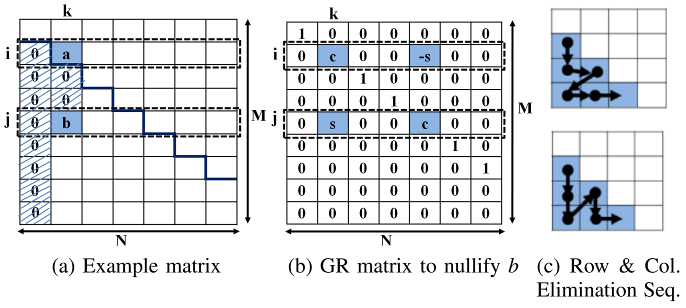
Energy Efficient Data Flow Transformation for Givens Rotation based QR Decomposition

Namita Sharma*, Preeti Ranjan Panda*, Min Li†, Prashant Agrawal†, and Francky Catthoor†
*Indian Institute of Technology Delhi, New Delhi, India
†IMEC, Leuven, Belgium

***Abstract*—QR Decomposition (QRD) is a typical matrix decomposition algorithm that shares many common features with other algorithms such as LU and Cholesky decomposition. The principle can be realized in a large number of valid processing sequences that differ significantly in the number of memory accesses and computations, and hence, the overall implementation energy. With modern low power embedded processors evolving towards register files with wide memory interfaces and vector functional units (FUs), the data flow in matrix decomposition algorithms needs to be carefully devised to achieve energy efficient implementation. In this paper, we present an efficient data flow transformation strategy for the Givens Rotation based QRD that optimizes data memory accesses. We also explore different possible implementations for QRD of multiple matrices using the SIMD feature of the processor. With the proposed data flow transformation, a reduction of up to 36% is achieved in the overall energy over conventional QRD sequences.**

I. Introduction

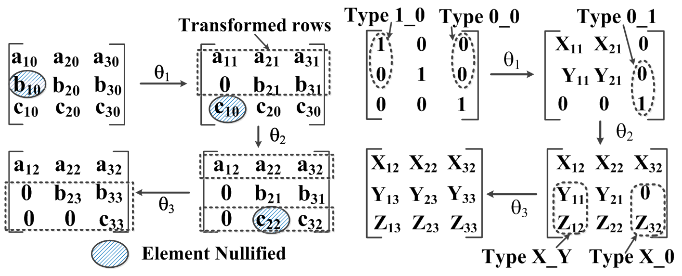
Embedded applications in the communication and multimedia domain widely use matrix decomposition as a sub-routine. QR Decomposition is popularly used for implementing functionalities such as matrix triangularization and inversion in MIMO (Multiple Input and Multiple Output) detection systems [1], [2]. It is also widely used in the domain of numerical analysis for solving a set of simultaneous linear equations and for computing eigen values. This finds applications in extracting principal components in data analysis, obtaining approximate solutions in image processing and coding, pattern recognition, and many other related areas. Exploration for energy efficient implementations of such a widely applicable functionality is an important area of research.



(a) Example matrix (b) GR matrix to nullify b (c) Row & Col. Elimination Seq.

Fig. 1: GR Matrix for a rotation in the example matrix

QR Decomposition of a matrix involves the factorization of a $M \times N$ matrix A, where $M \geq N$, into an orthogonal matrix Q of size $M \times M$ and a $M \times N$ upper triangular matrix R, with all the elements in the bottom $(M - N)$ rows as zero. Thus, QRD can be expressed as $A_{M\times N} = Q_{M\times M} \times R_{M\times N}$. Givens Rotation based matrix decomposition [1] can be summarized as follows. With each rotation, a new element in the lower triangular part of the input matrix is turned zero. Consider an example matrix shown in Figure 1(a) where a part of the matrix is already zeroed. Given two elements, a and b, of a column k from rows i and j respectively (such that $i \neq j$ and $1 \leq i, j \leq M$, $1 \leq k < N$) of matrix A, in which b is to be zeroed, we define $c = a/r, s = -b/r$ where $r = \sqrt{a^2 + b^2}$ in the Givens Rotation (GR) matrix (as shown in Figure 1(b)). A sequence of such transformations on the input matrix A results in converting it to an upper triangular matrix R. The same rotations are also performed on an identity matrix Q that has one in the diagonal entries to begin with, and is eventually filled in (Figure 2). Two well known sequences of Givens Rotations for obtaining an upper triangular matrix from an input matrix A are: *row elimination sequence* and *column elimination sequence* (Figure 1(c)).



(a) Rotations on input matrix (b) Rotations on Q matrix

Fig. 2: QRD with illustration for different types of operations

II. Related Work

The adoption of MIMO systems in modern wireless communication standards such as WLAN, LTE, and WiMAX has led to increasing computation complexity for matrix triangularization and inversion at the receiver end. Because of the higher reliability and lower computational complexity [1], [2] QRD is popularly used in MIMO-OFDM detection kernels. Power and energy constraints on the processors executing such systems make energy-efficient implementation of QRD an important point to be addressed for modern communication standards.

Previous research on QRD implementations has focused on hardware simplifications [3], [4], computation complexity reduction by using real decomposition method [5], and algorithmic choices such as Gram Schmidt [6], Householder transformations and Givens Rotation [1]. Givens Rotation based QRD is usually preferred for its implementation efficiency since it can be easily parallelized and also exploits the advantage of multiple zero entries in sparse matrices [1]. Architectures with the ability to perform operations on multiple pivots in parallel [7] or in pipelined fashion [8] have also been proposed to increase the throughput.

None of the above ASIC style implementations focuses on memory energy optimization which is very important in the context of our target processor architecture with wide memory interfaces, and hence, expensive memory accesses. In this work we propose an energy efficient data flow transformation for Givens Rotation based matrix decomposition that reduces the memory accesses and thus the overall energy for such architectures. We also explore different possible implementations using the SIMD feature of the processor.

III. Motivational Analysis

The number of rotations to be applied on an $N \times N$ square matrix to obtain an upper triangular matrix R is $\frac{N \times (N-1)}{2}$. These rotations involve different types of operations due to distinct operand types (as shown in Figure 2(b)) and can be categorized as: (1) Type X_Y: when both the entries on which rotation is performed are non-zero; (2) Type X_0: when one of the entries is non-zero and other is zero; (3) Type 0_0: when both the entries are zero; and (4) Type 1_0 or 0_1: when one of the entries is one and other is zero. Each of these is associated with different sets of computations depending on the operation type, and hence, different energies. For example, the Type X_Y operation below requires 4 MULs and 2 ADDs. For other types, we replace X and Y by 0/1, etc., and determine the operations.

$$\begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} c.X - s.Y \\ s.X + c.Y \end{bmatrix}$$

Table I summarizes the different operation and memory access counts when processing QRD for the conventional sequences (Figure 1(c)). We skip the detailed derivations due to space constraints. All the counts are in terms of scalar operations and individual memory word accesses. We observe that there is a trade-off between the memory accesses and computations – one sequencing strategy is better in one respect but worse in the other. Energy dissipation for the different categories are different and are architecture dependent; the final energy value depends on both counts. This analysis motivates us to determine an energy efficient data flow transformation for Givens Rotation based QRD.

Operations	**Q Matrix**		**Input Matrix**
	Row Seq.	**Col. Seq.**	
X_Y	$\frac{2N^3-6N^2+4N}{6}$	$\frac{N^3-4N^2+5N-2}{2}$	$\frac{N^3-N}{3}$
X_0	$N(N-2)$	$N(N-2)$	0
0_0	$\frac{N^3-3N^2+2N}{6}$	$\frac{N^2-3N+2}{2}$	$\frac{N^3-3N^2+2N}{6}$
0_1	N	N	0

	Input Matrix		**Q Matrix**
mem_acc	$\frac{4N^3-4N-6}{3}$	$\frac{4N^3-3N^2+5N-6}{3}$	$2N^3-2N^2$

TABLE I: Memory access and operation counts for conventional rotation sequences

IV. Proposed Strategy

For Givens Rotation based matrix decomposition of a $M \times N$ matrix, in column k, $(M-k)$ elements have to be nullified. Each rotation can be done by selecting a pair from any of the $(M-k+1)$ rows. The total number of possible orderings is given by the product of the possible choices for each column, which leads to an exponential number of possible rotation sequences with significant differences in the number of memory accesses and computations (as discussed in Section III). For a processor architecture with a memory interface of width W between the vector register file and Scratch Pad Memory (SPM), and SIMD FUs, the individual memory accesses and vector computations are expensive from an energy viewpoint. Thus, there is a need for a careful sequencing of the rotations to minimize the number of accesses and vector computations.

A. Processing Sequence

Some pruning strategies need to be adopted to arrive at a solution in the large exploration space for the rotation sequence in matrix decomposition. One important pruning strategy is: to nullify the element $A(i,j)$ of the input matrix, the pivot row (row used to null the element in another row) chosen should be such that all the elements $A(i,k)$ in the row with $k < j$ are already zero. Otherwise the already nullified entries will become non-zero. Another important requirement is to use the wide memory accesses judiciously. Availability of $2R$ vector registers of width W allows storing W consecutive elements from not more than R rows each of the input and *Q* matrices in the registers. The memory access counts discussed below are equal for both *Q* and *R* matrices as the corresponding rows are considered together.

In our proposed ordering, the matrix is divided into blocks in such a way that $(R-1)$ rows accessed in a block are not re-loaded in other blocks. As the processing progresses from block i to block $(i+1)$, the block size reduces by a square sub-matrix of 2$(R-1)$ rows from the previous block (as shown in Figure 3). A block is further divided into two sub-blocks, each with its own uniform nulling sequences (Figure 3). For the first sub-block, the nulling sequence proceeds from top to bottom nulling $(R-1)$ columns in each row while in the second sub-block of any block i the traversal is from the bottom to top for $(N-2i(R-1))$ rows; this is done to re-use the last accessed row. At the begining of each sub-block, $(R-1)$ rows are nullified at the desired positions for triangularization using column elimination sequence over the square block of size $(R-1) \times (R-1)$ (shaded squares in Figure 3). These are then used across the sub-block to nullify the entries in $(R-1)$ consecutive columns. This sequencing pattern is repeated over all blocks. Figure 3 shows an example of the proposed rotation sequence when R= 5.

The memory accesses associated with the proposed optimized sequence ($\#mem\_acc_{opt}$) can be estimated as:

$$\begin{aligned} \#mem\_acc_{opt}(R) &= 2\left\lceil \frac{N}{W} \right\rceil \left(\sum_{i=1}^{b}(N-(i-1)(R-1)) - (b-1)\right) \\ &\approx \left\lceil \frac{N}{W} \right\rceil \times \left(\frac{N^2}{R-1} + \frac{N(R-3)}{R-1} + 2 \right) \qquad (1) \end{aligned}$$

where $b = \left\lfloor \frac{N}{R-1} \right\rfloor$ is the number of sub-blocks into which the proposed sequence divides the rotations for matrix decomposi-

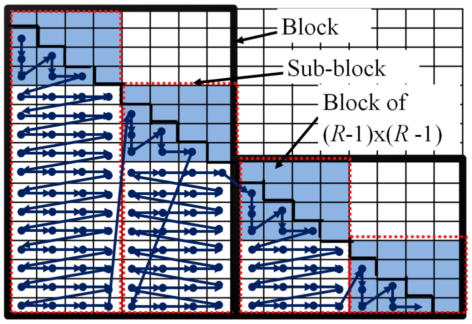


Fig. 3: Proposed sequence for 16×16 matrix (R = 5)

tion. The factor $\lceil \frac{N}{W} \rceil$ corresponds to N elements in a row, with W elements being accessed in a single load. Across the blocks one memory access is re-used without any extra memory access so a factor of $(b-1)$ is subtracted. A factor of two accounts for reads and writes performed on each element. In the proposed sequencing, the rotation sequence changes with the number of registers, which leads to different computation counts for Q matrix. Operations of Type 0_0 arise only in the first block of $(R-1)$ register accesses. The subsequent rotations carried out using these over the remaining rows, along with the rotation with i = R and j = $(R+1)$, only contribute to Type 0_0 count. The total count can thus be estimated as:

$$\#\text{Type 0\_0}_{opt}(R) = \sum_{j=2}^{N} \sum_{i=1}^{min(j-1,R-1)} (N-j) + (N-(R+1))$$
$$= \frac{3N^2(R-1) + 3N(1-R^2) + (R^3-R)}{6} \quad (2)$$

The total number of operations remaining the same at $N \times \frac{N(N-1)}{2}$, the increase in Type 0_0 count with the increasing number of registers causes a corresponding decrease in Type X_Y count (the sum of the two is constant).

Comparison of the proposed ordering with the conventional sequences: The rotation sequence for the conventional sequences is independent of the number of registers. Thus, the scalar operation counts remain the same (Table I) with the variation in memory accesses as follows. Details are omitted in the interest of brevity.

$$\#mem\_access_{row}(R) = \left\lceil \frac{N}{W} \right\rceil \times (N^2 + N(5-2R) + (R^2-3R-4)) \quad (3)$$

$$\#mem\_access_{col}(R) = \left\lceil \frac{N}{W} \right\rceil \times (N^2 + N(3-2R) + (R^2-R)) \quad (4)$$

Comparing the memory accesses for different sequences discussed above, we can conclude that $\#mem\_access_{opt} < \#mem\_access_{col} < \#mem\_access_{row}$. Thus, the proposed strategy is better than both the conventional sequences in terms of memory accesses. However, it has an additional compute overhead with respect to row sequence, as the Type 0_0 operation count comparison goes as: $\#\text{Type 0\_0}_{col} < \#\text{Type 0\_0}_{opt} < \#\text{Type 0\_0}_{row}$ (from Equation 2 and Table I) with reverse ordering for Type X_Y. This reduces the overall energy savings when computations are considered.

B. SIMD Implementation

Exploiting the SIMD feature of our target architecture, we propose two possible ways of implementing QR decomposition.

1) SIMD within matrix: Vector registers in the processor architecture are used for loading multiple elements from the rows of a matrix (as shown in Figure 4(a)). In this case, the rotation operation can be applied simultaneously on all W elements of a row fetched together. This results in $\lceil \frac{N}{W} \rceil$ operations of Type X_Y per rotation (as each row has at least one non-zero entry even in the input diagonal matrix). The distinction between different operation types (Type X_Y, etc.) is not made since the same operation type would have to be performed on all the W elements in a SIMD FU. The computation count is thus given by $\lceil \frac{N}{W} \rceil \times \frac{N(N-1)}{2}$. As a result, only reductions in memory accesses due to the ordering of rotation sequences (Equations 3, 4 and 1) contribute to the overall energy savings.

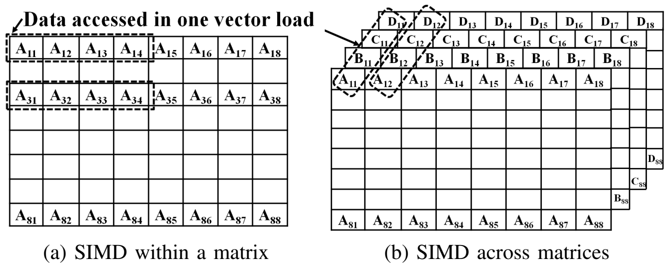

Data accessed in one vector load
A11 A12 A13 A14 A15 A16 A17 A18
A31 A32 A33 A34 A35 A36 A37 A38
A81 A82 A83 A84 A85 A86 A87 A88
(a) SIMD within a matrix
D11 D12 D13 D14 D15 D16 D17 D18
C11 C12 C13 C14 C15 C16 C17 C18
B11 B12 B13 B14 B15 B16 B17 B18
A11 A12 A13 A14 A15 A16 A17 A18
D88
C88
B88
A81 A82 A83 A84 A85 A86 A87 A88
(b) SIMD across matrices

Fig. 4: Data access pattern for input matrices

2) SIMD across matrices of same size: In this technique, the vector registers are used to access the elements across the matrices. Elements corresponding to the same position from W matrices are accessed together (Figure 4(b)). For this implementation, the different types of computations also play a significant role. The grouping of the matrices lead to the same operation type (X_Y, X_0, etc.), across the vector register (which is not the case when SIMD is employed within a matrix). Since the energy associated with SIMD operations of different types are significantly different, the variations in their count affect the overall energy. Considering C matrices of dimension $(N \times N)$, the memory access counts can be obtained by multiplying the expressions in Equations 3, 4 and 1 by $\lceil C/W \rceil \times N$ instead of $\lceil N/W \rceil$, as only elements of W out of C matrices can be operated upon in parallel and loads for all N elements in a row have to be performed. The corresponding operation counts can be estimated by multiplying the counts in Table I and Equation 2 (for the proposed sequence) by $\lceil C/W \rceil$ as one operation is now performed across C matrices.

V. Experimental Results

We evaluated our proposed data flow transformation by comparing the energy dissipation against that of the conventional row and column elimination sequences on matrices of different sizes, for different SIMD implementations. We implemented the algorithm on the processor template [9] shown in Figure 5 using an extension of the IMPACT compiler framework [10]. A cycle accurate processor simulator was used to obtain the run-time statistics for the analysis. The architecture comprises a Register file, Scratch pad memories (SPMs), and SIMD functional units. The register file has a wide interface (SIMD width W = 8, i.e., 256 bits) to both the SPMs and vector FUs and a narrower interface (32 bit wide) with VLIW FUs. The HDL model of the processor was synthesized using Cadence

RTL compiler and power simulations were carried out using Synopsys Primepower with a 40nm technology library. Energy numbers for memory structures in the processor were derived using a commercial memory compiler.

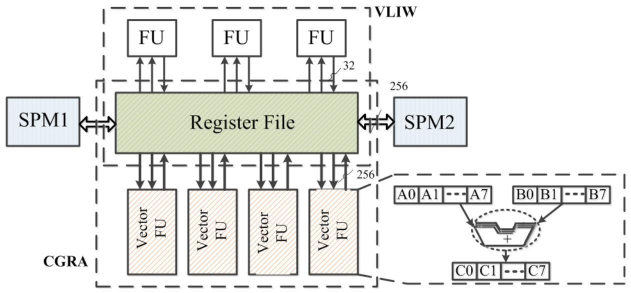


Fig. 5: Baseband Processor Instance

A. *SIMD within a matrix*

Figure 6(a) shows a comparison of the memory access counts normalized to the corresponding values for the proposed sequence (opt_seq) for different matrices of dimension $N \times N$. Comparing our proposed strategy with the conventional column sequence, the reduction in memory accesses varies from 26.1% for a small matrix of size 8×8 to 48.8% for N = 128. The percentage improvement over the conventional row sequence is higher, with 34.61% for a 8 × 8 matrix. We observe that these improvements increase with increasing matrix size. Figure 6(b) compares the total energy for the different rotation sequences. The overall energy savings are due to the differences in memory accesses only, with compute energy remaining unchanged (as discussed in Section IV-B1). For matrix sizes close to the number of available registers (as in $N = 4$), different rotation sequences result in similar memory access counts, achieving no energy savings.

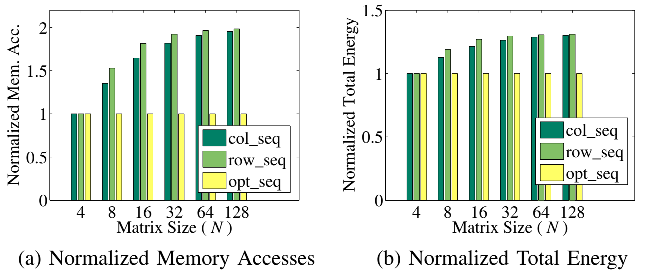


Fig. 6: Memory access and total energy comparison for SIMD within a matrix (# Registers $2R$ = 6, W = 8)

B. *SIMD across matrices*

Figure 7 (b) illustrates the improvements obtained with our proposed strategy against the conventional sequences assuming that W matrices are processed together. The reductions obtained in memory access counts are similar to those with SIMD within a matrix because the number of matrices under consideration is a multiple of W, which fully utilizes the vector accesses.

Here the total energy for row elimination sequence is generally lower than that for the column sequence (in contrast to SIMD within a matrix) because of the reduction in compute energy due to greater number of Type 0_0 operations. Thus, with

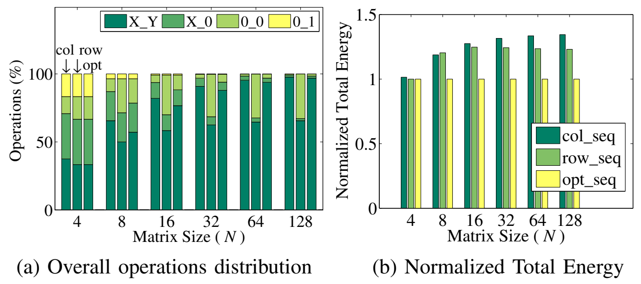


Fig. 7: Memory access and total energy comparison for SIMD across multiple (C) matrices ($C = W$, W = 8)

the proposed data flow transformation for QRD we report energy savings with respect to both memory access energy and compute energy compared to the conventional column sequence. Energy savings of 15% for N = 8 and 23.5% for N = 128 are obtained over the column sequence. Comparison against row sequencing shows that our proposed sequencing results in larger gain with respect to memory accesses, though it has a compute energy overhead. However, the total energy is still lower (upto 17% for N = 128), as shown in Figure 7(b).

VI. CONCLUSIONS

The sequence of rotations plays an important role in determining the computation and memory access energy in the Givens rotation based matrix decomposition algorithm. We presented a data flow transformation strategy for QRD targeting embedded processor systems with features such as SPMs, SIMD FUs, and vector register files with wide interfaces with both SPM and FUs. We also presented different implementations for QRD exploiting the SIMD feature of the architecture. Experimental results show a significant reduction (upto 36%) in overall energy across different implementations with our proposed strategy.

REFERENCES

[1] Z.-Y. Huang and P.-Y. Tsai, "Efficient Implementation of QR Decomposition for Gigabit MIMO-OFDM Systems," *TCS-I*, 2011.
[2] L. Ma, K. Dickson, J. McAllister, and J. McCanny, "QR Decomposition-Based Matrix Inversion for High Performance Embedded MIMO Receivers," *TSP*, 2011.
[3] A. Maltsev, V. Pestretsov, R. Maslennikov, and A. Khoryaev, "Triangular systolic array with reduced latency for QR-decomposition of complex matrices," in *ISCAS*, 2006.
[4] J. Rust, F. Ludwig, and S. Paul, "Low complexity QR-decomposition architecture using the logarithmic number system," in *DATE*, 2013.
[5] Y.-T. Hwang and W.-D. Chen, "A low complexity complex QR factorization design for signal detection in MIMO OFDM systems," in *ISCAS*, 2008.
[6] C. Singh, S. H. Prasad, and P. Balsara, "VLSI Architecture for Matrix Inversion using Modified Gram-Schmidt based QR Decomposition," in *VLSI Design*, 2007.
[7] M.-W. Lee, J.-H. Yoon, and J. Park, "High-speed tournament givens rotation-based QR Decomposition Architecture for MIMO Receiver," in *ISCAS*, 2012.
[8] M. Hofmann and E. J. Kontoghiorghes, "Pipeline Givens sequences for computing the QR decomposition on a EREW PRAM," *Parallel Computing*, 2006.
[9] N. Sharma, T. VanderAa , P. Agrawal , P. Raghavan, P. R. Panda, and F. Catthoor, "Data Memory Optimization in LTE Downlink," in *ICASSP*, 2013.
[10] B. Mei, S. Vernalde, D. Verkest, H. D. Man, and R. Lauwereins, "DRESC: a retargetable compiler for coarse-grained reconfigurable architectures," in *FPT*, 2002.