Journal of Parallel and Distributed Computing 174 (2023) 32–45

Contents lists available at ScienceDirect

Journal of Parallel and Distributed Computing

journal homepage: www.elsevier.com/locate/jpdc

LOCATOR: Low-power ORB accelerator for autonomous cars

Raúl Taranco *, José-Maria Arnau, Antonio González

Computer Architecture Department, Universitat Politècnica de Catalunya, Barcelona, Spain

ARTICLE INFO

Article history:
Received 21 April 2022
Received in revised form 10 November 2022
Accepted 3 December 2022
Available online 9 December 2022

Keywords:
ORB
ORB-SLAM
Hardware accelerator

ABSTRACT

Simultaneous Localization And Mapping (SLAM) is crucial for autonomous navigation. ORB-SLAM is a state-of-the-art Visual SLAM system based on cameras used for self-driving cars. In this paper, we propose a high-performance, energy-efficient, and functionally accurate hardware accelerator for ORB-SLAM, focusing on its most time-consuming stage: Oriented FAST and Rotated BRIEF (ORB) feature extraction. The Rotated BRIEF (rBRIEF) descriptor generation is the main bottleneck in ORB computation, as it exhibits highly irregular access patterns to local on-chip memories causing a high-performance penalty due to bank conflicts. We introduce a technique to find an optimal static pattern to perform parallel accesses to banks based on a genetic algorithm. Furthermore, we propose the combination of an rBRIEF pixel duplication cache, selective ports replication, and pipelining to reduce latency without compromising cost. The accelerator achieves a reduction in energy consumption of 14597× and 9609×, with respect to high-end CPU and GPU platforms, respectively.

1. Introduction

Simultaneous Localization and Mapping (SLAM) [7][11] is a crucial component in autonomous navigation systems and has attracted much interest from both academia and industry in recent years. SLAM is a fundamental task for higher-level activities such as path planning or navigation, and it is widely used in applications such as self-driving cars [14]. SLAM techniques build a map of an unknown environment and localize the exploring agent in that map using the onboard sensors. Vision sensors are the most popular because cameras are inexpensive and compact, providing vast information about the environment. Other alternatives employ LIDAR [17] (laser) technology. The systems based on LIDAR have great precision, and there are efforts to accelerate the critical task to make it effective [28]. Other proposals employ hybrid systems that simultaneously fuse the information from multiple sensors such as cameras, LIDAR, or Inertial Measurement Units (IMUs) to improve the SLAM accuracy [3,4,37].

Among Visual SLAM solutions, feature-based ones have received particular attention because of their robustness to large motions and illumination changes compared to other approaches [8]. However, these methods present significant challenges mainly due to real-time inherent constraints and energy consumption budget available on potential targets [14].

This work focuses on a state-of-the-art [8,12] SLAM solution that is one of the most popular modern methods due to its robustness: ORB-SLAM [18]. This system combines Features from Accelerated Segment Test (FAST) [25] and Binary Robust Independent Elementary Features (BRIEF) [2]. The former identifies the features in an image, whereas the latter generates a robust descriptor for each feature. Together with an orientation estimator, these two parts give rise to Oriented FAST and Rotated BRIEF (ORB) [26]. These features identify corners on the processed images and apply a rotation to generate their descriptors to provide viewpoint and rotational-invariant properties. According to our experiments, ORB-SLAM spends more than 60% of the execution time (see Fig. 1) extracting features (ORB Extraction).

In this paper, we propose a heterogeneous architecture for ORB-SLAM that combines *LOCATOR*: a **L**ow-power **O**RB a**C**celerator for **A**u**T**onom**O**us ca**R**s for feature extraction and a mobile CPU for the remaining tasks such as tracking, local mapping, and loop closing [18]. Due to the irregular memory access patterns, computing the Rotated BRIEF (rBRIEF) descriptor is the most challenging part. Once the system classifies a pixel as a corner (i.e., a feature in the image), it computes a 256-bit descriptor. The computation consists of 256 comparisons between pairs of pixels in the corner neighborhood. The locations of the 256 pairs of pixels do not follow any regular pattern and change dynamically due to the rotation angle. Previous ORB accelerators modify the rBRIEF algorithm to obtain a more hardware-friendly version but at the cost of significant accuracy loss [15,30]. We believe that it is not an acceptable trade-off

* Corresponding author.
E-mail addresses: taranco@ac.upc.edu (R. Taranco), jose.maria.arnau@upc.edu (J.-M. Arnau), antonio@ac.upc.edu (A. González).

https://doi.org/10.1016/j.jpdc.2022.12.005

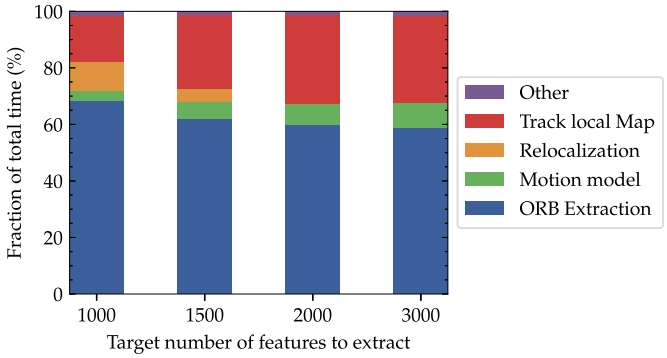

Fig. 1. Average relative execution time of each part of ORB-SLAM running on a CPU. ORB feature extraction takes the majority of execution time.

in the context of self-driving cars, and hence, our approach obtains an accuracy comparable to the software-based solutions.

Our solution 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 with 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 allows overlapping the conflict resolution with subsequent pixel accesses.

In this paper, we make the following contributions:

- We analyze the performance and energy consumption of ORB-SLAM on a state-of-the-art CPU. Our results show that feature extraction is the primary performance and energy bottleneck.
- We propose a high-performance, energy-efficient, and functionally accurate hardware accelerator called LOCATOR for ORB feature extraction, which is the main bottleneck of ORB-SLAM.
- We present a technique to generate a new rBRIEF static scheduling of the required operations that minimizes the number of conflicts when accessing the on-chip memory structures required for the descriptor computation. The genetic algorithm technique obtains a 1.3x speedup over the first version of the accelerator.
- We propose a combination of techniques to reduce the latency of the rBRIEF unit in the average and worst cases. These techniques consist of the repeated rBRIEF pattern pixel duplication cache and pipelining. The increased cost of the unit in area and power due to these techniques is mitigated by selective port replication, based on the characteristic rBRIEF pattern utilization of the memory structure. In general, these techniques reduce an extra 2.26% of the average feature extraction frame processing and a 10.42% of the worst-case execution time with negligible impact in area and energy.
- The experimental results obtained show that the accelerator achieves a reduction in energy consumption of 14597× and 9609×, with respect to high-end CPU and GPU platforms, respectively.

The remainder of the document is organized as follows: Section 2 provides some background on ORB-SLAM. Section 3 describes the basic LOCATOR design. Section 5 presents the experimental results. Section 6 reviews related work and, finally, Section 7 sums up the main conclusions and future work.

2. Background

ORB-SLAM [20][19] is a localization system that estimates the actual trajectory of an agent equipped with a camera while building a representation of the surroundings by storing it in a map. We are interested in its performance in autonomous driving, where it has been ranked at the top of the available open-source algorithms [9]. All the tasks needed to localize the agent are based solely on manipulating ORB [26] features. Fig. 1 shows the relative execution time broken down in various target ORB features to extract per frame. Since ORB extraction spends up to 60% of the total algorithm processing time, in this work, we focus on accelerating it on an ASIC while the remaining tasks are performed on an embedded co-located processor.

ORB extraction can be divided into four main steps [26]: pyramid building (subsection 2.1); FAST Keypoint Detection (subsection 2.2); orientation estimation (subsection 2.3) and rBRIEF generation (subsection 2.4). The next subsection provides further details on ORB feature extraction.

2.1. Pyramid building

FAST does not produce multi-scale features, but scale invariance is desirable. A scale pyramid of images is used to achieve it. The pyramid consists of several levels, with versions of the original image reduced and smoothed.

2.2. FAST keypoint detection

Features From Accelerated Segment Test (FAST), first introduced in [24], is a corner detector that can extract feature points. FAST performs a test to classify a candidate pixel, p, as a corner/no corner using a 7×7 window centered around it. This test compares the intensity of p with the intensities of the 16 pixels forming a Bresenham circle around the candidate, as shown in Fig. 2a. A corner is detected at the candidate pixel p if the intensities of at least $n = 12$ contiguous pixels out of the 16 are all above or below the intensity of p by a threshold, t.

Finally, Non-Maximal Suppression (NMS) is applied as a post-processing method that removes some corners based on a score. Only the corners with the local maximum score within a neighborhood prevail (typically a fixed-sized square patch centered on the considered pixel).

2.3. oFAST: FAST keypoint orientation

ORB uses the intensity centroid [26] to compute the orientation component of FAST. The intensity centroid assumes that a corner's intensity is offset from its center, and this vector can be used to impute an orientation.

The moment of a patch can be defined as [23]:

$$m_{pq} = \sum_{x,y} x^p y^q I(x, y) \tag{1}$$

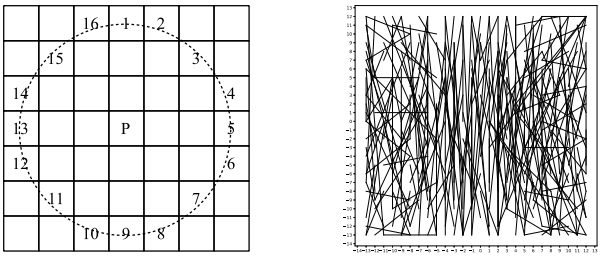

Fig. 2. (a) Bresenham circle of radius 3 showing the pixel access pattern for FAST around the candidate pixel P. (b) ORB pattern positions for a rotation of 0º. A line connects each pair of points required for an intensity test.

where $I(x, y)$ is the pixel's intensity at the relative position x, y within the patch, and p and q are naturals indicating the moment order in each dimension. With this definition, it is possible to compute the orientation centroid:

$$C = (\frac{m_{01}}{m_{00}}, \frac{m_{10}}{m_{00}}) \tag{2}$$

And then, compute the angle of the vector formed between the center point of the corner, O, and the centroid C, $\vec{OC}$.

$$\theta = atan2(m_{01}, m_{10}) \tag{3}$$

$Atan2$ is the quadrant-aware version of the arctangent.

In addition, it is possible to compute the $sin(\theta)$ and $cos(\theta)$ using the moments in the following way:

$$sin(\theta) = \frac{m_{10}}{\sqrt{m_{01}^2 + m_{10}^2}}, \qquad cos(\theta) = \frac{m_{01}}{\sqrt{m_{01}^2 + m_{10}^2}} \tag{4}$$

oFAST is the combination of the segment test that determines if a pixel is a corner and the orientation computation.

2.4. Rotation-aware BRIEF descriptor generation

The Rotation-Aware BRIEF (rBRIEF) descriptor [2] is a bit string description of an image patch constructed from a set of binary intensity tests. A binary test, τ, is defined by:

$$\tau(p_1, p_2) = \begin{cases} 0 & , I(p_1) < I(p_2) \\ 1 & , I(p_1) \geq I(p_2) \end{cases} \tag{5}$$

where p_1 and p_2 are two 2D points and $I(p_i)$ is the intensity of the point p_i. The feature is defined as a vector of n binary tests:

$$f_n(p) = \sum_{1 \leq i \leq n} 2^{i-1} \tau(p_{1_i}, p_{2_i}) \tag{6}$$

It is recommended to smooth the image before performing these tests, for example, with a Gaussian blur filter. The vector length usually is $n = 256$.

One of the most attractive features of ORB is its in-plane rotation invariance. To this goal, the coordinates of the points required for the binary intensity tests of rBRIEF are rotated according to the orientation, θ, employing the corresponding rotation matrix, R_θ. Let us define a matrix of dimension $2 \times n$ that contains the coordinates of the n points (x_i, y_i) of the binary tests:

$$S = \begin{pmatrix} x_1, ..., x_n \\ y_1, ..., y_n \end{pmatrix} \tag{7}$$

Then, the coordinates of the locations of the pairs required to compute the descriptor after rotation are given by:

$$S_\theta = R_\theta S \tag{8}$$

3. Hardware accelerated ORB

In this section, we describe the architecture of LOCATOR and its optimization for ORB feature extraction.

3.1. Hardware architecture overview

Fig. 3 shows the architecture of the proposed accelerator. The overall design employs streaming-based dataflow inspired by previous works in the field [13,35,15,30]. Note that our solution differs significantly from these previous proposals, as explained in Section 6.

The accelerator 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 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 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 Fig. 3) 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 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 Fig. 3), 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.

3.2. 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. Fig. 4 illustrates a sliding window similar to the ones used 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.

3.3. rBRIEF unit

The rBRIEF unit is responsible for implementing the most challenging part of the ORB extraction. It requires providing memory accesses to 512 (see Fig. 2b) positions that depend on the feature

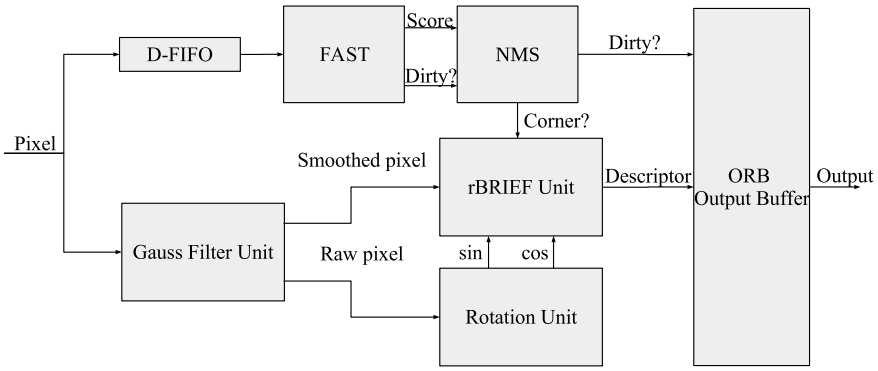

Fig. 3. The basic architecture of LOCATOR.

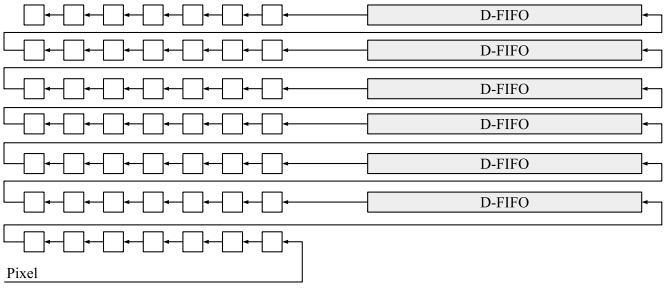

Fig. 4. 7 × 7 sliding window structure.

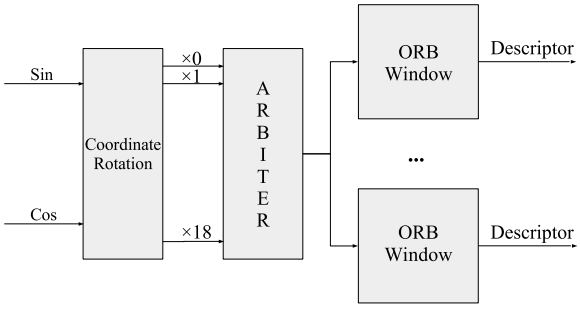

Fig. 5. rBRIEF Unit architecture overview.

angle to compute the rBRIEF descriptor, making it hard to find an optimal access schedule.

Fig. 5 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. (6)). 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. The solution presented in this paper, which includes several optimizations, is 1.67× faster than the naive design while showing equivalent area and power consumption.

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 0º rBRIEF pattern coordinates (stored in a LUT), the sine, the cosine, and efficient implementation of multiplications that utilizes just additions and shifts [32].

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.

3.3.1. 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 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. Fig. 6 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 5).

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.

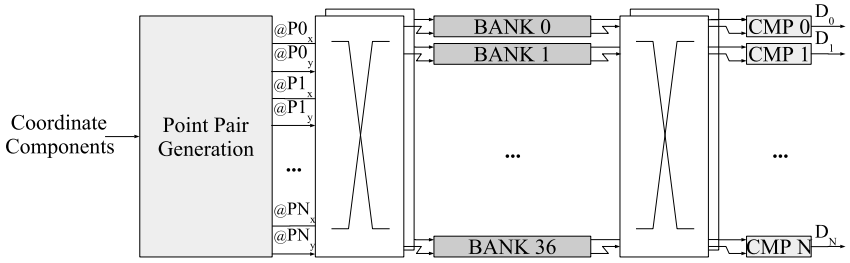

Fig. 6. ORB Window architecture that allows parallel access to multiple pairs (stored in the banks shown in dark grey) per cycle.

3.3.2. 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 [26], 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 $P = \{P_1, P_2, ..., P_N\}$, and a set of groups $G = \{G_1, G_2, ..., G_{\frac{N}{gsize}}\}$, where *gsize* is a fixed parameter that indicates the size of the group of pairs. An assignment is represented by a tuple $< P, G >$. 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 [1], 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 [10].
- Mutation: The mutation is performed each generation, shuffling each chromosome of individuals with a given probability. The mutation swaps pairs between groups.

Table 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

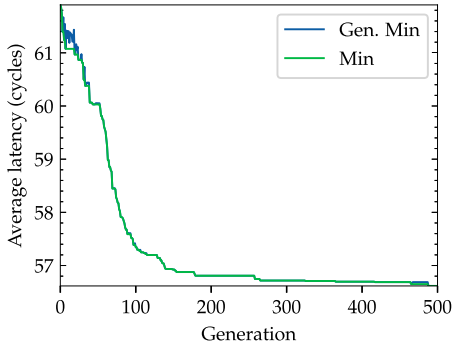

Fig. 7. Example of convergence of the genetic algorithm to a local minimum for a $gsize = 8$.

Table 1 summarizes the parameters used for the GA. Fig. 7 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.

3.3.3. Point intensity duplication cache

An additional approach proposed to reduce conflicts is exploiting the potential temporal locality 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

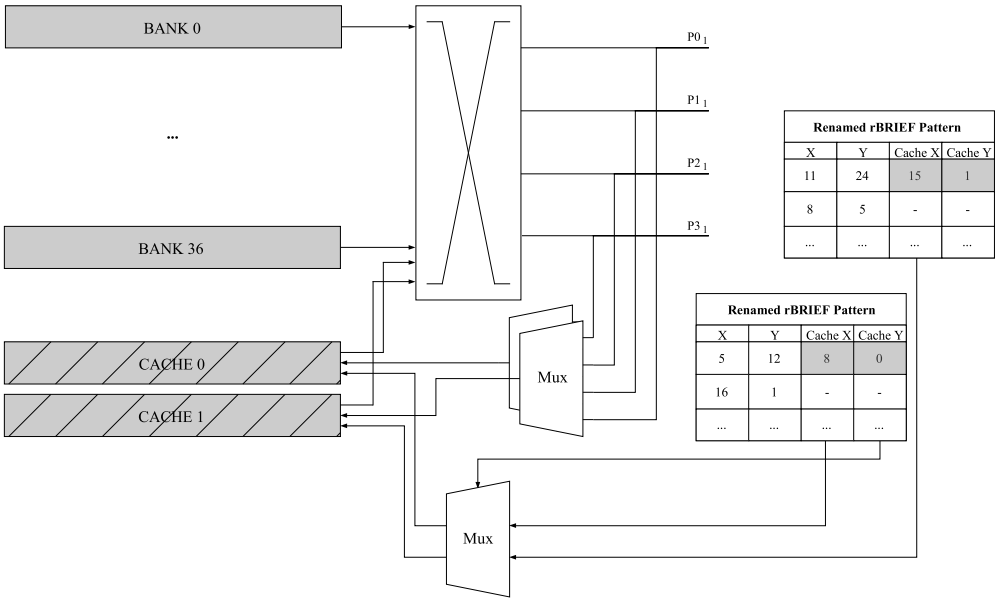

Fig. 8. The basic structure of the duplication cache (new cache banks shown filled with a line pattern) for a group size of 4 pairs. Note that only the first point of each pair PN_1 is represented.

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. Fig. 8 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 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 3.3.2. 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.

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. Fig. 9 illustrates an example of the described renaming scheme.

3.3.4. 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

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
...	...	...	...	...

Fig. 9. 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).

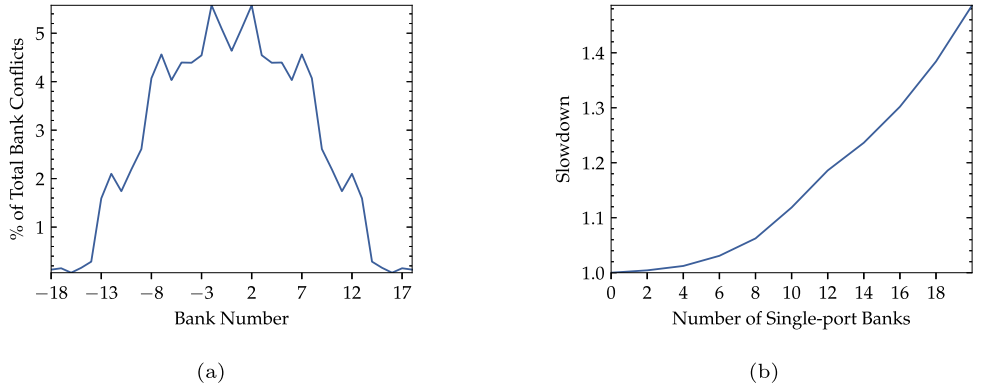

Fig. 10. (a) Percentage of total conflicts (Y-axis) generated for each bank on average for all angles. The X-axis shows the index of the banks in the range [-18, 17]. The bank with an index of zero is the central bank. (b) The slowdown of the average access latency of a replica when increasing the number of external single-ported banks. The parallel access group had a size of 4 pairs in this experiment.

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 Fig. 10a. The rBRIEF pattern, shown in Fig. 2b, 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. Fig. 10b 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 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 5 shows the resulting performance improvement and energy efficiency balance obtained by combining the duplication cache and the technique presented in this subsection.

3.3.5. Pipelining

When a bank access conflict occurs in the baseline, the conflicting bank accesses are serialized while all other accesses for the

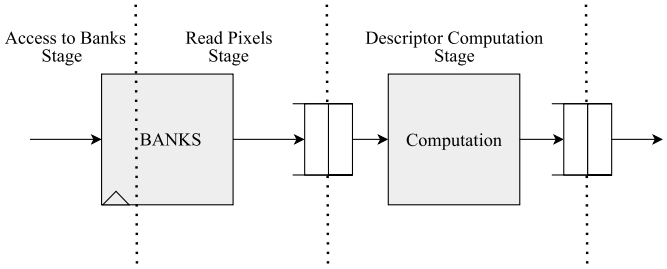

Fig. 11. The proposed stages and pipeline mechanism of each replica of the rBRIEF. The FIFOs between the stages have a length of 2.

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.

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. Fig. 11 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. Fig. 12a 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).

Finally, it is worth noting some details about integrating the pipelining mechanism with the duplication cache optimization discussed in the subsection 3.3.3. 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.

3.4. FAST detector unit

Fig. 13 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), 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.

To perform the segment test and score computation, we use a similar solution as other works [29]. 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.

3.5. 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.

3.6. 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).

3.7. 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 (8). The precision of the calculations is key since errors in the rotation of the points

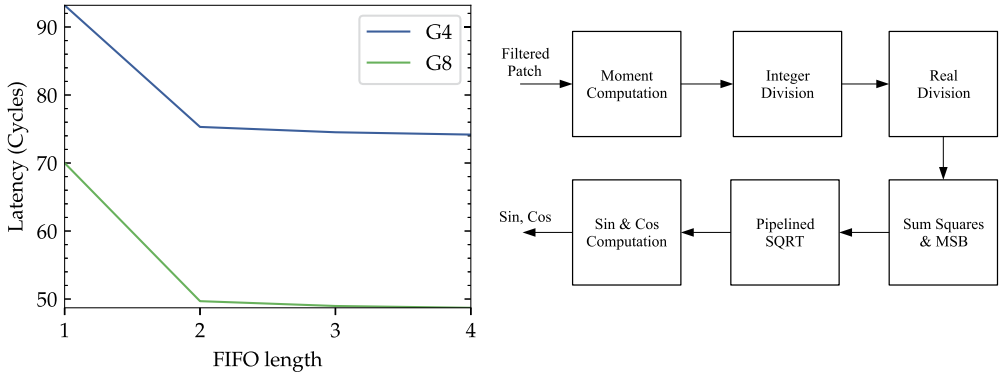

Fig. 12. (a) Sensitivity to the length of the FIFOs in the replica pipelining stages of the rBRIEF unit for a group size of 4 (G4) and 8 (G8) pairs processed in parallel. (b) The pipelined architecture of the Rotation Unit.

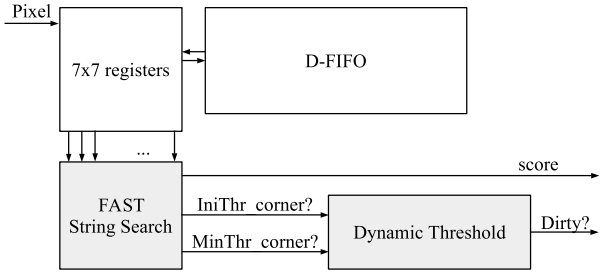

Fig. 13. Architecture of the FAST unit.

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 [35,22,30]. 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 Fig. 12b 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 (1) can be reformulated as follows:

$$\begin{aligned} m_{01_{n+1}} &= m_{01_n} + 18 \times (C_n + C_{n-37}) - S_n, \\ m_{10_{n+1}} &= m_{10_n} + xC_n - xC_{n-37}, \\ m_{00_{n+1}} &= m_{00_n} + C_n - C_{n-37}, \end{aligned} \tag{9}$$

where,

$$\begin{aligned} S_{n+1} &= S_n + C_n - C_{n-36}, \\ C_n &= \sum_{1 \le x \le 37} p_{x,n}, \\ xC_n &= \sum_{1 \le x \le 18} x \times (p_{38-x,n} - p_{x,n}), \end{aligned} \tag{10}$$

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 [32] (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:

$$log_2(\frac{1}{\sqrt{x}}) = -\frac{1}{2} log_2(x) \tag{11}$$

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:

$$y_{n+1} = y_n(\frac{3}{2} - \frac{x}{2} {y_n}^2) \tag{12}$$

The final stage computes the sine and cosine, applying Equation (4) and updating the signs of the final values. The fixed-point representation of the trigonometric values uses 16 bits for the fractional part.

4. Evaluation methodology

We have developed Register-Transfer-Level (RTL) models of the ORB accelerator described in Section 3 by leveraging the PyMTL [16] framework. We tested different versions of the architecture, varying the number of rBRIEF window replicas and comparing the results with the improved architecture and the modifications detailed in the previous section. Table 2 shows the parameters employed in the experiments. In order to estimate area and critical path delay, we translate the PyMTL models into Verilog and synthesize them using Yosys [36] with the open-source 45 nm FreePDK45 1.4 [27]. Moreover, we obtain the power dissipation of the gate-level netlist of the accelerator using Synopsys Design Compiler [31].

As the software baseline, we use an open-source ORB-SLAM implementation [20]. We measure the performance of this implementation on a CPU and a GPU platform with parameters shown in Table 2. We use Intel RAPL [33] and the Nvidia System Management Interface (nvidia-smi) to measure CPU and GPU energy consumption, respectively. The GPU evaluation employs the OpenCV

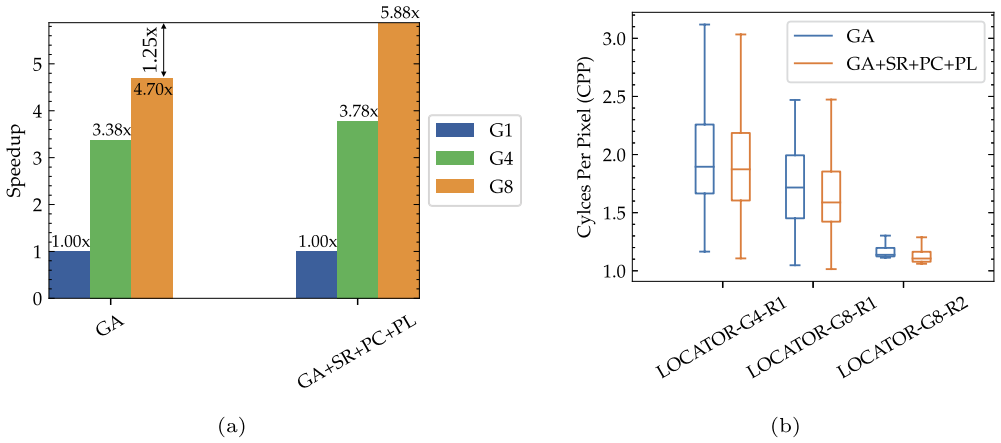

Fig. 14. (a) rBRIEF replica speedup respect to the baseline implementation (*G1*). G*N* refers to the maximum number of pairs accessed in parallel. (b) Box plot representing the distribution of LOCATOR processing latencies (Cycles Per Pixel) of the frames of KITTI sequences for different versions of the accelerator.

Table 2
Hardware parameters for LOCATOR, CPU, and GPU platforms.

	Parameter	Value
LOCATOR	Technology, Frequency	45 nm, 400 MHz
	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	14 nm, 4.2 GHz
	L1, L2, L3	0.25 MiB, 1 MiB, 8 MiB
	Thermal Design Power	91 W
GPU	Model	GeForce GTX1080
	CUDA cores	2560
	Memory	8 GB
	Thermal Design Power	180 W

ORB CUDA implementation we manually instrument to measure its performance.

On the other hand, we perform the experimental evaluation using the KITTI data set [9]. We use the odometry benchmark that comprises various recordings from drivings around the city of Karlsruhe. This benchmark consists of 22 grayscale stereo sequences.

Finally, we use an evolutionary computation framework for rapid prototyping and testing of ideas called DEAP [6] to implement the genetic algorithm described in Section 3.3.2.

5. 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 3. The configuration labeled as *CPU* corresponds to the high-performance OpenCV [1] software implementation of ORB feature extraction running on a CPU with parameters shown in Table 2. The ORB implementation contains a version optimized by explicit vectorization using SSE2 instructions for FAST detection. 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 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, thresholds, etc.

On the one hand, Fig. 14a shows the speedup of a single replica of the rBRIEF unit compared to other versions. The speedup is calculated taking into account 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 (PC), and Pipelining (PL) techniques compared to only using the Genetic Algorithm (GA) optimization. The *G4* and *G8* versions obtain a 1.12× and 1.25× 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, Fig. 14b 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. For clarity, we omitted its evaluation, 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. Fig. 15a details the 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. Fig. 15b shows the results of our experiments.

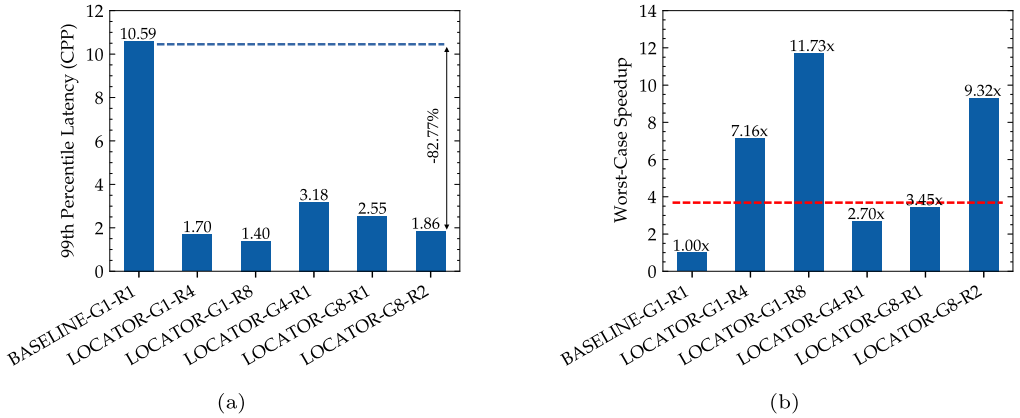

Fig. 15. (a) 99th Percentile Latency measured in Cycles Per Pixel (CPP) with all the optimizations proposed applied when processing the KITTI dataset. (b) LOCATOR speedup in the worst-case scenario compared to the baseline. The red dotted line indicates the minimum speedup threshold needed to reach our real-time consideration.

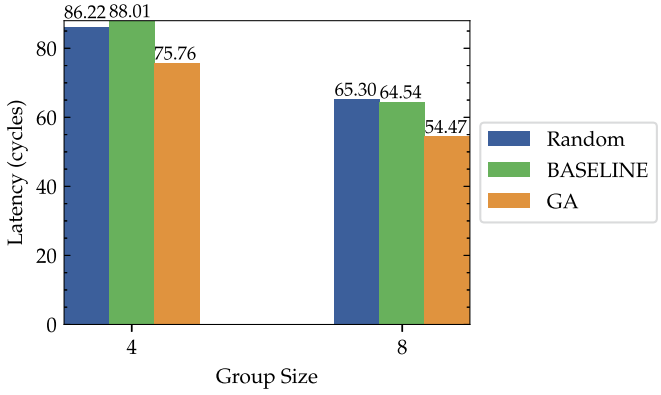

Fig. 16. Average number of penalty cycles due to bank conflicts when computing an rBRIEF descriptor. The results correspond to the latency incurred when accessing the baseline implementation with and without the GA optimization.

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×.

The red line in the Fig. 15b indicates the minimum speedup that must be obtained compared to the baseline performance to perform the ORB extraction in Full HD frames in 100 ms 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, Fig. 16 reports the effectiveness of the genetic algorithm (Section 3.3.2) 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 to be 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 Fig. 16.

Fig. 17a shows the speedup achieved by the ORB GPU implementation and LOCATOR with respect to the *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× speedup. Configurations *LOCATOR-G1-R4* and *LOCATOR-G1-R8* achieve speedups of 4.8× and 8.1× respectively. On the other hand, *LOCATOR-G4-R1* and *LOCATOR-G8-R1* deliver 4.82× and 5.66× speedups, respectively. Finally, *LOCATOR-G8-R2* obtains a 8× speedup with respect to the *CPU* and a 1.37× speedup against the *GPU*. 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 number of replicas. It also obtains 1.66× speedup compared with *LOCATOR-G1-R4* with half replicas.

The accelerator significantly reduces power dissipation, as illustrated in Fig. 17b. This huge power reduction is due to several reasons. First, the accelerator includes a specifically designed streaming architecture for ORB extraction that exhibits high throughput and large 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 14597×, 9609×, and 1.64× average reduction of energy consumption per frame compared with the CPU, the 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.

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× speedup against the CPU in the overall execution time. The accelerator achieves 9609× average reduction of energy con-

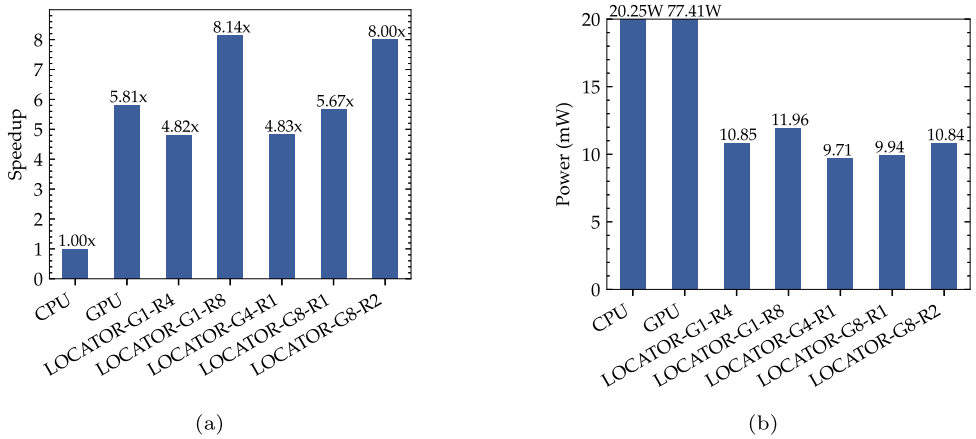

Fig. 17. (a) speedup achieved by the accelerator compared with the CPU. (b) Power dissipation of the CPU and the accelerator.

Table 3
Comparison with previous works. PPW stands for Performance Per Watt.

Work	Algorithm	Implementation	Performance	mW	PPW
[21]	FAST-BRIEF	ASIC, 130 nm, 78.3 k gates, 128 kB SRAM	122 fps, FHD, 200 MHz	182	670
[38]	ORB-like	ASIC, 65 nm, 127 k gates, 205 kB MEM	135 fps, FHD, 200 MHz	87.5	1542
[35]	ORB	FPGA, Arria V GX, 449 DPS, 206000 LEs, 231973 REGs, 1047 kB BRAM	110.9 fps, FHD, 230 MHz	5340	20
[5]	ORB	FPGA, Stratix V, 8 DPS, 25648 LUTs, 21791 REGs, 1208 kB BRAM	67 fps, VGA, 203 MHz	4559	14
[13]	FAST-BRIEF	ASIC, 65 nm, 28 kB SRAM	2170 fps, VGA	1131	1918
[30]	ORB	FPGA, XCZU9EG, 33 DPS, 28168 LUTs, 9528 REGs, 188 kB BRAM	108 fps, FHD, 200 MHz	873	123
[15]	FAST+RS-BRIEF	FPGA, XCZ7045, 111 DPS, 56954 LUTs, 67809 REGs, 78 BRAM	55.87 fps, VGA, 100 MHz	1963	28
This work	**ORB**	**ASIC, 45 nm, 32 kB SRAM**	**120 fps, FHD, 400 MHz**	**10.84**	**15260**

sumption per frame compared with the ORB GPU implementation. Finally, *LOCATOR-G8-R2* reduces 99th percentile latency an 82.77% and obtains a speedup of 9.32× in the worst-case scenario compared with the baseline accelerator, achieving real-time even in critical circumstances.

6. Related work

Table 3 provides a quantitative comparison with previous works. Our solution achieves high performance and shows the best energy efficiency, achieving a large improvement in Performance Per Watt (PPW).

Prior research used stream-based implementations on FPGAs. Work in [34,35] propose a streaming architecture similar to *LOCATOR-G1-R4*, without selective replication, the duplication cache, and the pipelining techniques, to extract ORB features using Harris-Stephens corners. Moreover, they propose an architecture for multilevel feature extraction with a replicated version of the accelerator per pyramid level. The rBRIEF bottleneck is solved through replicas with a different replication factor depending on the pyramid level. This solution employs an angle discretization of 64 values per sector. Our solution is different as it is based on an ASIC instead of an FPGA, and we avoid angle discretization to preserve accuracy.

Work in [22] proposes a streaming architecture for rBRIEF on FPGA, leveraging replication of window buffers to reduce latency of descriptor generation. Our proposal avoids replication to a large extent and exploits parallelism in the computation of the rBRIEF descriptor by processing multiple pairs of pixels at a time.

A SLAM accelerated solution is introduced in [15], proposing an architecture for feature extraction and matching on an FPGA while the rest of the components of SLAM run on a CPU. The authors propose a hardware-friendly pattern to generate BRIEF descriptors. This pattern reduces the rotation operation to a bit vector rotation operation instead of a costly trigonometric calculation. The drawback of this approach is the accuracy degradation and the unpredictable effects derived from changing the functional properties of rBRIEF.

Another architecture for ORB extraction is proposed in [29,30]. The architecture comprises a streaming front-end to generate the image scale pyramid and feature detection. The rBRIEF generation is carried out by a non-streaming back-end. This back-end consists of a four-issue super-scalar architecture that dynamically schedules points to compute the descriptor bits. However, a large discretization of the angles is used. Our solution is different as we do not sacrifice accuracy to simplify the hardware implementation. We solve the issues with rBRIEF computation by combining a static ordering generated with a genetic algorithm, a duplication cache, selective ports replication, and pipelining.

7. Conclusions and future work

7.1. Conclusions

In this paper, we propose *LOCATOR*: a **L**ow-power **O**RB a**C**celerator for **A**u**T**onom**O**us ca**R**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, avoiding accuracy loss 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 se-

lective replication of ports. Our experimental results show that LOCATOR achieves a speedup of 8× and a reduction in energy consumption of 14597× and 9609×, with respect to high-end CPU and GPU platforms, respectively.

7.2. Future work

There are several extensions of the presented work whose exploration could be fruitful in future works. The accelerator proposal only processes one tile of the Gaussian pyramid of images at a time. We believe that exploring the challenges and trade-offs of the on-chip descriptor generation of the pyramid could be a promising line of research. There is also potential to study the communication pattern of the accelerator with the host. Furthermore, future research should also consider the hardware programmability aspect more carefully. Programmability is a fundamental characteristic since it allows the hardware to be flexible and adaptable to algorithm changes. In an environment such as autonomous driving undergoing intense development, there may be significant changes in the algorithms, the parameters used, and the input data scale. We believe that providing an instruction repertoire that abstracts the accelerator's particularities could also be exciting.

CRediT authorship contribution statement

Raúl Taranco: Conceptualization, Investigation, Methodology, Software, Writing – original draft, Writing – review & editing. **José-Maria Arnau:** Conceptualization, Methodology, Writing – original draft, Writing – review & editing. **Antonio González:** Conceptualization, Funding acquisition, Methodology, Project administration, Supervision, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work has been supported by the CoCoUnit ERC Advanced Grant of the EU's Horizon 2020 program (grant No 833057), the Spanish State Research Agency (MCIN/AEI) under grant PID2020-113172RB-I00, the ICREA Academia program and the FPU grant FPU18/04413.

References

[1] G. Bradski, The OpenCV Library, Dr. Dobb's J. Softw. Tools (2000).
[2] M. Calonder, V. Lepetit, C. Strecha, P. Fua, Brief: binary robust independent elementary features, in: European Conference on Computer Vision, Springer, 2010, pp. 778–792.
[3] L. Caltagirone, M. Bellone, L. Svensson, M. Wahde, LIDAR–camera fusion for road detection using fully convolutional neural networks, Robot. Auton. Syst. 111 (2019) 125–131.
[4] Z. Chen, J. Zhang, D. Tao, Progressive lidar adaptation for road detection, IEEE/CAA J. Autom. Sin. 6 (2019) 693–702.
[5] W. Fang, Y. Zhang, B. Yu, S. Liu, FPGA-based orb feature extraction for real-time visual SLAM, in: 2017 International Conference on Field Programmable Technology (ICFPT), IEEE, 2017, pp. 275–278.
[6] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, C. Gagné, DEAP: evolutionary algorithms made easy, J. Mach. Learn. Res. 13 (2012) 2171–2175.
[7] J. Fuentes-Pacheco, J. Ruiz-Ascencio, J.M. Rendón-Mancha, Visual simultaneous localization and mapping: a survey, Artif. Intell. Rev. 43 (2015) 55–81.
[8] B. Gao, H. Lang, J. Ren, Stereo visual slam for autonomous vehicles: a review, in: 2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC), IEEE, 2020, pp. 1316–1322.
[9] A. Geiger, P. Lenz, R. 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.
[10] D.E. Goldberg, R. 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.
[11] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, T. Hamada, An open approach to autonomous vehicles, IEEE MICRO 35 (2015) 60–68.
[12] I.A. Kazerouni, L. Fitzgerald, G. Dooly, D. Toal, A survey of state-of-the-art on visual slam, Expert Syst. Appl. (2022) 117734.
[13] S.-K. Lam, G. Jiang, M. Wu, B. Cao, Area-time efficient streaming architecture for fast and brief detector, IEEE Trans. Circuits Syst. II, Express Briefs 66 (2018) 282–286.
[14] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M.E. Haque, L. Tang, J. Mars, The architectural implications of autonomous driving, ACM SIGPLAN Not. 53 (2018) 751–766.
[15] R. Liu, J. Yang, Y. Chen, W. Zhao, 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.
[16] D. Lockhart, G. Zibrat, C. Batten, PyMTL: a unified framework for vertically integrated computer architecture research, in: 47th IEEE/ACM Int'l Symp. on Microarchitecture (MICRO), 2014, pp. 280–292.
[17] M. Magnusson, A. Lilienthal, T. Duckett, Scan registration for autonomous mining vehicles using 3d-ndt, J. Field Robot. 24 (2007) 803–827.
[18] R. Mur-Artal, J.D. Tardos, ORB-SLAM2: an open-source SLAM system for monocular, stereo, and RGB-d cameras, IEEE Trans. Robot. 33 (2017) 1255–1262.
[19] R. Mur-Artal, J.D. Tardós, Visual-inertial monocular SLAM with map reuse, IEEE Robot. Autom. Lett. 2 (2017) 796–803.
[20] R. Mur-Artal, J.M.M. Montiel, J.D. Tardos, Orb-slam: a versatile and accurate monocular SLAM system, IEEE Trans. Robot. 31 (2015) 1147–1163.
[21] J.-S. Park, H.-E. Kim, L.-S. 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, IEEE Trans. Circuits Syst. Video Technol. 23 (2012) 832–845.
[22] T.H. Pham, P. Tran, S.-K. Lam, High-throughput and area-optimized architecture for rBRIEF feature extraction, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 27 (2018) 747–756.
[23] P.L. Rosin, Measuring corner properties, Comput. Vis. Image Underst. 73 (1999) 291–307.
[24] E. Rosten, T. Drummond, Fusing points and lines for high performance tracking, in: ICCV, vol. 2, Citeseer, 2005, pp. 1508–1515.
[25] E. Rosten, T. Drummond, Machine learning for high-speed corner detection, in: European Conference on Computer Vision, Springer, 2006, pp. 430–443.
[26] E. Rublee, V. Rabaud, K. Konolige, G. Bradski, ORB: an efficient alternative to sift or surf, in: 2011 International Conference on Computer Vision, IEEE, 2011, pp. 2564–2571.
[27] J.E. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W.R. Davis, P.D. Franzon, M. Bucher, S. Basavarajaiah, J. Oh, 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.
[28] K. Sugiura, H. Matsutani, A universal lidar slam accelerator system on low-cost FPGA, IEEE Access 10 (2022) 26931–26947.
[29] R. Sun, P. Liu, J. Wang, C. Accetti, A.A. Naqvi, 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.
[30] R. Sun, J. Qian, R.H. Jose, Z. Gong, R. Miao, W. Xue, P. Liu, A flexible and efficient real-time orb-based full-hd image feature extraction accelerator, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. (2019).
[31] Synopsys, Synopsys suite, https://www.synopsys.com/, T-2022-03-SP1 version.
[32] Y. Voronenko, M. Püschel, Multiplierless multiple constant multiplication, ACM Trans. Algorithms 3 (2007), 11–es.
[33] V.M. Weaver, M. Johnson, K. Kasichayanula, J. Ralph, P. Luszczek, D. Terpstra, S. Moore, Measuring energy and power with papi, in: 2012 41st International Conference on Parallel Processing Workshops, IEEE, 2012, pp. 262–268.
[34] J. Weberruss, L. Kleeman, T. Drummond, ORB feature extraction and matching in hardware, in: Australasian Conference on Robotics and Automation, 2015, pp. 2–4.
[35] J. Weberruss, L. Kleeman, D. Boland, T. Drummond, 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.
[36] C. Wolf, Yosys open synthesis suite, http://www.clifford.at/yosys/, 0.2 version.
[37] X. Zeng, Z. Wang, Y. Hu, Enabling efficient deep convolutional neural network-based sensor fusion for autonomous driving, arXiv preprint, arXiv:2202.11231, 2022.
[38] W. Zhu, L. Liu, G. Jiang, S. Yin, S. Wei, A 135-frames/s 1080p 87.5-mw binary-descriptor-based image feature extraction accelerator, IEEE Trans. Circuits Syst. Video Technol. 26 (2015) 1532–1543.

Raúl Taranco received his BSc degree in Computer Engineering in 2017 from Universidad de Cantabria (UC) and his MSc degree in High-Performance Computing in 2019 from Universitat Politècnica de Catalunya (UPC). Since 2018, he is part of UPC's Architecture and Compilers research group, where he is currently pursuing his Ph.D. His research mainly focuses on energy-efficient and high-performance hardware for autonomous driving systems. Contact him at taranco@ac.upc.edu.

Jose-Maria Arnau received Ph.D. on Computer Architecture from the Universitat Politècnica de Catalunya (UPC) in 2015. Next, he worked as a postdoctoral researcher at UPC in the area of energy-efficient architectures for cognitive computing. He joined Semidynamics in 2021, where he works as a hardware engineer developing RISC-V cores. Contact him at jose.maria.arnau@upc.edu.

Antonio González (PhD 1989) is a Full Professor at the Computer Architecture Department of the Universitat Politècnica de Catalunya, Barcelona (Spain), and the director of the Architecture and Compilers research group. He was the founding director of the Intel Barcelona Research Center from 2002 to 2014. His research has focused on computer architecture and compilers, with a special emphasis on cognitive computing systems and graphics processors in recent years. He has published over 400 papers, and has served as associate editor of five IEEE and ACM journals, program chair for ISCA, MICRO, HPCA, ICS and ISPASS, and general chair for MICRO and HPCA. He is a Fellow of IEEE and ACM.