Journal of Signal Processing Systems (2023) 95:863–875
https://doi.org/10.1007/s11265-022-01833-9

Design of an Application-specific VLIW Vector Processor for ORB Feature Extraction

Lucas Ferreira[1] · Steffen Malkowsky[1] · Patrik Persson[2] · Sven Karlsson[3] · Kalle Åström[2] · Liang Liu[1]

Received: 24 May 2022 / Revised: 21 October 2022 / Accepted: 27 December 2022 / Published online: 30 January 2023

Abstract
In computer-vision feature extraction algorithms, compressing the image into a sparse set of trackable keypoints, empowers navigation-critical systems such as Simultaneous Localization And Mapping (SLAM) in autonomous robots, and also other applications such as augmented reality and 3D reconstruction. Most of those applications are performed in battery-powered gadgets featuring in common a very stringent power-budget. Near-to-sensor computing of feature extraction algorithms allows for several design optimizations. First, the overall on-chip memory requirements can be lessened, and second, the internal data movement can be minimized. This work explores the usage of an Application Specific Instruction Set Processor (ASIP) dedicated to perform feature extraction in a real-time and energy-efficient manner. The ASIP features a Very Long Instruction Word (VLIW) architecture comprising one RV32I RISC-V and three vector slots. The on-chip memory subsystem implements parallel multi-bank memories with near-memory data shuffling to enable single-cycle multi-pattern vector access. Oriented FAST and Rotated BRIEF (ORB) are thoroughly explored to validate the proposed architecture, achieving a throughput of 140 Frames-Per-Second (FPS) for VGA images for one scale, while reducing the number of memory accesses by 2 orders of magnitude as compared to other embedded general-purpose architectures.

Keywords Vision-based SLAM · Feature extraction · ORB · ASIP

1 Introduction

Many hurdles need to be overcome as robots make the transition into full autonomous devices, chief among them is the efficient computation of Simultaneous Localization And Mapping (SLAM). The real-time inference complexity, from unstructured data to estimates of pose and 3D map in embedded systems, hinders the rollout of those appliances into the market. The use-cases of these autonomous devices are constantly growing, as with Unmanned Aerial Vehicles (UAVs) for instance, which are potentially employed in a broad range of different applications from the delivery of goods [1], to traffic monitoring [2], surveillance [3], search-and-rescue [4], precision farming [5], environmental surveying [6] and assessment of woodlots [7]. These applications feature in common the need for navigation and object avoidance, while performing mission-specific tasks to the fullest.

In the case of highly automated cars, in which the vehicle is in charge of performing all safety-critical functions for the entire trip, the navigation algorithms are performed in a heterogeneous computing system comprised of a combination of server-grade processors, and several GPUs. In [8], it has

✉ Lucas Ferreira
lucas.ferreira@eit.lth.se

Steffen Malkowsky
steffen.malkowsky@eit.lth.se

Patrik Persson
patrik.persson@math.lth.se

Sven Karlsson
svea@dtu.dk

Kalle Åström
kalle.astrom@math.lth.se

Liang Liu
liang.liu@eit.lth.se

[1] Electrical and Information Technology (EIT), Lund University, Lund, Sweden

[2] Center for Mathematical Sciences, Lund University, Lund, Sweden

[3] DTU Compute, Technical University of Denmark, Kongens Lyngby, Denmark

been shown that the combined power of such platforms sums up to 5 kW, when considering the necessity of redundant computations. To put this into perspective, the computation of such algorithms in level 4 autonomous vehicles draws roughly a quarter of the overall current median energy consumption of 19.6 kWh/100 km across the portfolio of Tesla cars [9]. On the other hand, for miniaturized devices such as pico-drones, [10] highlights the gap between the 100 mW power required for a 100 mg drone to reach stable flight, and the 3 W of power consumed by the general-purpose off-the-shelf processor integrated in the smallest commercial UAV executing SLAM algorithms.

In order to overcome this challenge, in this work we propose an Application Specific Instruction Set Processor (ASIP) whose instruction set and memory architecture are focused in feature extraction algorithms such as Oriented FAST and Rotated BRIEF (ORB), a crucial building block of indirect-sparse SLAM. The following sections are organized such that the SLAM background, prior art, and ORB algorithm are introduced in sections 1 to 3, followed by the dataflow, dependencies, and kernel operations analysis discussing the intrinsic parallelism opportunities in section 5. The other half concerns architectural details of the proposed hardware implementation and its evaluation.

2 SLAM Background

In the context of visual-based based SLAM, we can categorize the models into four different groups, based on how the input data to the model are processed (direct *vs* indirect), and the resolution of the 3D reconstruction (dense *vs* sparse) [11]. The choice of which SLAM paradigm to support in hardware plays a crucial role in alleviating the operational complexity and memory requirements of the resulting system. Therefore, a brief overview of the trade-offs involved is needed.

In general, SLAM is a greedy algorithm that, given the set of noisy measurement data $\mathbf{Y}$, estimates the camera poses and geometry $\{\mathbf{P}, \mathbf{X}\}$ that maximizes a likelihood function as

$$\{\mathbf{P}^*, \mathbf{X}^*\} = \text{argmax}_{\{\mathbf{P},\mathbf{X}\}} P(\mathbf{Y}|\mathbf{P}, \mathbf{X}). \tag{1}$$

Concerning the input, indirect methods [12, 13] obtain the measurement data $\mathbf{Y}$ by a two-step process. First, the images undergo a pre-processing step that transforms them into a sparse set of keypoints, in which point correspondences can be found across different frames, as illustrated in Fig. 1. A following probabilistic step is necessary in order to obtain a set of true correspondences, discarding outlier observations, and the results are interpreted as the noisy measurement data in Eq. 1. Consequently, the optimization target becomes minimizing the reprojection error between the reconstructed 3D points and their projections into the cameras, together with the measured data. Although sensitive to false point correspondences, these indirect methods are robust to scaling, rotation, and illumination changes, as the geometry is retrieved based on features. From a device perspective, indirect methods have the upper hand as images are sequentially compressed into features that encode the surroundings of pixels, and thus are suitable for privacy-aware systems.

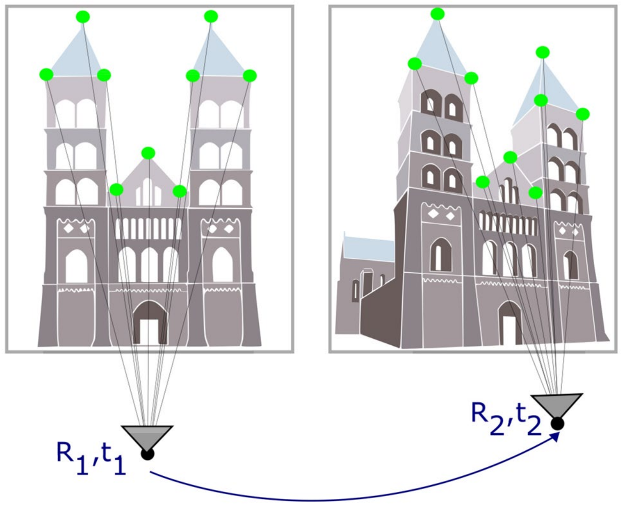

Figure 1 Representation of an indirect-sparse SLAM pipeline. Features (in green) are extracted and matched in a camera pair, which enables both the estimation of relative camera pose, and reconstruction of a sparse 3D model.

Direct methods [14, 15], on the other hand, use the raw RGB image data, capturing the actual differences of pixel intensities over time as the noisy inputs to the model. The process of pose estimation consists of finding a solution that projectively warps a reference frame into another based on minimizing the photometric error [16]. An advantage of these models over indirect methods is that pixels are not required to be recognizable by themselves, thus steps such as feature extraction and matching are not used. However, direct methods are sensitive to illumination variations and noise which are modelled as outliers, as pixels are compared based on a constant brightness assumption [17]. Another drawback is that direct methods have a narrower attraction basin [14], and consequently are more sensitive to errors of the initial pose estimates.

Regarding the reconstruction $\mathbf{X}$, sparse SLAM systems result in a cloud of scattered 3D points, which are inferred based on a sparse set of the input data, whereas dense methods seek to reconstruct all the observed 2D image points. The computation in real time of dense methods may not align with the constraints of low-power embedded systems, as every observed pixel must be reconstructed.

As the number of cameras and map points grow, even when considering sparse methods, they rapidly render the embedded computation of Eq. (1) infeasible. This is due to the fact that the size of the Jacobian matrix, computed in bundle adjustment, is proportional to the number of covisible points times the number of cameras that observed them. Consequently, navigation based on a very scarce map of the local surroundings paves the way forward for power-constrained devices, emphasizing the need for efficient hardware solutions supporting indirect-sparse SLAM algorithms.

Typically, the structure from motion pipeline for indirect-sparse SLAM systems consist of: extracting and matching features across the frames; retrieving an initial estimate of relative camera poses, and initial reconstruction of the 3D points based on the triangulation of correspondences; optimizing those initial pose and 3D-point estimates as in Eq. (1) based on the reprojection error (local bundle adjustment). As more frames are observed, the same processes are repeated, and a global bundle adjustment over all 3D points and cameras needs to be performed.

3 Prior Art

Motivated by the computational complexity and recurrence of feature extraction in indirect-sparse SLAM systems, it becomes a target for hardware acceleration. There are several feature extraction algorithms, most noticeable in the context of SLAM are: Scale-Invariant Feature Transform (SIFT) [18], Speeded Up Robust Features (SURF) [19], and ORB [20]. Although SIFT produces arguably the most robust features, the operations involved in the algorithm are more complex than the others, while the SURF algorithm is under patent protection. This leaves us with ORB which has features that are scale- and rotation-invariant, has lower complexity, and is an open source algorithm.

Several real-time implementations of ORB can be found in the literature, across different architectures. Originally ORB is implemented on a general-purpose CPU, achieving 65 Frames-Per-Second (FPS). In [21] an embedded implementation on a System on Chip (SoC) is found, reaching 179 FPS for 400×400 images. On FPGAs as in [22], a real-time multi-level implementation of ORB can be found, which identifies corners with only the Feature from Accelerated Segmented Test (FAST) and supports VGA images, at 67 FPS. ASICs such as [23] has been proposed for ORB, with high-throughput / energy efficiency, yet offering no support for multi-scale or Harris Corner Detector (Harris) feature extraction.

To the best of our knowledge our previous work was the first [24] implementation of a C programmable ASIP, specialized in ORB for a single scale, and the purpose of this current work is to provide further details on the memory access patterns and operational complexity involved in ORB, dataflow optimizations to reduce data traffic, and Very Long Instruction Word (VLIW) ASIP architecture optimizations.

4 ORB Feature Extraction

The main algorithmic steps in ORB feature extraction are scale space generation, followed by a two-level feature extraction, and the generation of the descriptors for each of the remaining keypoints, which are illustrated in Fig. 2. The inputs of each stage change substantially during the execution of the algorithm, the continuous stream of input pixels becomes sparse after the first level of feature extraction (step 2 in Fig. 2). After the second level (step 3), in practice, less than 1% of the input pixels remain and become features, which are assigned a descriptor and streamed out to the output. This steep gradient of input loads imposes a challenge to hardware implementations, and motivates the distribution of operations over VLIW slots.

4.1 Scale Space Generation

The scale space generation in ORB comprises two interleaved steps, which are replicated eight times as in [20]. The first of those is the convolution of the input samples, which are either pixels from the original image or from the previous iteration, with a 5×5 Gaussian kernel. The latter step embodies a bilinear interpolation function, which downsamples the convolution output by a factor of 1.2 per axis, or in other words reduces every five pixels into four (per axis). Since input size is reduced for each iteration of the scale space generation, the result is also called the image pyramid. Features are extracted across the different levels so that the resulting features are invariant to changes in scale, as objects appear bigger or smaller depending on how close the camera is. Ideally the final extracted features are uniformly sampled across the image, and the different scales.

4.2 Two-Level Feature Extraction

The keypoint extraction process focuses on the identification of small image patches with both high vertical and horizontal gradients, providing a means for consistent recognition of such regions independent of perspectives and illumination. In ORB the screening for those areas, namely corners, is attributed to two feature extraction algorithms, FAST and Harris, which are interposed by a filtering phase called Non-Maximum-Suppression (NMS).

Given the much higher complexity of Harris, the scales are prescreened with the less complex FAST algorithm, which suffers from high response along the edges, but nonetheless discards most of the pixels over the different scales,

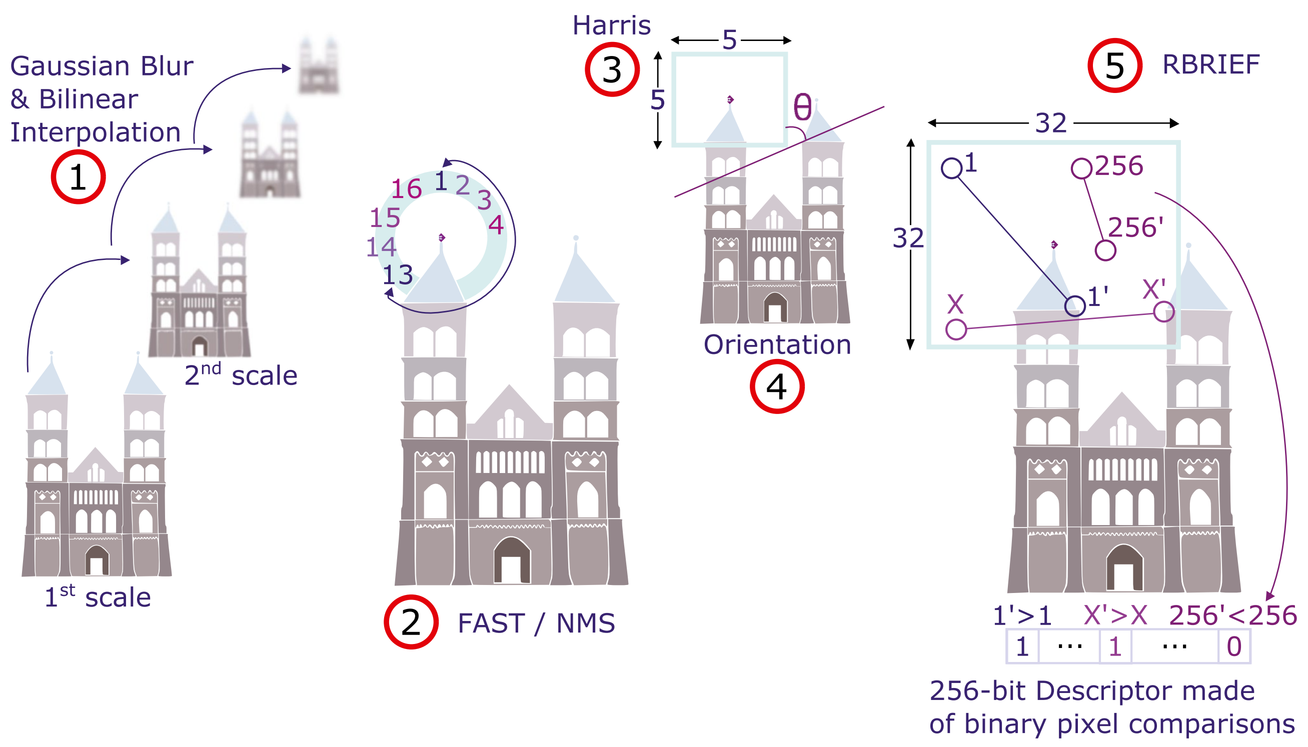

Figure 2 High level representation of ORB feature extraction. Depicted in 1 the scale space generation process; In 2 and 3, the two-level feature extraction routine; In 4 and 5, the tasks related with the descriptor generation for each keypoint.

with typically around 2% of the pixels across the image pyramid lasting this criteria [25]. A Bresenham circle of radius 3, centered on each pixel under FAST consideration, and a user-specified threshold comprise the input of the algorithm. If 12 (out of 16) consecutive pixels in the circle are either brighter or darker than the circle's center plus the threshold, then the pixel in question is a FAST keypoint. One shortcoming of FAST is the tendency that many features are extracted at the same regions in the image, forming bundles which opposes the ideal of well-distributed features across a scale. Consequently, NM

S is applied to the FAST output to pick out the keypoint with the highest combined contrast score.

A 5×5 patch of image, in the appropriate scale, centered in each feature of the sparse set post FAST and NMS keypoints constitutes the input load of Harris. Within each patch, the vertical and horizontal gradients are computed by means of convolution with Sobel filters. These derivatives are then combined into a matrix M, as in Eq. (2)

$$M = \sum_{(x,y)\in W} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}, \tag{2}$$

where I_y, and I_x are analogous to the vertical and horizontal convolutions of the patch with the correspondent Sobel filter. The Harris response r is then obtained as shown in Eq. (3),

$$r = \det(M) - k\,\mathrm{tr}(M)^2, \tag{3}$$

where k is an empirical constant within the closed interval $[0.04, 0.06]$. In ORB, selecting the N best features is equivalent to picking the N highest Harris response keypoints.

4.3 Descriptor Generation

A descriptor provides a means for a keypoint to be matched across different frames, as they encode the aspects of the keypoint and its surrounding into a binary vector. To put it into context, given one descriptor in a reference frame and a set of all descriptors from another, a point correspondence is found by selecting which descriptor from the set minimizes the Hamming distance with the one in the reference frame. Consequently, the degree of similarity between two features can be retrieved thanks to its descriptors.

In ORB, descriptors are generated in two steps, first by assigning an orientation to each Harris' keypoint, followed by the Rotated-BRIEF (rBRIEF) algorithm, which takes the orientation into account and results in the descriptor. The step of assigning the orientation to each keypoint renders the extracted features robust to variations in rotation, one of the key qualities of ORB.

Given a 5×5 image patch centered in a Harris keypoint, the orientation is found by taking

$$\theta = \text{atan2}\left(m_{10}, m_{01}\right), \tag{4}$$

where m_{10} and m_{01} correspond to the first order image moments as in

$$m_{pq} = \sum_{x,y} x^p y^q I(x, y). \tag{5}$$

The descriptors are constructed with rBRIEF, which consists of a rotation-aware variation of Binary Robust Independent Elementary Features (BRIEF). This algorithm includes two inputs, a patch of the related scale space of size 32×32 centered in the keypoint, and the previously calculated orientation. Within the patch, 256 pairs of pixels are compared in order to check which element is brighter than the other. Step 5 in Fig. 2 illustrates the pattern and three comparison pairs for a given orientation. For other angles the pattern has to be rotated accordingly. The binary result of those comparisons are stored in a 256-bit string, which forms the descriptor of the keypoint.

5 Operation Analysis and Architecture Considerations

In this section, we perform a detailed analysis on the kernel operations involved in ORB feature extraction and the corresponding data flow.

5.1 Data flow and dependencies

As images are produced by the sensor in a row-major sequential form, known as scanlines, different algorithmic blocks in ORB could potentially be carried out in parallel as the scanlines are streamed. The advantages of such a streaming architecture are twofold. First is the savings in memory space, as only a small subset of the input image and scales need to be stored on chip. Second is the reduced latency, as the algorithm starts as soon as the sensor data is available.

Figure 3 illustrates the overlapping schedule of the sub-blocks in ORB based on their dependencies and the flow of

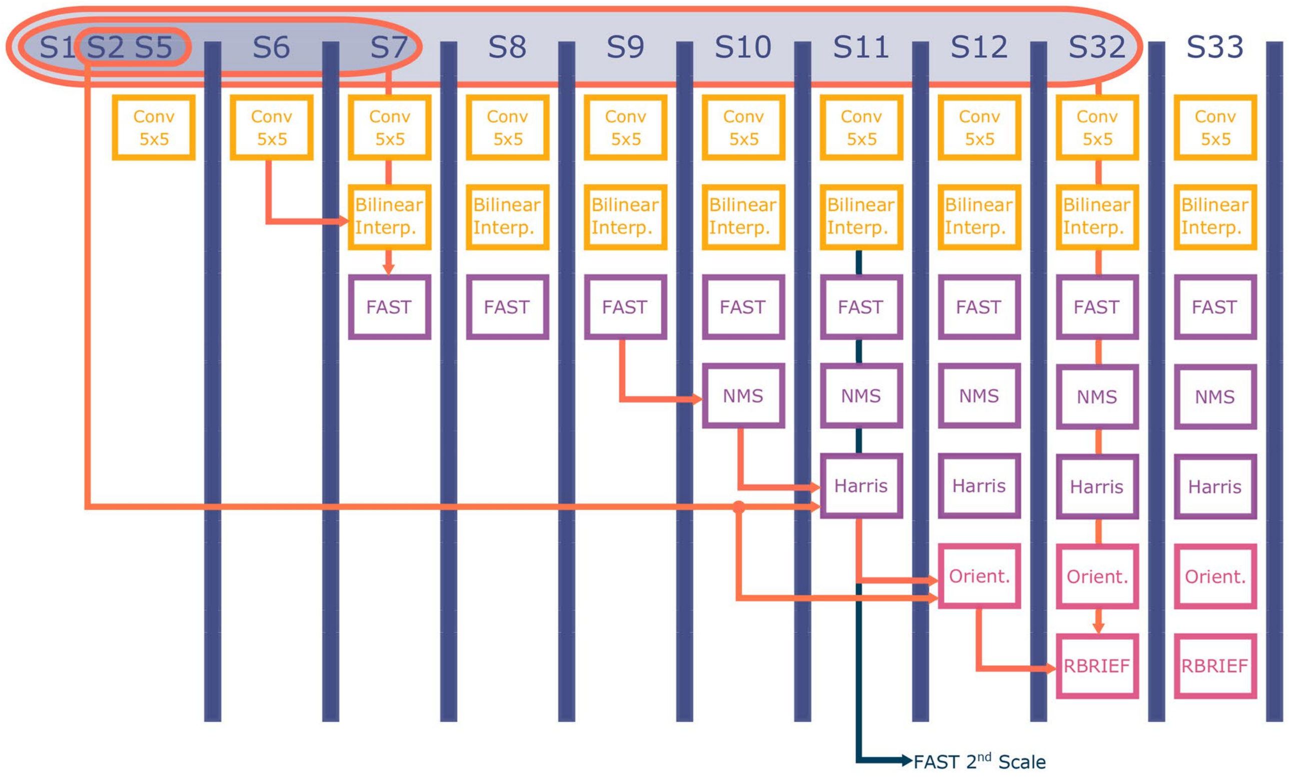

Figure 3 This figure illustrates the scheduling of operations based on the dependencies of each sub-algorithm in ORB. Space-space algorithms are represented in yellow, keypoint extraction and descriptor generation, in purple and pink respectively. Note that the sub-algorithms can have overlapped execution, since each operate on different image/scale space parts.

data from the image sensor. Scanlines are represented with the S prefix, varying from scanline 1 to 33 (e.g S1..S33). The data dependencies are indicated by the orange arrows, and are drawn only once across different blocks at the first moment in which the dependencies are satisfied, for visualization purposes. Note, for instance, that the orange arrow from FAST to NMS is drawn when three FAST scanlines are available, which is when the first NMS has its dependencies met. Although the convolution between input image and 5×5 Gaussian kernel can start as soon as the first scanline is streamed out, it is convenient to buffer five scanlines before executing the operation, as illustrated in the image. The reason is that the number of memory accesses can be reduced if all the data needed to produce one convolution result is read with a single load operation. The same argument is valid for the subsequent convolution blocks that operate on results of the prior scales. On the the other hand, for the bilinear interpolation, it suits that two scanlines of convolution results are buffered in order to minimize the number of memory accesses, as the dependency arrow indicates in the Fig. 3. By these means a single load of the required input data (2×2 pixel patch) can be performed for each pixel of the result.

As regarding FAST, every pixel across the scale space that can be centered in a Bresenham circle is evaluated, with each circle spanning over seven scanlines and columns. Thus, in total, a disjoint set of 17 pixels (16 lying on the Bresenham circle and 1 center pixel) need to be loaded in order to decide whether or not a pixel is a FAST keypoint. According to these rates of generation and geometry of the input, the number of accesses to memory and buffering can be reduced by a joint effort of ultimately loading one Bresenham circle per cycle. For this, it suits that FAST is performed with seven scanlines of initial latency, as it waits for a moving window comprising the seven latest scanlines to be valid per scale space, as shown in Fig. 3.

The dataflow and dependencies for Harris and the evaluation of a keypoint's orientation are similar, as they operate on the same patch of image with 5×5 shape, and are activated based on the result of a prior operation. The same is valid for rBRIEF, with the only difference of the input size, which spans over 32 scanlines, and therefore impose a lower bound on the number of scanlines to be cached per scale space.

One cross-level optimization can be performed by the identification of the local data traffic between the different algorithms, as for instance within the scale-space generation or between FAST-NMS algorithms. In the latter case FAST consumes the appropriate input and its output becomes the input to NMS as illustrated in Fig. 3. Another cross-level solution relies on the observation that Harris and the keypoint's orientation algorithm share the same input. Those two findings were used to derive the VLIW slots of the proposed solution, such that the degree of overlapped execution is not completely parallel, with the aim of striking a balance between performance and hardware costs.

5.2 Kernel Operations & Operational Rate

In order to explore the parallelism of the aforementioned algorithms, while minimizing the number of accesses to memory, a deeper understanding of the kernel operation in ORB, together with its rates of generation and consumption is necessary. Table 1 gathers the operations across the various algorithmic steps.

Parallelism with the respect to the operations in the scale space generation process can be explored within a scale by block-wise convolutions along the buffered scanlines, or across the different pyramid levels, or both. It is

Table 1 Kernel Operations within the various algorithmic phases in ORB. Vector operations listed have the potential for Single Instruction, Multiple Data (SIMD) execution, which can be executed either in a single cycle, if enough resources are available, or in a multi-cycle fashion compromising performance and possibly requiring additional intermediate buffers.

Operation	Scale Space	FAST & NMS	Harris & Orientation	rBRIEF
Vector	$\circledast\ A_{5\times5}$	$\mathbf{a}_{12\times1} > a \parallel \mathbf{a}_{12\times1} < a$	$A_{3\times3} \times A_{3\times3}$	$a_{256\times1} > 0$
	$A^{-1}_{4\times4}$	$\max\{A_{3\times3}\}$	$A_{3\times3} \cdot A_{3\times3}$	$a_{256\times1} \cdot a_{256\times1}$
	A^t		$\det(A_{2\times2}) \parallel \mathrm{trace}(A_{2\times2})$	$a_{256\times1} - a_{256\times1}$
	$A_{4\times4} \times a_{4\times1}$		$\sum^{3,3} A_{ij} \parallel \sum^{5,5} A_{ij}$	
	$A_{4\times1} \cdot a_{1\times4}$		$a_{1\times5} \cdot a_{1\times5}$	
Scalar		$+$	$-\ /\ \times\ /\ \div\ /$ atan2	
Workset	All pixels in each scale	Most pixels in each scale[1]	$20\% \rightarrow 2\%$	less than 1%

[1]This workset comprises all pixels except the ones in the image borders, which cannot serve as a center to a Bresenham circle.

advantageous to have matched throughput between the number of pixels read per cycle and the rate in which they are processed, in order to minimize the pressure on the register files or local buffers. For instance, in the case that a 5×5 image patch is loaded in one cycle, ideally a matrix multiplication with a 5×5 kernel can be also performed in one cycle, otherwise the data has to be buffered according to the consumption rates.

As FAST filters out most of the pixels across the image pyramid, it introduces an intermittent work regimen for all the subsequent processing steps, of which the first in line is NMS. Its inputs encompass the combined FAST contrast and pixel's coordinates across the scale space for both the current pixel under examination and its first neighborhood. If all pixels in the 3×3 patch are keypoints, a combination of a single load to access to the patch, together with a concurrent max pool operation simplifies the dataflow and eliminates unnecessary intermediate buffering steps.

The sudden change of work regimen from operating in every pixel across the scale space to 20% or less of the pixels, imposes a challenge for hardware implementations as the control flow of the nested loop, which contains subsequent algorithms, is seldom activated. A dynamic branch prediction scheme, exploring the tendency that many keypoints are found in the same regions of the image, may contribute to an efficient implementation.

Harris links together a sequence of three operations, namely the calculation of image derivatives, computation of M in Eq. (2), and evaluation of r in Eq. (3). As the derivatives operate on the same input, which is the 5×5 patch centered on the keypoint, they can be performed in parallel once the necessary data are made available. Data movement can be reduced in the following step, if the three distinct products of derivatives can be performed in a single cycle, given the shared input. At last, the determinant of M, a 2×2 matrix, and its trace (scaled by a constant) can also be handled in parallel to compute the Harris response of a single post-NMS FAST keypoint. The sequential nature of those operations creates opportunities for pipelined SIMD architectures, that have the advantage of higher throughput.

The process of assigning the orientation for a keypoint is also a sequential process, which is suitable for the exploration of SIMD operations. The calculation of the first-order moments operates on the same data, and its operations consists of a weighted sum based on the pixel location within the input patch. Given that the load operation can be performed in a single cycle, the data can be consumed in similar rates by calculating the first-order moments in parallel, computing the weighted sum of each as a dot product followed by a sum of columns, transforming a row vector into a scalar. Once the moments are obtained, iterative methods such as CORDIC may be employed to calculate atan2, in order to retrieve the angle.

In rBRIEF, the challenge lies on how to explore the parallelism available, given that each comparison pair is independent, while the pixels pairs to be loaded are distant from each other image-wise, together with the fact that the pattern selection varies with the angle which is obtained during runtime. The extent to which this algorithm can be parallelized contributes significantly to the overall system throughput, and in the worst case, if no parallelism is considered, 512 cycles would be required for just loading the operands for one keypoint. This means, for real-time operation at 30 FPS and extracting 2000 features per frame, the system needs to run at least at 33 MHz, which is a lot given the amount and complexity of all other operations in ORB.

5.3 Key findings

In summary, the key findings of this operations and dataflow analysis section are:

- Processing in ORB can leverage the streaming nature of how the image is produced, as scanlines, to alleviate the on-chip memory requirement, as only a small subset of the most recent scanlines per scale space are used when adhering to stream processing;
- The sub-algorithms in ORB are sequential, cascading a sequence of inter-block dependable operations (*e.g* Convolution $5 \times 5 \rightarrow$ Bilinear Interpolation $\rightarrow$ FAST $\rightarrow \dots \rightarrow$ rBRIEF), but at the same time they can have overlapped execution with earlier sub-blocks operating in more recent scanlines in order to meet real-time constraints;
- Given the rBRIEF requirements, the minimum of the 32 most recent scanlines are necessary to be stored on-chip per scale-space;
- A combination of single-cycle memory accesses to usual patterns in ORB - patches: 2×2, 3×3, 5×5, Bresenham circle, and rBRIEF related - in connection to vectorized execution of operations (as in Table 1) in equal rates, increase performance of the overall system while reducing internal data movement;
- The FAST-NMS algorithms operate on local data traffic, while Harris and keypoint's orientation algorithms have shared data input. On the other hand, rBRIEF has high complexity given the irregular access pattern.The slots of a VLIW processor can leverage those properties.

6 The ASIP Architecture

The operation analysis of the prior section conveys several opportunities for the exploration of data parallelism inherent to the different matrix operations involved with ORB feature extraction, taking into consideration the input/output data rates to minimize redundant memory accesses, data movement, and storage of the intermediate values. Leveraged by those findings an ASIP architecture is expanded on, which comprises scalar and vector slots, augmented by a reconfigurable multi-access pattern vector memory that provides single cycle access for the varying related patterns.

As the different algorithms in ORB have overlapped execution, within and across the different scale spaces, as illustrated in Fig. 3 with the scheduling of operations, we propose a VLIW architecture striking a balance between the different work-load regimens and operational complexity or the extent of the sequential operations involved. In outline, three vector slots specialized in FAST and NMS, Harris and keypoint's orientation, and rBRIEF, and an additional lightweight and general purpose RISC-V scalar slot, all described in detail in later subsections. A high-level diagram of the proposed architecture can be seen in Fig. 4, which focuses on data, storage and control units while masking away the micro-architectural details.

6.1 VLIW Partitioning

As operations are concurrently performed across the width of the VLIW instruction, the number of slots implemented is a trade-off between a non-flexible architecture with extensive parallelism and a RISC-like general purpose processor performing a single operation per cycle. A decision of a four-slot VLIW was made in order to make the processor flexible enough to accommodate further extensions to various feature extraction algorithms, while leveraging the fact that the sub-blocks in ORB can operate in different regions of the image at the same time.

The primary factor motivating our VLIW slot division was to maintain a balanced operation complexity so that the maximum frequency achieved by the ASIP is not limited by the critical path of a single complex operation. It also contributes to a unified number of pipeline stages across the slots, making the micro-architecture simpler as all slots can write back at the same stage. With this purpose, for instance, the wide SIMD and sometimes dependent operations in rBRIEF has its own VLIW slot, whose operations' complexity is equivalent to the other slots.

In addition, the decision of grouping together sub-blocks with similar work-load regimen also played an important role in the division of the VLIW slots, which is a compromise between slot utilization versus resource sharing, minimization of data movement, and size of intermediate results buffer. We have chosen to populate one VLIW slot with FAST (and NMS) instructions, which is performed for most

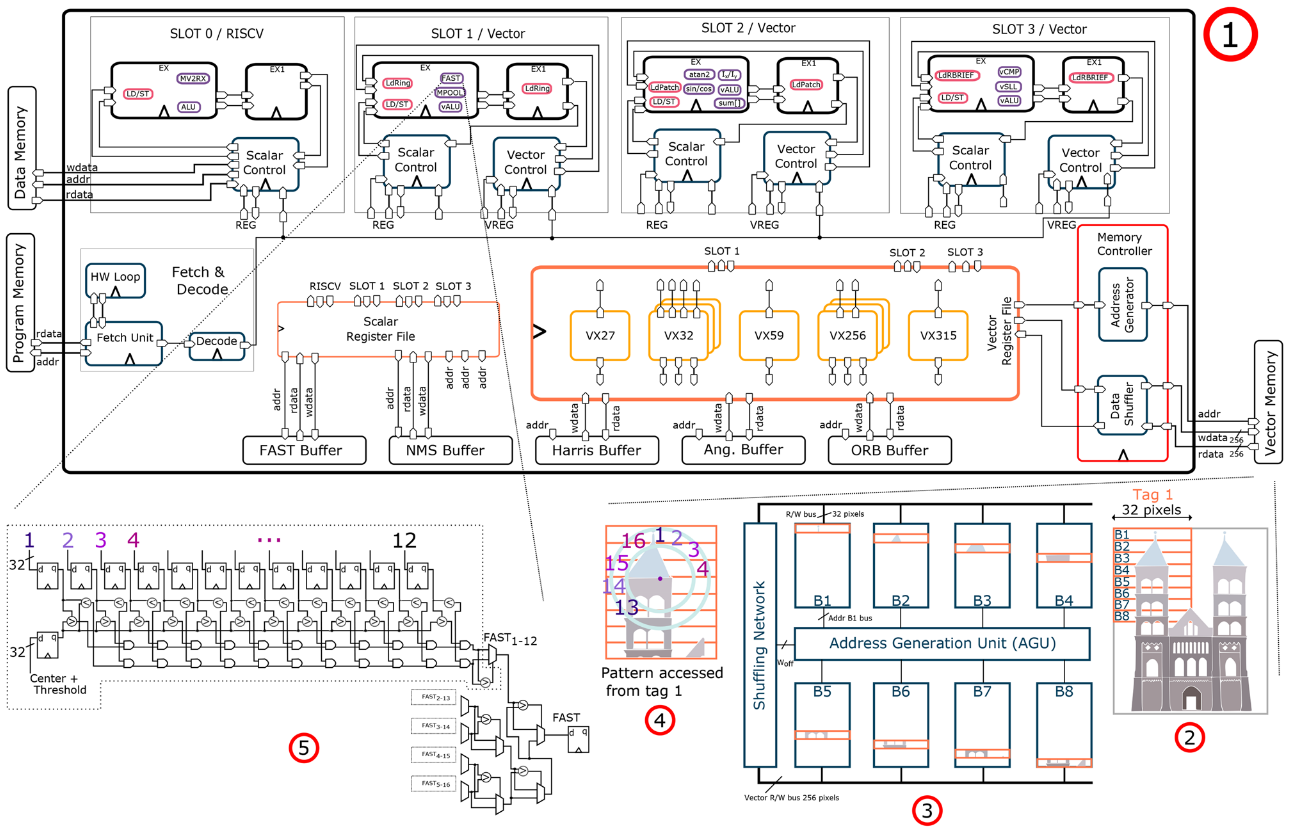

Figure 4 Bird's-eye view of the processor.

of the pixels across the scale space, even though the utilization of other slots is reduced, so that the buffer for intermediate results is simplified, and data movement reduced. Figure 4 part 1 illustrates the partition of the slots, and how the parallelism within the sub-blocks of ORB is explored.

6.2 Vector Memory

The vector memory as indicated in Fig. 4 is an implementation of the reconfigurable multi-access pattern vector memory in [26], conveniently enabling single-cycle access to the various and recurrent load operations in ORB. The vectorized load of the irregular, and sometimes disjoint, patterns made possible by this memory allows for the sub-blocks in ORB to read and consume data at the same rates, minimizing the number of memory accesses.

A generalized architecture for the vector memory comprises a plurality of banks, all addressed by an Address Generation Unit (AGU), whose readout is intercepted by a shuffling network in order to select the final pixels that belong to the load pattern. Each memory bank consists of words which are arrays of pixels, of which its length is a design choice based on the geometry of the supported single-cycle access pattern and the complexity of the AGU (more details in this trade-off in [26]).

In Fig. 4 part 2 the image distribution process is illustrated. Internally, an input image is subdivided into tags, which are rectangular-shaped patches of images, whose width is equivalent to a wordlength number of pixels, and height equal to the number of banks.

The image is distributed over the different banks such that each row of a given tag is assigned to a different memory bank, as in part 3 of Fig. 4. In other words, an image row is broken into several segments (each of tag width, which is a wordlength), and all those map into consecutive addresses of the same bank. As shown in part 2 of the figure, the other image rows undergo the same process, but are mapped to the other banks, which results that a row is mapped to the same memory bank at every number of banks equal amount of image rows.

Since a tag is scattered row-wise over the different banks, the number of tags that can be fetched in a clock cycle is given by the number of memory ports, and constitutes the largest possible pattern that can be fetched at once. If, for instance, dual ports are used patterns of pixels spanning over two adjacent tags can be loaded per cycle.

A readout operation consists of four steps: pattern specification, addresses calculations in the AGU, memory access, and the selection of pixels within bank words. The first step consists of specifying the pattern to be loaded as a vector of pixel positions, based on the image coordinates that forms the pattern. If the pattern spans over a tag, or if the combination of pixels are not conflicting (*i.e* not having more accesses per bank than the number of ports), the AGU will calculate the addresses to be fetched at run time, in each bank, and the position of the pixels that belong to the pattern in each word, called word offsets. Lastly, after the words are fetched from the different banks, the shuffling networks stream out the pattern based on the previously calculated word offsets as in part 4 of the figure.

In this work, an instance of this vector memory is implemented and it comprises eight banks, each featuring a single port with words that are 256 bits long (accommodating 32 pixels), and overall memory size of 64 KB, which is indeed over-dimensioned when accounting for the necessary scanlines of all the eight levels of scale space in ORB for VGA images. It provides single-cycle access to all the recurrent load patterns in ORB, except rBRIEF which requires in the worst possible case eight cycles.

As this ASIP is intended to be best placed near-sensor, where the image is directly saved into the vector memory, we assume that the image is already pre-loaded. In case the image needs to be fetched, in order to preserve performance, pseudo-dual ports containing one R/W port, and another R only port is required, which has a slight area overhead in comparison to single-port memories.

6.3 Scalar Slot and Register Files

The scalar slot features a lightweight RISC-V, including the base instruction set RV32I, with the standard 5 pipeline stages (IF, ID, EX, EX1, WB). This slot is primarily responsible for the control flow of the program executed in the ASIP, *i.e* controlling the PC, and performing scalar operations. However, the RISC-V has also the duty of loading the image data into the different vector registers, and thus custom extensions to the original Instruction-Set Architecture (ISA) were implemented. In addition the RISC-V is also extended with a fetch unit providing zero overhead loops.

The various register files that are implemented in our architecture, can be divided into scalar and vector types. In Fig. 4, in the vector register file, the one named as VX32, for instance, corresponds to a set vector registers capable of storing various 32-bit elements. The different registers' types are from the fact that operands for the custom intrinsics are either pixel coordinates within the scale-space, here encoded with 27 bits (12 for the height, 12 for the width, and 2 for the scale space), a 32-bit general pixel-related data type, a combination of both ($27 + 32 = 59$), a descriptor, or the output data ($256 + 59 = 315$). Thus, the main motivation for the various length and types of vector registers was to reduce code size for unnecessary type castings, while increasing the utilization of the employed hardware, since the different registers are tailored for the intrinsics of each slot.

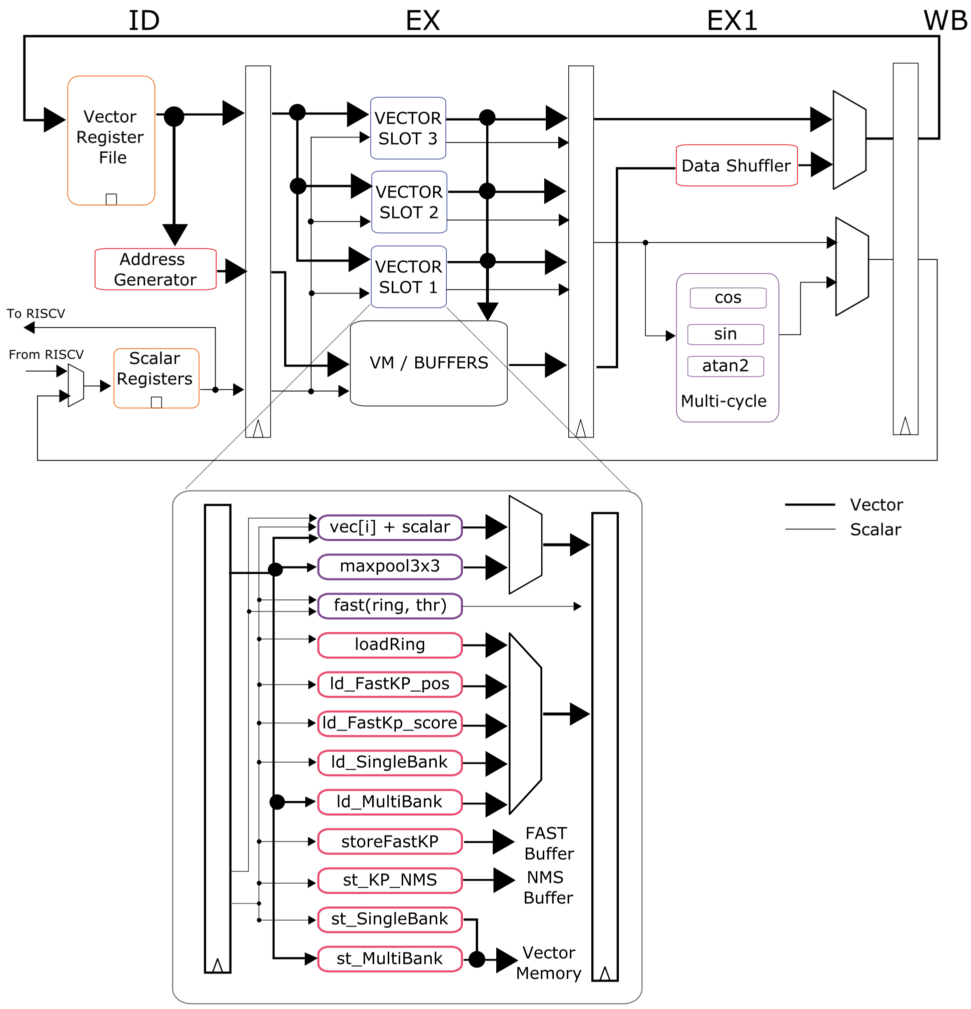

Figure 5 The vector slot, with datapath components of slot 1 in further details, from the decode to write-back stage. Storage related units and instructions are highlighted in shade of red, while in purple the computation instructions.

6.4 Vector Slots and Intermediate Buffers

The vector slot 1 is in charge of performing operations related to the FAST keypoint extraction and NMS, as illustrated in greater detail in Fig. 5. It features, for instance, the vectorized compare ring intrinsic, of which given one Bresenham circle, a center pixel, and a threshold, it evaluates if any of the 5 possible combinations of 12 consecutive pixels are either brighter or darker than the center. An illustration of the hardware implementation responsible for execution of this instruction is in part 5 of Fig. 4.

Slot 1 also features the ordinary memory intrinsics capable of the single-cycle load and store from single or all banks from the vector memory. In addition, as the FAST outputs are ultimately coordinates of pixels that meet the criteria together with the comparison scores, these are merged together in a data structure which is temporarily stored in the FAST buffer. The benefit of this approach is that FAST could potentially operate in different scales and at distinct image regions, interleaved by the NMS operations which shares the same slot. Similarly, NMS has its own buffer to store the keypoints that meet its criteria.

The vector slot 2 comprises operations related to Harris and the evaluation of a keypoint's orientation. It features, for instance, operations such as the matrix multiplication of 3×3 matrices, the vertical/horizontal image derivatives, and evaluation of atan2 (via iterative multi-cycle CORDIC). Furthermore, the vector load of a 5× patch centered in an image pyramid coordinate, and storage of Harris and orientation data structures are also assigned to this slot.

The vector slot 3 focuses on the rBRIEF, which, for instance, includes a vectorized rotation of the zero-angle BRIEF pattern based on the orientation of the Harris keypoint found, a vectorized comparison for the intensities of 16 pixel pairs per cycle, and vectorized shift-logical left of 16 elements at a time. Intrinsics which access either the vector memory or the orientation intermediate buffers to feed those operations are implemented, as well as to buffer the final keypoints extracted.

The buffers are exposed to the slots either through to the scalar or the vector registers as shown in Fig. 4, and as anecdotes of its instructions the operations store FAST keypoint (storeFastKP), and store keypoint post-NMS (st_KP_NMS). In those buffers, the data structure that represents a keypoint can be saved, according to its level of readiness. For instance, at Harris stage, a keypoint has a 32-bit response, and a 27-bit position within the scale-space, thus in Harris buffer a 59-bit value is stored, or the closest superior even number of wordlength (60-bits) given commercial SRAM availability. The overall capacity of the utilized buffers are 35 KB, which are also overdimensioned given commercial constraints for minimum number of words, maximum wordlength, and as mentioned parity of word.

6.5 Programmability

The ORB feature extraction processor has been implemented with Synopsys ASIP-Designer, which among other things produces RTL and a C/C++ compiler based on the user high level description of the processor model. The code snippet in Listing 1, illustrates how FAST is programmed with our ASIP.

```
1  bool FAST(pixel_sxy_pos pixelCoordinates,
2            int Threshold,
3            bufferAddr chess_storage(FASTvw)*
4            p_FastBuffer,
5            uint4 FastBuffer_wordOffset)
6  {
7      v17uint32_t pixelRing =
8          loadRing(pixelCoordinates);
9      uint32 fastScore =
10         fast(pixelRing, Threshold);
11     if(fastScore > 0)
12     {
13         storeFastKp(p_FastBuffer,
14                     fastScore,
15                     pixelCoordinates,
16                     FastBuffer_wordOffset);
17         return true;
18     }
19     return false;
20 }
```

Listing 1: C code with function-like intrinsics for FAST evaluation

The snippet includes hardware-primitives that are added to the C compiler such as loadRing(), in line 8, and fast(), in line 10, which respectively loads the 17 pixels in the Bresenham circle, and evaluates FAST. If the pixel is indeed a keypoint, fast score will be greater than zero, and the keypoint data structure containing the score, pixel coordinates in the scale space will be saved in the FAST buffer.

Table 2 Synthesis results with area breakdown for the ASIP.

Block	Area [mm^2]	% of Total
Vector Memory	0.236	41.7%
Program Memory	0.029	5.1%
Data Memory	0.014	2.5%
Buffers	0.184	32.5%
ASIP	0.103	18.2%

7 Implementation and Evaluation

The design has been synthesized in 22 nm technology with the area breakdown represented in Table 2.

Cycle accurate simulation results, obtained for the case of ORB feature extraction of 1000 keypoints from a single-scale VGA image, indicates that the ASIP takes roughly 3.1 million cycles to process one image. Power-annotated (VCD) post-synthesis simulations, with such setup, shows that the maximum achievable frame rate is 140 FPS, and considering the worst process corner, the power consumption is estimated at 90 mW at a frequency of 430 MHz. This power analysis does not include off-chip accesses for streaming out the extracted keypoints, given that only 32 can be kept locally.

Table 3 gathers state-of-the-art implementations of ORB, and puts this work into context. To the best of our knowledge there is no processor architecture focused on feature extraction in general, and more specifically on ORB. Thus, it is difficult to conduct fair comparison between implementations on different platforms and targeting for different applications, as with the CPUs [20], SoC [21], FPGAs [22, 27] and ASIC [23] implementations. On the other hand, the robustness of the extracted keypoints in Harris-lacking implementations such as [22, 23, 27] is compromised since the keypoint extraction phase only utilizes FAST. FAST has high response along edges [20], thus not specifically identifying only image corners.

Overall more generic architectures suffer from high power consumption and lower throughput, but features programmability or reconfigurability. In contrast, dedicated ASICs such as [23] have the highest energy efficiency, but cannot adapt to algorithmic parameters. ASIPs, such as ours, can be of relevance as it strikes balance between both architectures, with a high throughput at a quite reasonable energy-efficiency levels.

When it comes to the precision, a design-choice of keeping a constant 32-bit fixed-point was made in our architecture, opposed to 8-bits in [22] or 21-bits in [23], as the proposed architecture is a processor and needs to support to a certain extend algorithmic explorations which may require more precision. Additionally, in our approach, no Harris features are dropped, leaving to the user the design choice of sort

Table 3 Current work in the light of hardware implementation of ORB feature extraction.

	[20]	[21]	[22]	[27]	[23]	**This**
Input Size / # Features	VGA / 1000	400 × 400 / 500	VGA / N/A	FHD / 1000	FHD / 2000	**VGA / 1000**[1]
# Scales	7	3	2	4	1	**1**[2]
Extractor	FAST + Harris	FAST + Harris	FAST	FAST	FAST	**FAST + Harris**
NMS	Yes	Yes	No	Yes	Yes	**Yes**
Architecture	GPCPU, Intel i7	SoC, TI TDA2x - EVE	FPGA, Stratix V, 1.18 MB BRAMs	FPGA, XC7Z045, 73 KB BRAMs	ASIC, 45 nm, 32KB SRAM	**ASIP, 22 nm, 64 KB SRAM**
Performance	65 FPS, 2.8 GHz	179 FPS, 650 MHz	67 FPS, 203 MHz	42 FPS, 100 MHz	120 FPS, 400 MHz	**140 FPS**[3], **430 MHz**[3]
Power [mW]	N/A	N/A	4590	N/A	10.34	**90**[3]
Energy Efficiency [mJ/Frame]	N/A	N/A	68	N/A	0.086	**0.642**[3]

[1]These parameters are flexible as this processor is programmable.
[2]The current version of the processor supports multi-scale keypoint extraction, when performing the scale-space operations in Tab. 1 using the standard non-optimized RISC-V operations.
[3]These estimates are a function of image size, and the number of keypoint extracted.

the features data structures, in which the Harris response is encoded.

8 Conclusion

As discussed in the previous section, this work makes a case for processors dedicated to feature extraction, currently efficiently supporting ORB. This paves the way forward for cloud-based sparse visual-SLAM algorithms given the low-power, near-sensor computing, and privacy-aware means of performing real-time feature extraction, since keypoints and not the full image could eventually be streamed out from near the sensor to the cloud.

Our proposed solution includes a three-slot VLIW, with a SIMD vector processor, and one slot including the RV32I RISCV ISA, balancing the contrasts of workload present in the set of sub-algorithms constituting ORB. Although supporting efficiently, one single-scale, our system achieves a 140 FPS at 430 MHz, when extracting 1000 keypoints in a VGA image, drawing 90 mW.

In future work, we plan to efficiently support scale-space generation, feature matching, and to increase performance of already identified bottlenecks regarding VLIW utilization.

Author Contribution All coauthors contributed to this publication.

Funding Open access funding provided by Lund University. This study was funded by the Lunds Tekniska Högskola, Lunds universitet.

Data Availability Data sharing not applicable to this article as no datasets were generated or analysed during the current study.

Declarations

Ethics Approval Not applicable.

Competing Interest The authors have no relevant financial or non-financial interests to disclose.

References

1. Chiang, W. -C., Li, Y., Shang, J., & Urban, T. L. (2019). Impact of drone delivery on sustainability and cost: Realizing the UAV potential through vehicle routing optimization. *Applied Energy, 242*, 1164–1175. https://doi.org/10.1016/j.apenergy.2019.03.117
2. Kanistras, K., et al. (2013). A survey of unmanned aerial vehicles (UAVs) for traffic monitoring. In: 2013 International Conference on Unmanned Aircraft Systems (ICUAS), pp. 221–234. https://doi.org/10.1109/ICUAS.2013.6564694
3. Liu, Y., Liu, H., Tian, Y., & Sun, C. (2020). Reinforcement learning based two-level control framework of UAV swarm for cooperative persistent surveillance in an unknown urban area. *Aerospace Science and Technology*, *98*, 105671. https://doi.org/10.1016/j.ast.2019.105671
4. Erdos, D., Erdos, A., & Watkins, S. E. (2013). An experimental UAV system for search and rescue challenge. *IEEE Aerospace and Electronic Systems Magazine, 28*(5), 32–37. https://doi.org/10.1109/MAES.2013.6516147
5. Candiago, S., et al. (2015). Evaluating multispectral images and vegetation indices for precision farming applications from UAV images. *Remote Sensing, 7*(4), 4026–4047. https://doi.org/10.3390/rs70404026
6. Li, D., Wang, X., & Sun, T. (2016). Energy-optimal coverage path planning on topographic map for environment survey with unmanned aerial vehicles. *Electronics Letters, 52*(9), 699–701. https://doi.org/10.1049/el.2015.4551
7. Franklin, S. E., & Ahmed, O. S. (2018). Deciduous tree species classification using object-based analysis and machine learning with unmanned aerial vehicle multispectral data. *International Journal of Remote Sensing, 39*(15–16), 5236–5245. https://doi.org/10.1080/01431161.2017.1363442
8. Liu, S., et al. (2017). Computer architectures for autonomous driving. *Computer, 50*(8), 18–25. https://doi.org/10.1109/MC.2017.3001256
9. European Union Energy label. Tesla Support. (2021). https://www.tesla.com/en_EU/support/european-union-energy-label
10. Suleiman, A., et al. (2019). Navion: A 2-mw fully integrated real-time visual-inertial odometry accelerator for autonomous navigation of nano drones. *IEEE Journal of Solid-State Circuits, 54*(4), 1106–1119. https://doi.org/10.1109/JSSC.2018.2886342
11. Engel, J., Koltun, V., & Cremers, D. (2018). Direct sparse odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 40*(3), 611–625. https://doi.org/10.1109/TPAMI.2017.2658577
12. Mur-Artal, R., Montiel, J. M. M., & Tardós, J. D. (2015). ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics, 31*(5), 1147–1163. https://doi.org/10.1109/TRO.2015.2463671
13. Klein, G., & Murray, D. (2007). Parallel tracking and mapping for small AR workspaces. In: 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality, pp. 225–234. https://doi.org/10.1109/ISMAR.2007.4538852
14. Newcombe, R. A., Lovegrove, S. J., & Davison, A. J. (2011). DTAM: Dense tracking and mapping in real-time. In: 2011 International Conference on Computer Vision, pp. 2320–2327. https://doi.org/10.1109/ICCV.2011.6126513
15. Engel, J., Schöps, T., & Cremers, D. (2014). LSD-SLAM: Large-scale direct monocular SLAM. In: ECCV.
16. Hartley, R., & Zisserman, A. (2003). Multiple wiew geometry in computer vision, 2nd Edn. Cambridge University Press, New York, NY, USA.
17. Park, S., Schöps, T., & Pollefeys, M. (2017). Illumination change robustness in direct visual SLAM. In: 2017 IEEE International Conference on Robotics and Automation (ICRA), pp. 4523–4530. https://doi.org/10.1109/ICRA.2017.7989525
18. Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision, 60*(2), 91–110. https://doi.org/10.1023/B:VISI.0000029664.99615.94
19. Bay, H., et al. (2008). Speeded-up robust features (SURF). *Computer Vision and Image Understanding, 110*(3), 346–359. https://doi.org/10.1016/j.cviu.2007.09.014
20. Rublee, E., Rabaud, V., Konolige, K., & Bradski, G. (2011). ORB: an efficient alternative to SIFT or SURF. In: 2011 International Conference on Computer Vision, pp. 2564–2571. https://doi.org/10.1109/ICCV.2011.6126544
21. Viswanath, P., Swami, P., Desappan, K., Jain, A., & Pathayapurakkal, A. (2015). ORB in 5 ms: an efficient SIMD friendly implementation. In: Jawahar, C. V., Shan, S. (eds.) *Computer Vision - ACCV 2014 Workshops*, pp. 675–686. Springer, Cham.
22. Fang, W., Zhang, Y., Yu, B., & Liu, S. (2017). FPGA-based orb feature extraction for real-time visual slam. In: 2017 International Conference on Field Programmable Technology (ICFPT), pp. 275–278. https://doi.org/10.1109/FPT.2017.8280159
23. Taranco, R., Arnau, J. -M., González, A. (2021). 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), pp. 11–21. https://doi.org/10.1109/SBAC-PAD53543.2021.00013
24. Ferreira, L., et al. (2021). Energy-efficient application-specific instruction-set processor for feature extraction in smart vision systems. In: 2021 55th Asilomar Conference on Signals, Systems, and Computers, pp. 324–328. https://doi.org/10.1109/IEEECONF53345.2021.9723114
25. Bailo, O., Rameau, F., Joo, K., Park, J., Bogdan, O., & Kweon, I. (2018). Efficient adaptive non-maximal suppression algorithms for homogeneous spatial keypoint distribution. *Pattern Recognition Letters*, *106*. https://doi.org/10.1016/j.patrec.2018.02.020
26. Ferreira, L., et al. (2021). Reconfigurable multi-access pattern vector memory for real-time ORB feature extraction. In: 2021 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1–5. https://doi.org/10.1109/ISCAS51556.2021.9401698
27. Sun, R., et al. (2017). A 42fps full-HD ORB feature extraction accelerator with reduced memory overhead. In: 2017 International Conference on Field Programmable Technology (ICFPT), pp. 183–190. https://doi.org/10.1109/FPT.2017.8280137