An Architecture for Solving the Eigenvalue Problem on Embedded FPGAs

Alwyn Burger[1(✉)], Patrick Urban[1], Jayson Boubin[2], and Gregor Schiele[1]

[1] University of Duisburg-Essen, 47057 Duisburg, Germany
{falwyn.burger,gregor.schiele}@uni-due.de,
patrick.urban@stud.uni-due.de
[2] The Ohio State University, Columbus, OH, USA
boubin.2@osu.edu
http://www.uni-due.de/es, http://www.osu.edu

Abstract. Resource-limited embedded devices like Unmanned Aerial Vehicles (UAVs) often rely on offloading or simplified algorithms. Feature extraction such as Principle Component Analysis (PCA) can reduce transmission data without compromising accuracy, or even be used for applications like facial detection. This involves solving eigenvectors and values which is impractical on conventional embedded MCUs.

We present a novel hardware architecture for embedded FPGAs that performs eigendecomposition using previously unused techniques like squared Givens rotations. That leads to a 3x performance improvement for 16×16 covariance matrices over similar approaches that use much larger FPGAs. Offering higher than 30 fps at only 68.61 µJ per frame, our architecture creates exciting new possibilities for intelligent mobile devices.

Keywords: Hardware architecture · FPGA · Feature extraction

1 Introduction

Eigendecomposition and feature extraction have been the focus of continued research for many years [19,25,26]. Algorithms like principle component analysis (PCA) allow us to simplify a dataset to only its important features by identifying its distinguishing eigenvectors. By projecting data into a reduced eigenspace (the space described by the eigenvectors), we can simplify problems like facial detection and recognition to a comparison of a few eigenvalues, i.e. the relative weight of each eigenvector. More applications of these techniques are being developed, e.g. in the field of convolutional neural networks (CNNs) where PCA can find dominant features and compress network structures [12].

However, PCA's batched nature and computational complexity makes it infeasible for resource-limited devices. In power-limited applications such as Unmanned Aerial Vehicles (UAVs) that rely on camera feeds, feature extraction

A. Brinkmann et al. (Eds.): ARCS 2020, LNCS 12155, pp. 32–43, 2020.
https://doi.org/10.1007/978-3-030-52794-5_3

could offer data size reduction through local preprocessing. Adding the online learning capabilities of incremental PCA (IPCA) further allows the devices to incorporate incoming images into the training set – thereby continuously improving its performance.

To enable this, an accelerator architecture is required that efficiently performs eigendecomposition on an embedded FPGA. This offers improved energy efficiency for small devices over GPUs, and additionally provides flexibility over ASICs as it can be reconfigured to deploy another accelerator at runtime. Delegating this complex computational task to a local FPGA promises considerably improved processing power over doing everything on a MCU.

However, most techniques for doing eigendecomposition such as the QR algorithm [11] strongly depend on trigonometric functions or square roots to compute a Givens rotation matrix [14] which are resource inefficient on such devices. Although alternatives like Squared Givens Rotations (SGR) [9] would be considerably more efficient, they introduce scaling issues and have to the authors' knowledge not been successfully used in the QR algorithm.

In this paper we present a revolutionary hardware architecture design for performing eigenvalue decomposition (EVD) on an embedded FPGA. By using a number of state-of-the-art optimization techniques in a novel way, our system is capable of increasing processing speed by 3–4x over current literature without compromising accuracy.

Our main contributions are

1. a highly resource-optimized computing architecture for solving eigenvalue problems,
2. that is scalable from tiny embedded FPGAs to standard desktop models through a fully homogeneous network of processing elements,
3. and offers pipelined single clock processing elements for maximum processing speed.

We present our solution by looking at related work in Sect. 2, followed by an overview of our solution in Sect. 3. The details of the technical contributions follows in Sect. 4, after which we evaluate our solution in Sect. 5. Finally, we study the application case of UAVs in Sect. 6 and conclude with some final thoughts in Sect. 7.

2 Related Work

Incremental PCA [1,6] is a relatively recent development. It offers us the crucial benefit of online training and avoids the expansion of the covariance matrix as the training dataset is expanded. Conventional eigensolver algorithms have been found to be ill-suited to GPU architectures [18] even though they can achieve nearly 5x speedup over CPUs. QR decomposition (which computes a single iteration of the QR algorithm) specifically has been implemented using different GPU-based accelerator architectures [17,18].

Similar to our approach, Guerrero-Ramírez et. al. [15] presented the first eigensolver based on systolic arrays that implements the QR algorithm using

FPGAs. These arrays describe a network of processing elements, where each partially computes a function and passes to their neighbors. In this case, they iteratively calculate trigonometric functions. Their implementation improved processing time by a factor of 1.17x–1.37x compared to CPU architectures.

A slower solution that includes a full PCA solver was shown by Korat [19]. It uses significantly more FPGA resources than the previously mentioned work, and showed that some of the components such as mean calculation and data normalization are very inefficient on FPGAs.

Ultimately, these authors were limited by having to iteratively approximate trigonometric functions using the COordinate Rotation DIgital Computer (CORDIC) algorithm [21] – causing severe slowdown for more processed bits [23]. Additionally, their resource consumption is impractically high for an embedded FPGA. Other projects that use systolic arrays for QR decompositions on FPGAs [8,27] have similar limitations. To the best of the authors' knowledge our work represents the first FPGA implementation of the QR algorithm using systolic arrays based on an algorithm that does not rely on trigonometric functions.

3 Solution Design

At the core of our EVD (see Fig. 1) is the triangular systolic array (a) to perform QR decomposition. It is composed of two types of nodes: boundary (b) on the diagonal of the triangular matrix and internal (c) off the diagonal. This iteratively computes the eigenvalues and eigenvectors of a provided covariance matrix, entering in a skewed order (d). The QR-array results can be fed back into the system using the buffer (e) until the result converges, at which point the deskewed output (f) is presented. The scaled output of each step of the QR array is down-scaled (g).

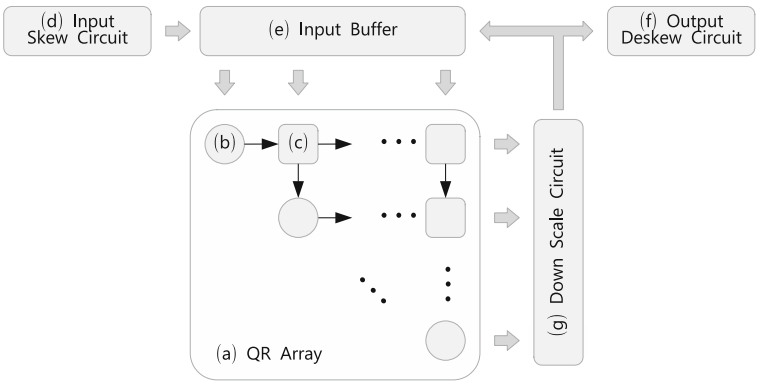

Fig. 1. Parallel triangular systolic array processor to determine the eigenvalues and eigenvectors by calculating the QR decomposition based on SGR in an iterative manner.

The starting point for our solution is the QR decomposition. We first consider a real symmetric matrix A_0 of dimensions $n \times n$, which is the covariance matrix for the PCA to be applied. The rank of this matrix corresponds with the number of eigenvectors being computed, effectively controlling the number of features being extracted. The approximate determination of the eigenvalues and eigenvectors is done with the QR algorithm. It is an iterative application of the QR decomposition, which factorizes a matrix by means of plane rotations, e.g. Givens rotations.

Each QR iteration is given as:

$$[Q_i, R_i] = qrd(A_i) \tag{1}$$

$$A_{i+1} = R_i Q_i \tag{2}$$

and is performed n times over the matrix A until its diagonal elements converge to the eigenvalues. The collection of eigenvectors Q themselves could be determined by calculating the product of all these orthogonal matrices Q_i:

$$Q = \prod_{i=0}^{n} Q_i \tag{3}$$

Using SGR allows us to first use A_i to compute R_i, and even to solve Eq. 1 by processing the identity matrix I to compute Q_i. The orthogonal similarity transformation A_{i+1} follows by processing R_i. Furthermore, the eigenvectors Q in Eq. 3 can be determined efficiently by processing each computed Q_i. As all of these are processed in the same way, we can reuse the processing elements for improved efficiency.

Each iteration in the QR algorithm thus consists of the input sequence $S = \{A, Q, R\}$. The problem remains that all R_i and Q_i are scaled by the SGR, meaning it cannot be directly used for further iterations.

4 Technical Implementation Contributions

Our primary contribution addresses the internal structure of the processing elements in the QR array. We improve upon the latency of current state-of-the-art algorithms by using the square-root-free algorithm proposed by Döhler [9] to avoid the associated latency. It allows our processing elements to have a latency of only one clock cycle.

Although SGR has been used for *QR decomposition*, it has not been applied to the *QR algorithm* due to scaling problems. Since results should be fed through multiple iterations, this would cause overflow errors. To the authors' best knowledge SGR has therefore not been used for EVD using the QR algorithm.

4.1 SGR Result Scaling

The SGR algorithm scales each calculated QR decomposition [9], which means that it cannot be used for the QR algorithm directly. Especially when using fixed-point representation, this will quickly cause overflow.

We found the result to be as shown in Eq. 4, which shows that the eigenvalues λ_i found on the diagonal of $R*$ are squared. Additionally, other values are linearly scaled with the value of λ_i^2. Similarly, each column in $Q*$ is scaled.

$$\underset{\text{A}}{\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ a_{21} & \cdots & a_{2n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix}} \xrightarrow[\text{QR}]{\text{SGR}} \underset{\text{R*}}{\begin{bmatrix} \boldsymbol{\lambda_1^2} & \lambda_1 r_{12} & \cdots & \lambda_1 r_{1n} \\ 0 & \boldsymbol{\lambda_2^2} & \cdots & \lambda_2 r_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \boldsymbol{\lambda_n^2} \end{bmatrix}} \underset{\text{Q*}}{\begin{bmatrix} \lambda_1 q_{11} & \cdots & \lambda_n q_{1n} \\ \lambda_1 q_{21} & \cdots & \lambda_n q_{2n} \\ \vdots & \ddots & \vdots \\ \lambda_1 q_{n1} & \cdots & \lambda_n q_{nn} \end{bmatrix}} \tag{4}$$

A well-known approach for determining reciprocal square roots [10] is given by iteratively solving the Newton Method

$$y_{i+1} = \frac{1}{2}(3y_i - y_i^3 x_{in}) \qquad y_0 = 0.5 \tag{5}$$

until it converges to $y = \frac{1}{\sqrt{x_{in}}}$. However, this can be slow under a bad *initial guess* y_0 very different from the actual result. An interesting approach to this was coined for the video game *Doom*[1], where the initial guess is varied depending on the input value.

Extending on this concept, we have developed a novel way to use lookup-tables (LUTs) for using this with fixed-point numbers. By choosing from a pre-computed set of appropriate y_0 based on the input x_{in}, we can reduce the number of iterations required for convergence. Given a sufficiently large LUT with 128 24-bit entries to create a very accurate initial guess, we can directly solve Eq. 5 in a single iteration.

4.2 Shared Division

Solving EVD using SGR requires two divisions [7,9,20], which for a matrix width of n would result in $\frac{1}{2}(n^2-n)$ dividers. Since they are non-trivial to implement in hardware (particularly the reciprocal of the divisor), this would be very resource-intensive.

Therefore, we studied the schedule of active nodes in the array as shown in Fig. 2. As division is only required in diagonal mode, this shows that only one division occurs per row. This allows us to share the dividers more efficiently, and to reduce the required number to n. For a 16×16 covariance matrix, this leads to a reduction of 104 divider circuits.

4.3 HDL Optimizations

Similarly, large binary multipliers occupy substantial logic resources in FPGAs. One can build a sequential circuit using multiplexers on the inputs that cycles a single multiplier for multiple usages. The basic idea is to first get the result of A * B in a register, then to multiply that by C.

[1] https://github.com/id-Software/DOOM.

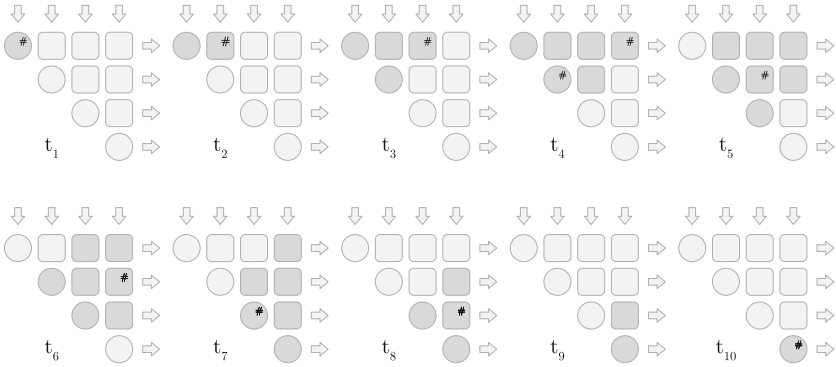

Fig. 2. Propagation through the array at time t_i highlighting the active nodes

Additionally, the DSPs are optimized using a technique called *retiming*, which involves moving registers across combinatorial logic to improve the design performance without affecting the input or output behavior of the circuit [22]. Despite the optimized interconnection in dedicated logic, adder chains used to implement binary multipliers in DPS slices cause delays.

Based on anecdotal evidence, this technique improved our maximum frequency possible from 247.64 MHz to 373.13 MHz. This increase of 50.67% greatly boosts performance, as the worst case slack is greatly improved.

5 Evaluation

Before our approach can be applied to a practical system, we must first evaluate how well it performs. It is aimed at embedded FPGAs that have been shown to be very capable in applications such as small neural networks [5,24]. Not only must we ensure that our design is efficient enough to fit this resource-constrained class of FPGAs, but also that the resulting performance is adequate to offer real-world usability.

5.1 Resource Utilization

Firstly, we consider the resource consumption on the FPGA. As detailed in Sect. 4, the greatest impact on this is through the size of the processed matrix. Larger matrix sizes enable the computation of more eigenvectors at increased complexity, thereby extracting more identifiable features. Therefore, we varied this size in Table 1 and captured the number of resources consumed by each solution.

Note that these results are an absolute number and is valid for the entire 7 series devices from Xilinx, as they are all based on the same architecture. This provides a convenient way to choose the correct FPGA to use for a specific application, based on the limiting hardware resource. For example, the Spartan 7 range varies in available DSP slices from 10 on the S6 to 160 on the S100. It

Table 1. Synthesis results for Xilinx-7 series FPGAs in absolute numbers

Matrix width	Logic cells	Flip-flops	DSP slices
4×4	3,940	1,497	15
8×8	11,616	5,548	45
16×16	36,165	21,612	153

also shows that the implemented homogeneous architecture is easily adaptable to larger-scale deployment, as a larger FPGA could simply support a larger matrix and thereby enable larger inputs and more complex applications.

To put these numbers in context, we compare them to the most recently published CORDIC-based eigensolvers [15,19] in Fig. 3. We consider specifically the logic cells and DSP slices, as these are commonly the limiting factors.

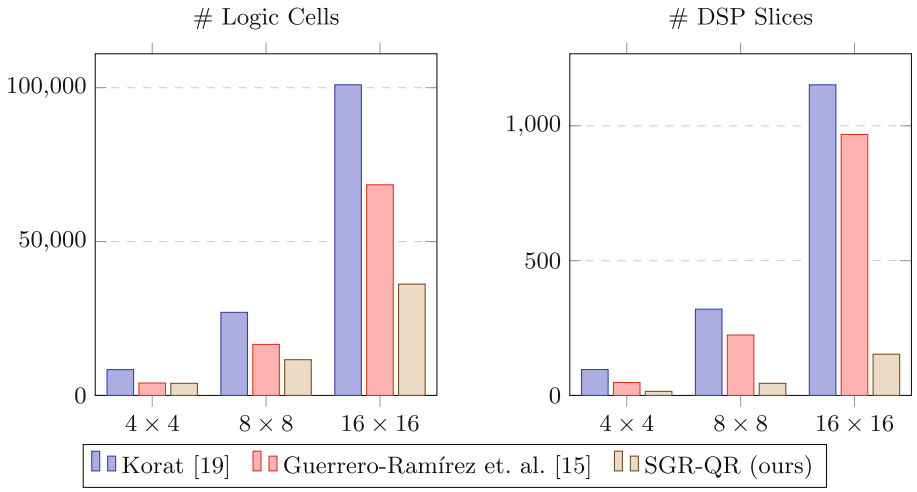

Fig. 3. Comparison of the resource utilization of different matrix sizes with related work

Omitting the additional logic required by a CORDIC-based approach significantly improves our resource consumption, as almost half of the logic cells are saved. More importantly, the number of DSP blocks are reduced by almost 85%. This allows us to use FPGAs with significantly fewer resources, or to support a larger covariance matrix.

5.2 Throughput

Before the system's throughput rate can be calculated, the maximum operating frequency f_{max} must be determined using a static time analysis. Table 2 lists the

maximum possible clock rates for all targets as the matrix size is varied. As the other solution is not open source, only the clock frequencies achievable in [15] are provided for comparison. Unsurprisingly, the maximum clock rate at which the implemented design can be operated decreases with increasing logic density.

Table 2. Maximum operating frequencies in [MHz] depending on the matrix width

Target	f_{max} matrix width		
	4×4	8×8	16×16
XC7S100	239.01	228.31	219.11
XC7A100	265.75	237.87	237.98
XC7K70	339.90	272.18	252.46
EP4SGX230 [15]	235.32	220.15	201.35

To determine the throughput rate, the combined latency of the processing elements must be considered. Each has a latency of $p = 6$ clock cycles. The number of iterations to be performed is set to $k = 30$ for a direct comparison with related work.

Firstly, the latency of initially filling the FIFO buffers is $L_{FIFO} = 3n - 1$ cycles, where n is again the matrix width. Each of the QR iterations requires $L_{QR} = 24n - 6$ while the inverse square root consumes a constant $L_{Sqrt} = 12$ clock cycles.

This leads to a model of the overall latency L and throughput T of

$$L(n,k) = L_{FIFO} + k \cdot (L_{QR} + L_{Sqrt}) \tag{6}$$

$$T(n,k) = \frac{f_{max}}{L(n,k)} = \frac{f_{max}}{24nk + 3n + 6k - 1} \text{ solutions}/s \tag{7}$$

where each solution refers to a complete calculation of all eigenvalues and -vectors [15]. The maximum operating frequency f_{max} results from the static timing analysis results shown in Table 2.

Figure 4 compares the throughput of our approach to a CORDIC-based approach [15] and a desktop CPU. The SGR-QR was implemented on a Xilinx Spartan-7 XC7S100, and a fixed point representation of 24 bits was chosen to match the input signals in each DSP48 block. Note that the frequency of the memory is assumed to be at least as fast as the main clock f_{max}.

The SGR-QR is faster than the CORDIC-based approach implemented on the considerably larger Virtex-7 (3.81x for 4×4 to 4.26x for 16×16 matrices). The benefits of our highly parallel architecture over higher clocked CPUs become particularly evident for larger matrices. This is due to our approach's linear runtime, while CPU implementations are commonly $O(N^3)$ and single-threaded.

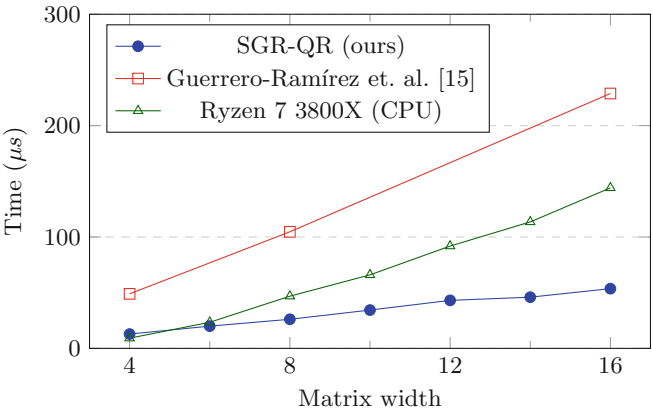

Fig. 4. Time in µs required to compute a single eigenpair of different matrix sizes

5.3 Estimated Power Usage

Using the maximum clock frequency from Table 2, the implementation results for a number of embedded FPGAs from the Xilinx-7 Family are shown in Table 3.

Table 3. Implementation results for matrix size 16×16

Target	LUT	FF	DSP	Power [W]
Spartan-7 XC7S100	49%	17%	96%	1.402
Artix-7 XC7A100	49%	17%	64%	1.379
Artix-7 XC7A200	23%	8%	21%	1.238
Kintex-7 XC7K70	76%	26%	64%	1.425
Kintex-7 XC7K160	31%	11%	26%	1.214

Apart from the proportional resource consumption for a number of devices, the estimated power usage is also provided by the Vivado software of Xilinx. This is the active consumption of the device, highlighting the importance of processing speed to offset the cost of keeping the FPGA powered.

6 Application Case Study

Our system is designed with high energy and resource efficiency in mind in order to support the small, battery-powered devices used in many pervasive or organic computing applications. One example is a fully autonomous aerial system (FAAS) that combines unmanned aerial vehicles (UAV), edge computers, and data centers to create intelligent systems. They should autonomously explore their environment and accomplish high level goals without human intervention [3], which requires expensive techniques such as facial detection.

UAVs typically only carry small batteries with flight times between 15 and 25 min and therefore rely on offloading tasks to edge and cloud systems [4]. Transferring images between edge and UAV is costly, taking on the order of seconds in prior work [4]. Prior work on micro aerial vehicles with in-situ vision systems performed detections locally on UAV. Increased frame rates and decreased power-consumption were achieved by downsampling (5–12 fps) and compressing incredibly small images (17 fps) to be used as input to neural networks [2,13]. In aerial applications this can lead to loss of critical information contained in small regions. Instead, our system can be used as a local facial detection algorithm or as preprocessing to reduce offloaded data to only the important features.

Therefore, we evaluated our architecture design using the well-known FDDB dataset [16]. A sliding window of 250×250 pixels is moved over an input image of resolution 640×480. The covariance matrix varies with the number of training images from 4 to 16 faces. For this dataset, 95% of the variance could be described with 62.5% of vectors – offering substantial data reductions. Processing speed of an EVD on the Spartan 7 S100 for different size covariance matrices are presented in Fig. 5. Using a naive classifier, increasing the matrix size from 4×4 to 16×16 increased the accuracy from 44.6% to 55.5% (in line with similar approaches [16]).

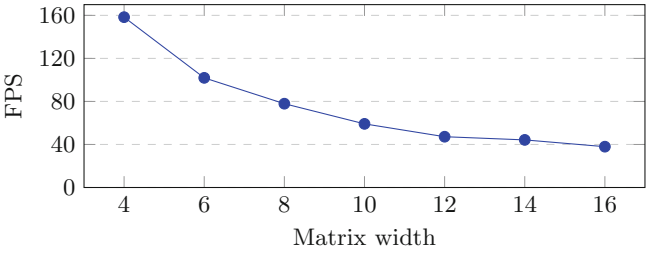

Fig. 5. Frames per second for facial detection application

The speed is reduced for larger matrices, but even at 16×16 the performance remains above 30fps. This shows the trade-off between speed and complexity, which can be combined with Table 3 to tailor the hardware choice. Each device's power usage allows us to estimate the energy usage per frame to between 3.14 µJ for $n = 4$ and 68.61 µJ for $n = 16$. Although related work does not provide this information, we are confident that our system is more energy efficient, as transmitting even an image preview (720×900) can take a UAV 1.4 s [4].

7 Conclusion and Future Work

We presented our approach for EVD on an embedded FPGA. Through optimizations like systolic arrays and dynamically scaling SGR results, we achieved an improvement of 3x performance over other approaches. Additionally, the architecture is resource optimized enough to be used even on small embedded FPGAs like a Xilinx Spartan 7.

In future work, we hope to implement this onto a set of drones augmented with FPGAs for real-world experiments. We also plan to investigate using this feature extraction method as a preprocessor for CNNs. By using the reconfigurability of the FPGA, we can switch between EVD to perform a learning feature extraction on incoming data followed by a neural network. This provides processing complexity heretofore impractical on embedded devices used in organic computing applications.

Acknowledgements. The authors acknowledge the financial support by the Federal Ministry of Education and Research of Germany in the KI-Sprung LUTNet project (project number 16ES1125).

References

1. Artac, M., Jogan, M., Leonardis, A.: Incremental PCA for on-line visual learning and recognition. In: Object Recognition Supported by User Interaction for Service Robots, vol. 3, pp. 781–784. IEEE (2002). https://doi.org/10.1109/icpr.2002.1048133
2. Boroujerdian, B., Genc, H., Krishnan, S., Cui, W., Faust, A., Reddi, V.: MAVBench: micro aerial vehicle benchmarking. In: 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pp. 894–907. IEEE (2018). https://doi.org/10.1109/MICRO.2018.00077
3. Boubin, J., Chumley, J., Stewart, C., Khanal, S.: Autonomic computing challenges in fully autonomous precision agriculture. In: IEEE International Conference on Autonomic Computing (ICAC), pp. 11–17 (2019). https://doi.org/10.1109/ICAC.2019.00012
4. Boubin, J.G., Babu, N.T., Stewart, C., Chumley, J., Zhang, S.: Managing edge resources for fully autonomous aerial systems. In: Proceedings of the 4th ACM/IEEE Symposium on Edge Computing, pp. 74–87. ACM (2019). https://doi.org/10.1145/3318216.3363306
5. Burger, A., Qian, C., Schiele, G., Helms, D.: An embedded CNN implementation for on-device ECG analysis. In: IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops) (2020)
6. Cardot, H., Degras, D.: Online principal component analysis in high dimension: which algorithm to choose? arXiv preprint arXiv:1511.03688 (2015)
7. Cerato, B., Masera, G., Viterbo, E.: Enabling VLSI processing blocks for MIMO-OFDM communications. VLSI Design **2**, 11 (2008). https://doi.org/10.1155/2008/351962
8. Chen, D., Sima, M.: Fixed-point CORDIC-based QR decomposition by Givens rotations on FPGA. In: International Conference on Reconfigurable Computing and FPGAs, pp. 327–332. IEEE (2011). https://doi.org/10.1109/ReConFig.2011.38
9. Döhler, R.: Squared givens rotation. IMA J. Numer. Anal. **11**(1), 1–5 (1991). https://doi.org/10.1093/imanum/11.1.1
10. Ercegovac, M.D., Lang, T., Muller, J.M., Tisserand, A.: Reciprocation, square root, inverse square root, and some elementary functions using small multipliers. IEEE Trans. Comput. **49**(7), 628–637 (2000). https://doi.org/10.1109/12.863031

11. Francis, J.G.: The QR transformation a unitary analogue to the LR transformation–Part 1. Comput. J. **4**(3), 265–271 (1961). https://doi.org/10.1093/comjnl/4.3.265
12. Garg, I., Panda, P., Roy, K.: A low effort approach to structured CNN design using PCA. IEEE Access **8**, 1347–1360 (2019). https://doi.org/10.1109/ACCESS.2019.2961960
13. Genc, H., Zu, Y., Chin, T.W., Halpern, M., Reddi, V.J.: Flying IoT: toward low-power vision in the sky. IEEE Micro **37**(6), 40–51 (2017). https://doi.org/10.1109/MM.2017.4241339
14. Golub, G.H., Van Loan, C.: Matrix Computations, 4th edn. The Johns Hopkins University Press, Baltimore (2013)
15. Guerrero-Ramírez, J.E., Velasco-Medina, J., Arce, J.C.: Hardware design of an eigensolver based on the QR method. Analog Integr. Circ. Sig. Process **82**(1), 125–134 (2014). https://doi.org/10.1109/LASCAS.2013.6519065
16. Jain, V., Learned-Miller, E.: FDDB: a benchmark for face detection in unconstrained settings. Technical report. UM-CS-2010-009, University of Massachusetts, Amherst (2010)
17. Johansen, T.A.H.: On the improvement and acceleration of eigenvalue decomposition in spectral methods using GPUs. Master's thesis, UiT Norges arktiske universitet (2016)
18. Kerr, A., Campbell, D., Richards, M.: QR decomposition on GPUs. In: Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units, pp. 71–78. ACM (2009). https://doi.org/10.1145/1513895.1513904
19. Korat, U.A., Alimohammad, A.: A reconfigurable hardware architecture for principal component analysis. Circuits Syst. Signal Process. **38**(5), 2097–2113 (2018). https://doi.org/10.1007/s00034-018-0953-y
20. Ma, L., Dickson, K., McAllister, J., McCanny, J.: MSGR-based low latency complex matrix inversion architecture. In: 9th International Conference on Signal Processing, pp. 410–413. IEEE (2008). https://doi.org/10.1109/ICOSP.2008.4697158
21. Meher, P.K., Valls, J., Juang, T.B., Sridharan, K., Maharatna, K.: 50 years of cordic: algorithms, architectures, and applications. IEEE Trans. Circuits Syst. I Regul. Pap. **56**(9), 1893–1907 (2009). https://doi.org/10.1109/TCSI.2009.2025803
22. Pan, P., Lin, C.C.: A new retiming-based technology mapping algorithm for LUT-based FPGAs. In: Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays, pp. 35–42. ACM (1998). https://doi.org/10.1145/275107.275118
23. Ren, M.: Cordic-based Givens QR decomposition for MIMO detectors. Ph.D. thesis, Georgia Institute of Technology (2013)
24. Schiele, G., Burger, A., Cichiwskyj, C.: The elastic node: an experimentation platform for hardware accelerator research in the internet of things. In: Proceedings of the IEEE International Conference on Autonomic Computing, ICAC, pp. 84–94 (2019). https://doi.org/10.1109/ICAC.2019.00020
25. Sorzano, C.O.S., Vargas, J., Montano, A.P.: A survey of dimensionality reduction techniques. arXiv preprint arXiv:1403.2877 (2014)
26. Turk, M.A., Pentland, A.P.: Face recognition using eigenfaces. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 586–591 (1991). https://doi.org/10.5120/20740-3119
27. Yu, H.: FPGA-based implementation of QR decomposition. Master's thesis, Arizona State University (2014)