. “Detect-SLAM: Making Object Detection and SLAM Mutually Beneficial”. In: *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. Mar. 2018, pp. 1001–1010. DOI: [10.1109/WACV.2018.00115](https://doi.org/10.1109/WACV.2018.00115). URL: <https://ieeexplore.ieee.org/abstract/document/8354219> (visited on 11/22/2023).

- [173] Wenping Zhu et al. “A 135-frames/s 1080p 87.5-mW binary-descriptor-based image feature extraction accelerator”. In: *IEEE Transactions on Circuits and Systems for Video Technology* 26.8 (2015). Publisher: IEEE, pp. 1532–1543.
- [174] Yuhao Zhu et al. “Euphrates: algorithm-SoC co-design for low-power mobile continuous vision”. In: *Proceedings of the 45th Annual International Symposium on Computer Architecture*. ISCA '18. Los Angeles, California: IEEE Press, June 2018, pp. 547–560. ISBN: 978-1-5386-5984-7. DOI: [10.1109/ISCA.2018.00052](https://doi.org/10.1109/ISCA.2018.00052). URL: <https://doi.org/10.1109/ISCA.2018.00052> (visited on 02/14/2023).