Dual-Triangular QR Decomposition with Global Acceleration and Partially Q-Rotation Skipping

Rui Fang, Siyang Jiang, Hsi-Wen Chen, Wei Ding, Ming-Syan Chen✉
Department of Electrical Engineering, National Taiwan University
{rfang, syjiang, hwchen, wding}@arbor.ee.ntu.edu.tw, mschen@ntu.edu.tw

***Abstract*—Efficient matrix operations have been deemed keys to efficient data analysis. Dual-Triangular QR Decomposition (DT-QRD) is a critical component in Tall and skinny QR decomposition (TS-QRD), which is a widely-used matrix operation with various applications, such as data compression and feature extraction. In order to accelerate DT-QRD, in this paper, we propose a new acceleration framework, including *Global Acceleration Schemes*, and *Partially Q-rotation Skipping*, which utilize the special DT structure in both Q and R matrix to reduce the latency and computation resource. Further, we employ the Systolic-Array Based Architecture (1D & 2D) for implementation to reduce the memory usage. Experimental results manifest that our framework achieves $169.70\times$ (1D) and $250.13\times$ (2D) speedup.**

***Index Terms*—Dual-triangular QR decomposition, High-Level Synthesis, Systolic Array**

I. Introduction

Tall and skinny QR Decomposition (TS-QRD), which factorizes a matrix into an orthogonal matrix Q and an upper triangular matrix R, covers various applications in the numerical analysis [9, 14]. For instance, a video streaming system generates thousands of pixels in a frame (represented by rows) and tens of frames (represented by columns). Several works [5] have introduced different acceleration schemes to typically accelerate the TS-QRD on powerful computation platforms, e.g., GPU. However, such accelerations on edge devices [11], e.g., FPGAs, are much more challenging due to the limitations on both storage and computational resource [1, 6, 15, 18]. Zhang *et al.* [18] have shown that directly employing TS-QRD decomposition on FPGAs may lead to unacceptable performance. To effectively decompose a TS-QRD matrix. Aslan *et al.* [1] proposed a fast QR decomposition method using a folded systolic array and the *CORDIC* algorithm. Rafique et al. [12] scheduled the data traffic and the computing time using Household QR decomposition on edge devices. Wang et al. [16] proposed a unified architecture with Householder reflection and Givens rotation.

Indeed, the TS-QRD accelerations schemes consist of two primary operations, i.e., *Normal QR Decomposition* and *Dual-Triangular QR Decomposition (DT-QRD)* [2, 7]. To speed up the Normal QR Decomposition, Desai et al. [4] and Tan *et al.* [13] accelerated Normal QR Decomposition via the Gram-Schmidt method and the modified Gram-Schmidt method, respectively. In addition, Givens rotation is deployed [11] by combining the *CORDIC* and the systolic array. Liu *et al.* [8] accelerated the QR decomposition by High-Level Synthesis. To accelerate DT-QRD, Jiang *et al.* [7] propose a systolic-array-based accelerator using the Dual-triangular structure. Still, they only notice the DT structure in the R matrix, ignoring the special form in the Q matrix.

In this paper, we deeply investigate the DT structure in submatrices (both matrix Q and R) and propose a simple yet efficient acceleration strategy, namely, *Global Acceleration Schemes (QS)* and *Partially Rotation Skipping (RS)*, to reduce latency under limited computational resources. Then, we design two systolic-array-based accelerators (1D & 2D) to reduce latency by High-Level Synthesis. Experiments demonstrate that our accelerator achieves $169.70\times$ (1D) and $250.13\times$ (2D) speedup on average.

II. Methodology

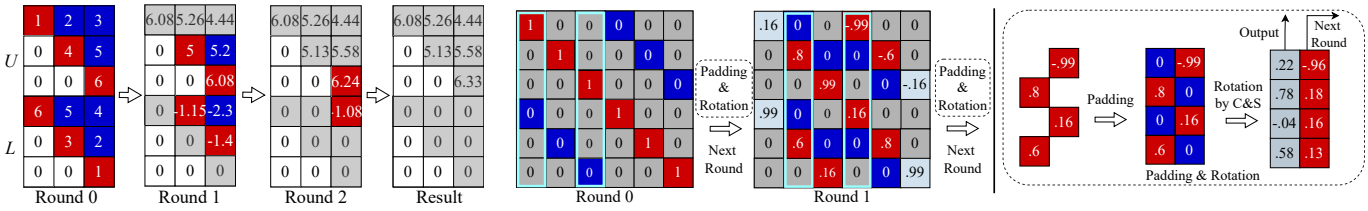
Preliminaries. A denotes a matrix, a_i represents the i^{th} columns of matrix A, and $a_{i,j}$ represents the element in the i^{th} row, the j^{th} columns of A. DT-QRD aims to factor the target DT matrix $A \in \mathbb{R}^{2n\times n}$, into an orthogonal matrix $Q \in \mathbb{R}^{2n\times 2n}$ and upper triangular matrix $R \in \mathbb{R}^{2n\times n}$, i.e., $A = QR$. The matrix A can be divided into two upper triangular matrices: the upper matrix $U \in \mathbb{R}^{n\times n}$ and the lower matrix $L \in \mathbb{R}^{n\times n}$. In real applications, we only have to use the upper part of matrix R (the rotation result of U) as the entries in the lower part of R are zeros.

Specifically, we employ Givens rotation [7] for our baseline method. In addition, compared with other QR decomposition methods, such as Gram-Schmidt [4] and Household reflection [10], Givens rotation is especially suitable for sparse matrix (like DT matrix) [7] because of its high parallelism and great numerical stability [7] with two steps, i.e., *rotation parameters generation* and *rotation*. Specifically, the Givens rotation QR decomposition computes R and the corresponding Q from the DT matrix A by applying a sequence of rotations G_i to eliminate all the non-zero elements by t rounds.

$$R = (G_t G_{t-1} \cdots G_1)A, \text{ and } Q = G_1^T G_2^T \cdots G_t^T$$

where G_i^T is the transpose matrix $G_i \in \mathbb{R}^{2n\times 2n}$.

Global Acceleration Schemes (GS). As shown in Fig. 1(a), our workflow is to compute the diagonal elements in parallel and then obtain the result using pipeline. In particular, we reorganize the upper matrix U and bottom matrix L for data prepossessing, namely the diagonal part (the red boxes) and the non-diagonal part (the blue boxes). Because of the DT structure, rotation parameter generation only happens in the



(a) (b)

Fig. 1: (a) Illustrative Example. (b) Partially Q-Rotation Skipping.

Algorithm 1: Recursive DT-QRD Accelerations

Input: Upper Triangular matrix $U \in \mathbb{R}^{n\times n}$, $L \in \mathbb{R}^{n\times n}$
Output: Unitary matrix $Q \in \mathbb{R}^{2n\times 2n}$; Upper Triangular matrix $U \in \mathbb{R}^{n\times n}$;

/* **Apply *GS* */**
$Q = I^{2n\times 2n}$;
for *k = 0 to n-1* **do**
  for *i = k to n-1* **do**
    $norm = sqrt(u_{i,i}^2 + l_{i-k,i}^2)$;
    $c_i = u_{i,i}/norm; \quad s_i = l_{i-k,i}/norm;$
    $u_{i,i} = norm;$
  for *i = 0 to n-k-1* **do**
    for *j = i+1 to n-k-1* **do**
      $u_{i+k,j+k} = c_{i+k}u_{i+k,j+k} + s_{i+k}l_{i,j+k};$
      $l_{i,j+k} = -s_{i+k}u_{i+k,j+k} + c_i l_{i,j+k};$
    for *j = 0 to 2n-1* **do**
      /* **Apply *QS* */**
      if $j<i$ *or* $j>i+k$ **then**
        continue
      else
        $q_{j,i+k} = c_{i+k}q_{j,k+i} + s_{i+k}q_{j,i+n};$
        $q_{j,i+n} = -s_{i+k}q_{j,k+i} + c_{i+k}q_{j,i+n}$

diagonal section. As a result, we can reduce access time by allocating the diagonal part adjacently in memory. Next, the parallel rotation parameters generation is introduced, which produces the rotation parameters in parallel. Following that, pipeline rotation shortens the execution time by overlapping distinct processes. In addition, dataflow, a task-level pipeline, is employed. Specifically, the generation of rotation parameters and pipeline rotation can be initiated concurrently across different elimination rounds, overlapping processing time.

Partially Q-rotation Skipping (QS). Due to the particular structure of the DT matrix, we propose the *Partially Q-Rotation Skipping (QS)*, which aims to accelerate the computing matrix Q in DT-QRD. Since the matrix Q in DT-QRD is sparse regularly, i.e., the majority of elements are filled with 0, we only need to calculate a portion of the Q matrix in each round and send the non-zero data to the following round. Furthermore, to acquire the rotated matrix Q, zero padding is utilized as an alternative way to rotate the Q matrix rather than transmitting zero data from the previous round. In the left part of Fig. 1(b), due to the elements in grey boxes being zeros, we do not transmit those data to the next rounds to save the computation resource and time. In the right part of Fig. 1(b), the data calculated in each round (in red boxes) are selected by columns (in light blue boxes) and padded within zeros (blue boxes). After rotation, the data are sent to the next round. Note that the Q matrix appears in a predictable sparse form when the rotation pairs' positions are gradually moved toward the center of the DT matrix. Since *QS* avoids a lot of redundant computation, it is simple to conclude that the time complexity of the rotation matrix Q is reduced from $O(n^2)$ to $O(n)$. The above optimization scheme significantly reduces the computation overhead in calculation and accelerates the entire Q rotation process, especially in large-scale matrices.

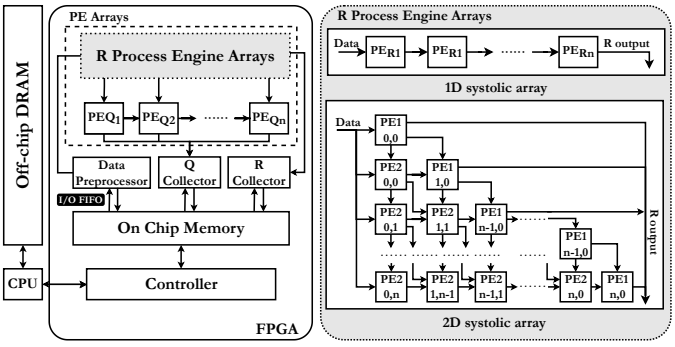


Fig. 2: Architecture Overview.

III. Implementation

Architecture Overview. Fig. 2 illustrates the overall architecture of our DT-QRD accelerator. Firstly, data is transferred from off-chip memory to on-chip memory (BRAM) using direct memory access. Next, after data prepossessing, the data stream is transferred into the sequence of R processing engine arrays for computing matrix R, and PE_{Qi} for matrix Q. Q Collector and R Collector is used for collecting the matrix Q and R from each PE_{Qi} and PE_{Ri}, respectively.

Systolic-Array Based Architecture. In R process engine arrays (in Fig. 2), we adopt the 1D & 2D systolic arrays (SA) to chain the sequence of PE_{Ri} and PE_{Qi}. In the 1D systolic array, we only have one type of PE_{Ri}, and it needs to take responsibility for the rotation parameter generation and rotation. In 2D systolic array, to obtain higher parallelism, we separate the PE_{Ri} into rotation parameter generation PE, i.e., PE_1, and the rotation PE, i.e., PE_2. Cooperating to dataflow, such splitting is efficient since the rotation data are passed to the next calculation rather than waiting for the parameter generation. The input matrix is partitioned each diagonal and

TABLE I: Comparison with previous work in same platform and frequency and OOM means out of memory.

Method		Metrics	Dual 4×4	Dual 8×8	Dual 12×12	Dual 16×16	Dual 24×24
OM [3]	i7-9700K	Speed Ratio	4.39x	6.21x	14.87x	11.19x	19.03x
	ZCU102	Cycle	3705	28337	119589	204558	1023559
		Speed Ratio	26.65x	103.42x	233.57x	251.3x	635.75x
		LUTs	1%	1%	1%	1%	1%
		FF	∼0%	∼0%	∼0%	2%	∼0%
		DSP48E	∼0%	∼0%	∼0%	∼0%	∼0%
		BRAM	0%	0%	0%	0%	0%
HLS LIB [17]		Cycle	1852	12006	40007	83208	347227
		Speed Ratio	13.32x	43.82x	78.14x	102.22x	215.67x
		LUTs	3%	3%	3%	3%	3%
		FF	2%	2%	2%	2%	2%
		DSP48E	1%	1%	1%	1%	1%
		BRAM	0%	0%	1%	2%	3%
1D SA [8]		Cycle	1723	8264	28247	55590	142704
		Speed Ratio	12.4x	30.16x	55.17x	68.29x	88.64x
		LUTs	8%	18%	30%	36%	46%
		FF	2%	5%	9%	11%	23%
		DSP48E	3%	5%	8%	10%	21%
		BRAM	3%	7%	11%	14%	73%
2D SA [8]		Cycle	709	3239	8806	OOM	OOM
		Speed Ratio	5.12x	11.82x	17.2x	OOM	OOM
		LUTs	11%	33%	87%	OOM	OOM
		FF	4%	13%	56%	OOM	OOM
		DSP48E	8%	30%	19%	OOM	OOM
		BRAM	14%	49%	44%	OOM	OOM
DT-SA [7]		Cycle	196	626	2387	4083	14498
		Speed Ratio	1.41x	2.28x	4.66x	5.02x	9.01x
		LUTs	8%	18%	28%	38%	62%
		FF	3%	7%	11%	16%	26%
		DSP48E	4%	9%	14%	20%	29%
		BRAM	8%	18%	28%	38%	65%
Ours 1D		Cycle	175	417	768	1141	2412
		Speed Ratio	1.26x	1.52x	1.51x	1.40x	1.51x
		LUTs	8%	10%	29%	39%	61%
		FF	3%	8%	20%	16%	26%
		DSP48E	5%	10%	14%	20%	30%
		BRAM	8%	16%	25%	37%	53%
Ours 2D		Cycle	139	274	512	814	1610
		Speed Ratio	1.0x	1.0x	1.0x	1.0x	1.0x
		LUTs	7%	19%	33%	52%	87%
		FF	3%	10%	20%	32%	79%
		DSP48E	5%	13%	25%	40%	64%
		BRAM	3%	11%	24%	40%	98%

transferred from BRAM to both PE_1s and PE_2s parallelly. Interim results from PE_1 are simultaneously sent to PE_2s so that they can process the R rotation at once, leading to a higher parallelism. To obtain a lower calculation interval, outputs from PE_2s are directly transferred to the next PE_2s. Note that compared to the 1D structure, the 2D structure also improves the dataflow efficiency since PEs of the 1D structure cause idleness due to the data dependencies between the generation and rotation. Nevertheless, PE_{Q_i} is used for rotating the matrix Q. Note that *GS* is deployed across PE_{Ri} and PE_{Qi} for decreasing the latency, i.e., once parameters are generated in PE_{Ri}, the rotation is launched.

IV. Experiments

Experiment Setup. To evaluate the performance of our accelerators, we choose five different methods as our baselines. i) *Original Method (OM)* [3] is a classical QR decomposition method, implemented on FPGA and CPU (i7-9700K). ii) *HLS LIB* is from the standard linear algebra library by Xilinx. iii) *1D* SA[8] and iv) *2D SA*[8] are general QR Decomposition methods based on systolic arrays. v) *DT-SA* [7] is an acceleration method for DT-QRD with the 1D systolic array. The speed ratio is calculated by dividing the value of five baselines by *Ours 2D*. All experiments are implemented on an Zynq-ZCU102 FPGA with a 100MHz clock frequency and 16-bit fixed point. [1]

Quantitative Evaluation. The result of our method compared to the other five baselines is shown in Table I. As can be seen that our method (*Ours 2D*) outperforms OM with 250.13× and 11.13× respectively on FPGA and CPU, and speeds up

[1]Our code is released in https://github.com/Rui-Fang/DTQR-HLS.git.

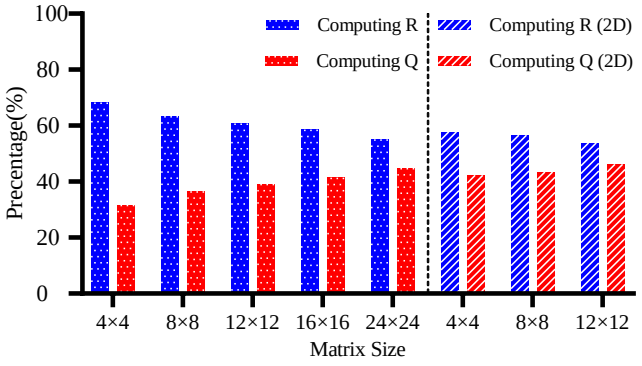


Fig. 3: Bottleneck of Computing Time.

HLS LIB by $90.63\times$ on average. In addition, our approach achieved $50.93\times$ speedups compared to 1D SA [8], with similar resource usage. Meanwhile, we significantly save resource utilization compared to 2D SA with $11.37\times$ speedups, demonstrating the effectiveness of our frameworks. Note that several cases in 2D SA (i.e., 16, and 24) are out of memory (OOM) due to the limited resources on FPGA, which also manifest the efficiency of our frameworks. Compared to the state-of-the-art method, i.e., DT-SA [7], Ours 2D outperforms $4.47\times$ on average because of the benefit of *Partially Q-Rotation Skipping* with skipping redundant computations. Note that our method effectively reduces more latency in large matrices, because *QS* skips more zero elements in large Q matrix, leading a latency and storage savings. Lastly, compared to *our 1D* accelerator, we observe that the 2D accelerator is faster than the 1D one by $1.44\times$ on average. It is reasonable since the 2D accelerator has a higher parallel degree with consuming more computation resources in the same matrix size. In addition, the 2D systolic array structure significantly decreases latency by increasing parallelism and decreasing the dataflow interval between PEs, with an acceptable resource increase. An interesting finding is that, compared to 2D SA, which usually consumes a large amount of resources (OOM in size 16, 24), it demonstrates that our accelerators effectively utilize the DT structure to speed up the DT-QRD.

Ablation Study and Bottleneck Analysis. Table II presents the ablation studies of three variances of our model i) *GS*, ii) *GS+QS*, iii) *GS+QS+2D*. All accelerating strategies improve the efficiency when the matrix dimension increases, demonstrating the scalability of our method. While only using *GS* scheme has relatively weak performance due to the overhead in data traffic and blocking data flowing to the next round without other methods, i.e., *QS* or *2D* significantly boosts efficiency. In terms of large matrix acceleration, we discovered that *QS* outperforms method *2D*. It reveals that computational overhead is the principal factor that causes delays in large DT-QRD. From Fig. 3, we observe that the bottleneck is in computing R in both 1D and 2D systolic accelerators. Meanwhile, the bottleneck of computing Q increases with the size of the matrix. The pipeline rotation and dataflow accelerating strategy can only effectively overlap the computation time but not decrease the computational time of Q.

TABLE II: Ablation study (*OM as the baseline*)

Matrix Size	Dual 4×4	Dual 8×8	Dual 12×12	Dual 16×16	Dual 24×24
GS	18.9x	45.27x	50.10x	53.90x	70.60x
GS+QS	21.17x	67.95x	155.71x	179.28x	424.36x
GS+QS+2D	**26.65x**	**103.42x**	**233.57x**	**251.3x**	**635.75x**

V. Conclusion

In this paper, we deeply investigate the DT structure both on matrix Q and R, and propose the systolic array based DT-QRD accelerators (1D & 2D) on FPGAs. Experimental results show that our accelerators achieve $169.70\times$ (1D) and $250.13\times$ (2D) on average speedup compared with OM, respectively. Our work is applicable to resource-limited computation, such as that needed for edge intelligence.

Acknowledgement

The work is in part supported by MOST Project No. 111-2221-E-002 -135 -MY3, Taiwan.

References

[1] Semih Aslan, Sufeng Niu, and Jafar Saniie. FPGA implementation of fast QR decomposition based on givens rotation. In *MWSCAS*, 2012.
[2] Nai-Yun Cheng and Ming-Syan Chen. Exploring dual-triangular structure for efficient r-initiated tall-skinny QR on GPGPU. In *PAKDD*, 2019.
[3] James W Demmel. *Applied numerical linear algebra*. SIAM, 1997.
[4] Parth Desai, Semih Aslan, and Jafar Saniie. FPGA implementation of Gram-Schmidt QR decomposition using high level synthesis. In *IEEE EIT*, 2017.
[5] Xinyu Hao, Shuangming Yang, Bin Deng, Jiang Wang, Xile Wei, and Yanqiu Che. A CORDIC based real-time implementation and analysis of a respiratory central pattern generator. *Neurocomputing*, 2021.
[6] Javier Hormigo and Sergio D Muñoz. Efficient floating-point givens rotation unit. *Circuits, Systems, and Signal Processing*, 2020.
[7] Siyang Jiang, Hsi-Wen Chen, and Ming-Syan Chen. Dataflow systolic array implementations of exploring dual-triangular structure in QR decomposition using high-level synthesis. In *IEEE ICFPT*, 2021.
[8] Jie Liu and Jason Cong. Dataflow systolic array implementations of matrix decomposition using high level synthesis. In *ACM/SIGDA FPGA*, 2019.
[9] Tsung-Hsien Liu, Yi-Kuang Ko, Yen-Ju Chiu, Wen-Yen Lin, and Yuan-Sun Chu. Hardware implementation of the preprocessing QR-decomposition for the soft-output MIMO detection with multiple tree traversals. *IEEE TCASII*, 2017.
[10] Per-Gunnar Martinsson, Gregorio Quintana OrtI, Nathan Heavner, and Robert van de Geijn. Householder QR factorization with randomization for column pivoting (hqrrp). *SJSC*, 2017.
[11] Sergio D Muñoz and Javier Hormigo. High-throughput FPGA implementation of QR decomposition. *IEEE TCS*, 2015.
[12] Abid Rafique, Nachiket Kapre, and George A Constantinides. Enhancing performance of Tall-Skinny QR factorization using fpgas. In *IEEE FPL*, 2012.
[13] Chong Yeam Tan, Chia Yee Ooi, and Nordinah Ismail. Loop optimizations of mgs-qrd algorithm for fpga high-level synthesis. In *IEEE SOCC*, 2019.
[14] Andrés E Tomás and Enrique S Quintana-Ortí. Cholesky and gram-schmidt orthogonalization for tall-and-skinny qr factorizations on graphics processors. In *ECPP*, 2019.
[15] Stylianos I Venieris, Grigorios Mingas, and Christos-Savvas Bouganis. Towards heterogeneous solvers for large-scale linear systems. In *IEEE FPL*, 2015.
[16] Xinying Wang, Phillip Jones, and Joseph Zambreno. A reconfigurable architecture for QR decomposition using a hybrid approach. In *IEEE ISVLSI*, 2014.
[17] VH Xilinx. Vivado design suite user guide-high-level synthesis, 2014.
[18] Shaoshuai Zhang, Elaheh Baharlouei, and Panruo Wu. High accuracy matrix computations on neural engines: A study of qr factorization and its applications. In *ACM HPDC*, 2020.