FPGA Based Systolic Array Implementation of QR Transformation Using Givens Rotations

Xiaojun Wang, Miriam Leeser
Department of Electrical and Computer Engineering, Northeastern University
xjwang,xjwang@ece.neu.edu

QR transformation is used in many signal processing applications including echo cancellation, denoising, and beamforming. It is well known for its computational stability and fast convergence. We have implemented a systolic array QR transformation on a Xilinx Virtex5 FPGA using the Givens rotation algorithm. The massively parallel nature of this algorithm is well suited to an FPGA architecture which can greatly reduce the computation time. The latency of our implementation is very small and scales well for large matrix sizes. It is also fully pipelined with a throughput of over 130 MHz for IEEE single precision floating-point format. The level of parallelism is determined by the matrix size, data format, and FPGA device used.

Design Architecture

The QR transformation of an $m \times n$ matrix A is given by $A = QR$, where Q is an $m \times m$ unitary matrix and R is an $m \times n$ upper triangular matrix. Givens rotations is one method for solving QR by applying a series of rotations to the rows of the original matrix. One rotation introduces a zero in a lower triangular 2×2 sub-diagonal matrix as shown in Eq. (1).

$$\begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix} \tag{1}$$

The c and s stand for sine and cosine parameters and can be computed as follows:

$$c = \frac{x}{\sqrt{x^2 + y^2}} \qquad s = -\frac{y}{\sqrt{x^2 + y^2}} \tag{2}$$

Since one rotation involves only two matrix rows, multiple rotations can be computed in parallel if they operate on different rows, leading to potentially massive parallelism. Previous work on QR using Givens rotations avoids divide and square root steps in the algorithm by using either Logarithmic Number System (LNS) [1] or CORDIC algorithm [2].

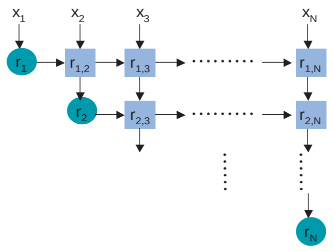
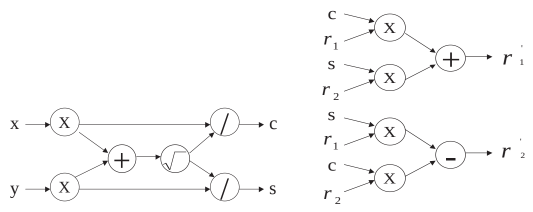


Figure 1: QR Systolic Array Structure

Using Givens rotations, QR transformation can be performed by a highly parallel implementation – a 2D systolic array architecture as shown in Fig. 1. This systolic array includes two types of computational processing element (PE) – diagonal and off-diagonal. The diagonal PE generates rotation parameters c and s as in Eq. (2) and broadcasts them to all off-diagonal PEs in the same row of the systolic array. The off-diagonal PE updates other elements of the two rows of the matrix involved in one rotation. It starts upon receiving the c and s from the diagonal PE from the same row. The updated values are then sent to the PEs in the next row of the systolic array. Upon update completion, the new values of the first off-diagonal PE are sent to the diagonal PE below it, stimulating the computation of that diagonal PE immediately. The new values of the rest of the off-diagonal PEs have to be stored in local memory before being passed to the off-diagonal PEs in the next row. This is because those next row off-diagonal PEs cannot start computation until they get the c and s from the diagonal PE of their row. For an $n \times n$ square matrix, the systolic array has n rows. Each row has one diagonal PE and a set of off-diagonal PEs. Every off-diagonal PE except the first one of a row has one local memory of 8 words. The number of off-diagonal PEs is n-1 in the first row, n-2 in the second row, etc. Altogether n diagonal PEs, $\frac{n*(n-1)}{2}$ off-diagonal PEs, and $\frac{(n-1)*(n-2)}{2}$ lo-



(a) Diagonal (b) Off-diagonal

Figure 2: Processing Element Data Flow

cal memories are needed. In addition to processing a square matrix, our implementation works for any size input matrix, with slightly different systolic array structure and number of PEs.

Fig. 2(a) shows one diagonal PE, which requires two multipliers, one adder, one square root and two dividers. All units are in floating-point format and are modules in the Northeastern University VFloat (Variable Precision Floating-Point) library[1]. All units including the floating-point divide and square root [3] are fully pipelined, making the diagonal PE fully pipelined. Fig. 2(b) shows one off-diagonal PE, which has four multipliers, one adder and one subtractor.

Results

The complete 2D systolic array was designed targeting a Xilinx XC5VLX220 FPGA. The design is in VHDL, synthesized using Synplify Pro 8.8; and the bitstream is generated using Xilinx ISE 9.1i. Two floating-point formats were implemented. One is IEEE single precision with 8-bit exponent and 23-bit mantissa; the other is a smaller format with 8-bit exponent and 11-bit mantissa. To achieve maximum parallelism, we explore fitting as many PEs as possible on a XC5VLX220 FPGA,which has 138240 slices, 192 blockRAMs, and 128 embedded DSPs. For an $m < n$ short matrix, resources vary with both the number of rows and columns. For an $m > n$ tall matrix, the resources are almost independent of the number of rows. A very tall matrix with $m >> n$ requires about the same resources as an $n \times n$ square matrix. Our experiments show that a Xilinx XC5VLX220 FPGA can fit an input matrix with up to 7 columns for 23-bit format and up to 12 columns for 11-bit format.

Table 1 gives the resources and speed for

[1]http://www.ece.neu.edu/groups/rcl/projects/

Table 1: Resources and Speed: XC5VLX220

Size	Slice	BRAM	DSP	Latency	Freq.
(8,23)	126585	56	102	954	132
(8,11)	120094	30	106	335	139

a 7×7 matrix with 23-bit format and a 12×12 matrix with 11-bit format. Both implementations have high throughput: 132 MHz for single precision and 139 MHz for small format. Fig. 3 shows the latency as matrix size varies from 2 to 7 for IEEE single precision floating-point format. We estimate the latency of a sequential implementation for comparison. For 23-bit format, the latency of one diagonal PE and one off-diagonal PE is 47 and 15 clock cycles respectively. So the conservative total latency estimate is $(47 + 15) * \frac{n*(n-1)}{2}$ for core computation. For a 7×7 matrix, it is about 1302, which is longer than 954 realized in our parallel implementation. The difference is more significant as matrix size grows. Compared to the sequential implementation with $O(n^2)$ latency, the latency of our systolic array implementation increases linearly with matrix size, making it scale well for larger matrices.

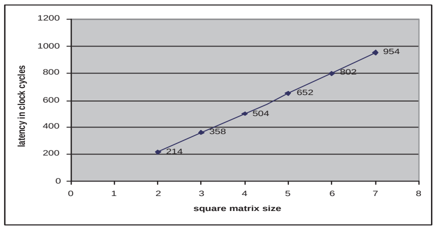


Figure 3: Latency vs. Matrix Size

References

[1] J. Schier and A. Hermanek, “Using logarithmic arithmetic to implement the recursive least squares (QR) algorithm in FPGA,” in *FPL’04*, pp. 1149–1151, Aug. 2004.

[2] D. Boppana, K. Dhanoa, and J. Kempa, “FPGA based embedded processing architecture for the QRD-RLS algorithm,” in *FCCM’04*, pp. 330–331, Apr. 2004.

[3] X. Wang, S. Braganza, and M. Leeser, “Advanced components in the variable precision floating-point library,” in *FCCM ’06*, pp. 249–258, Apr. 2006.