Hardware-Efficient Schemes of Quaternion Multiplying Units for 2D Discrete Quaternion Fourier Transform Processors

Abstract

In this paper, we offer and discuss three efficient structural solutions for the hardware-oriented implementation of discrete quaternion Fourier transform basic operations with reduced implementation complexities. The first solution - a scheme for calculating $sq$ product, the second solution – a scheme for calculating $qt$ product, and the third solution – a scheme for calculating $qt$ product, where $s$ is a so-called $j$-quaternion, $t$ is an $j$-quaternion, and $q$ is an usual quaternion. The direct multiplication of two usual quaternions requires 16 real multiplications (or two-operand multipliers in the case of fully parallel hardware implementation) and 12 real additions (or binary adders). At the same time, our solutions allow to design the computation units, which consume only 6 multipliers plus 6 two input adders for implementation of $sq$ or $qt$ basic operations and 9 binary multipliers plus 6 two-input adders and 4 four-input adders for implementation of $qt$ basic operation.

Keywords: discrete quaternion Fourier transform, fast algorithms, implementation complexity reduction, FPGA implementation

1. Introduction

Two dimension discrete Fourier transform (2D-DFT) have been widely used in image processing ever since the discovery of Fast Fourier transform (FFT) which made the computation of DFT feasible using a computer [1]. However, if we want to apply the classical 2D-FFT to color images, we must perform three separate 2D-FFTs. This is because every color image pixel has three values associated with it: the red, green, and blue components.

Until recently, it was not offered any discrete transform, which would perceive each of the pixel of the color image as a whole. Sangwine and Ell defined a new transform, called the discrete quaternion Fourier transform (DQFT) which allows to process simultaneously of all color components of the image [2-4]. The idea of DQFT is based on representation of color image pixels via quaternions – four-dimensional hypercomplex numbers discovered by Hamilton in 1843. Today there are several ways to calculate the 2D-DQFT [5-13]. Another way for calculating the QDFT has been reported in [14]. While it produces the same result as the other approaches, it is more efficient because leads to reducing the computation complexity. During the DQFT implementation using the method proposed in [14], it is necessary to perform three types of the quaternion multiplication operation, namely: left-sided quaternion multiplication, right-sided quaternion multiplication and two-sided quaternion multiplication. What is more, in all three cases only one quaternion is a usual quaternion, and the rest quaternions are constant quaternions, i.e. quaternions, which coefficients are real constants. Next we propose hardware-effective schemes to implement these operations.

2. Statement of the problem

The typical operations of the related two dimensional forward and inverse discrete quaternion Fourier transform are [14]:

- left-sided quaternion multiplication $sq$,
- right-sided quaternion multiplication $qt$ and
- two-sided quaternion multiplication $sqt$.

Where $q = q_0 + q_1i + q_2j + q_3k$ is a conventional quaternion, with three imaginary units $ij = k$, $ji = -k$, $i^2 = j^2 = k^2 = ijk = -1$ and four real variables $\{q_n\} = \{0,1,2,3\}$; $s = \alpha + \beta k$, $t = \gamma + \delta k$ are so-called $i$ – quaternion and $j$ – quaternion respectively [14], and $\alpha, \beta, \gamma, \delta$ are real constants.

During synthesis of the discussed schemes we use the fact that multiplication of two quaternions may be represented as vector-matrix product [15, 16]. The matrix appears in the software implementation has unique structural properties that allow performing its advantageous factorization [17]. Namely this factorization leads to significant reducing of the computational complexity of quaternion multiplication. Furthermore, since $s$ and $t$ are truncated quaternions and are in fact complex numbers, which located in the different domains of complex space, the corresponding matrices are sparse. This leads to additional effect in minimizing of the computational complexity. Finally, an additional effect can be achieved using the fact that the numbers $\alpha, \beta, \gamma$ and $\delta$ are real constants and their products can be calculated and stored in memory in advance.

3. The schemes

Let $X_{4x1} = [q_0, q_1, q_2, q_3]^T$ - be a column vector, that contains the all coefficients of quaternion $q$, and $Y_{4x1} = [y_0, y_1, y_2, y_3]^T$ - be a column vector containing the elements of $sq$ product.

The first scheme (for implementation $sq$ - kernel) can be written with the help of following matrix-vector calculating procedure:

$$\begin{equation}
Y_{4}^{(l)} = P_{4}^{(l)} X_{4}^{(l)} A_{6}^{(l)} D_{6}^{(l)} X_{4}^{(l)}
\end{equation}
$$

where

$$A_{6}^{(l)} = \begin{bmatrix}
1 & -1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix},$$

$$D_{6}^{(l)} = \text{diag}(\alpha, d_1, d_2, d_1, d_2, \alpha), \ d_1 = \alpha + \beta, \ d_2 = \alpha - \beta,$$

$$P_{4}^{(l)} = \begin{bmatrix}
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0
\end{bmatrix},$$

$$A_{4}^{(l)} = \begin{bmatrix}
-1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix},$$

Fig. 1 shows a data flow diagram of the proposed scheme for realization $sq$ product kernel. In this paper, data flow diagrams are oriented from left to right. Straight lines in the figures denote the operations of data transfer. The circles in these figures show the operation of multiplication by a real number inscribed inside a circle. Points where lines converge denote summation a dotted lines indicate the sign-change operations. We use the usual lines without arrows on purpose, so as not to clutter the picture.
Fig. 1. The data flow diagram of proposed scheme for implementation $sq$ product kernel

Let $Y_{4\times 1}^{(2)} = [y_0^{(2)}, y_1^{(2)}, y_2^{(2)}, y_3^{(2)}]^T$ be a column vector containing the elements of $qt$ product.

Then the second scheme (for implementation $qt$ - kernel) can be written with the help of following matrix-vector calculating procedure:

$$Y_{4\times 1}^{(2)} = P_{4\times 1}^{(2)} A_{4\times 6}^{(2)} D_{6\times 6}^{(2)} P_{6\times 4}^{(2)} X_{4\times 1}$$

where

$$P_{4\times 1}^{(2)} = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}, \quad A_{4\times 6}^{(2)} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \end{bmatrix},$$
$$D_{6\times 6}^{(2)} = \text{diag}(\alpha, d_1, d_2, d_3, d_4, d_5), \quad g_1 = \gamma - \delta, \quad g_2 = \gamma + \delta.$$

Fig. 2 shows a data flow diagram of the proposed scheme for implementation $qt$ product kernel.

Let $Y_{4\times 1}^{(3)} = [y_0^{(3)}, y_1^{(3)}, y_2^{(3)}, y_3^{(3)}]^T$ be a column vector containing the elements of $sqt$ product. Then the third scheme (for implementation $sqt$ - kernel) can be written with the help of following matrix-vector calculating procedure:

$$Y_{4\times 1}^{(3)} = A_{4\times 9} D_{9\times 9} W_{9\times 7} W_{7\times 4} P_{4\times 4} X_{4\times 1}$$

where

$$\begin{align*}
P_4 &= \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}, \\
D_9 &= \text{diag}(p_6, p_5, p_2, p_4, p_1, p_4, p_3, p_6, p_5),
\end{align*}$$

$\alpha = (\alpha - \beta)$, $p_2 = \alpha \delta$, $p_3 = \beta \delta$, $p_4 = (\alpha - \beta)(\gamma - \delta)$, $p_5 = \alpha (\gamma - \delta)$, $p_6 = \alpha (\gamma - \delta)$.

Fig. 3 shows a data flow diagram of the proposed scheme for implementation $sqt$ product kernel. The rectangles indicate the operations of multiplication by the matrices $H_2$ and $H_2$. 

\[ \text{Fig. 1. The data flow diagram of proposed scheme for implementation } sq \text{ product kernel} \]

\[ \text{Fig. 2. The data flow diagram of proposed scheme for implementation } qt \text{ product kernel} \]

\[ \text{Fig. 3. The data flow diagram of proposed scheme for implementation } sqt \text{ product kernel} \]
4. Conclusion

The article presents three new hardware-efficient schemes for the execution $sq \cdot product$, $qt \cdot product$ and $sqqt \cdot product$ kernels with reduced computational complexities. To reduce the hardware complexity (number of embedded adders and multipliers), we exploit the specific structural properties of the matrix-vector products that represent mentioned basic operations. So, the fully parallel implementation of $sq \cdot product$ and $qt \cdot product$ kernels require only 6 multipliers by real numbers, and 6 adders. In turn, a fully parallel implementation of $sqqt \cdot product$ kernel requires only 9 binary multipliers, 6 two-input adders and 4 four-input adders.

Reducing the number of multiplications is especially important in the design of specialized VLSI on-board DSP processors because minimizing the number of necessary multipliers also reduces the power dissipation and lowers the cost implementation of the entire system being implemented. This is because a hardware multiplier is more complicated unit than an adder and occupies much more chip area than the adder. (It is proved that the hardware complexity of an embedded multiplier grows quadratically with operand size, while the hardware complexity of an binary adder increases linearly with operand size). Even if the VLSI chip already contains embedded multipliers, their number is always limited. This means that if the implemented scheme has a large number of multiplications, the projected processor may not always fit into the chip and the problem of minimizing the number of multipliers remains relevant.

This problem becomes extremely challenging for applications requiring real-time processing at high throughput especially in digital signal and image processing. Hence, for meeting the high requirements to throughput and power-consumption constraints of real-time image processing systems, developing hardware-efficient schemes to implement them on the base of application specific integrated circuits (ASICs) or field-programmable gate arrays (FPGAs) is of paramount importance.

5. References

[1] Pratt W. K., Digital Image Processing, Part III, Chapter 8, pp. 185-212
3rd Edition, John Wiley & Sons, Inc. 2001. ISBNs: 0-471-37407-5.

[2] Sangwine S. J. Fourier transforms of colour images using quaternion or hypercomplex numbers, Electronics Letters, Vol. 32, No 21, 10 Oct 1996., Pages: 1979 – 1980 DOI: 10.1049/el:19961331

[3] Sangwine S. J. The discrete quaternion Fourier transform, 6th International Conference on Image Processing and its Applications, 1997 p. 790 – 793, DOI: 10.1049/ic:19971004

[4] Ell T. A., Sangwine S. J. Decomposition of 2D hypercomplex Fourier transforms into pairs of complex Fourier transforms. In F. Solina and A. Leonardis, editors, Computer Analysis of Images and Patterns, vol. 1689 of Lecture Notes in Computer Science, pp. 209–216. Springer, 1999, 8th International Conference - CAIP’99, Ljubljana, September 1–3, 1999 Proceedings.

[5] Said S., Le Bihan N., Sangwine S. J., Fast complexified quaternion Fourier transform, IEEE Transactions on Signal Processing 2008, Vol. 56, No 4., pp. 1522-1531.

[6] Chichyeva M. A., Peshina M. V. On various schemes of 2D-DFT decomposition with data representation in the quaternion algebra. //Image Processing and Communications, Institute of Telecommunications, Bydgoszcz, Poland, Vol. 2, No 1., pp. 13-20, 1996.

[7] Tariova G., Tariov A., Aspekty algorytmiczne redukcji liczby bloków mnożących w układzie do obliczania iolozcyn dwóch kwaternionów, Pomiary, Automatyka, Kontrola, No 7, 2010, str. 668-690.

[8] Parfenienku M., Park S. Y., Sparse-iteration 4D-QDGRIC Algorithms for multiplying quaternions, IEEE Transaction on Computers, 2016, v. 65, no 9, pp. 2859-2871.

[9] Curio A., Strategies for the Synthesis of Fast Algorithms for the Computation of the Matrix-vector Products, Journal of Signal Processing Theory and Applications, 2014, v. 3 No 1. pp. 1-19.