A Matrix-Vector Based Approach to FFT Implementations

Conference Paper · January 2010

DOI: 10.1109/ISSPIT.2009.5407501 · Source: IEEE Xplore

4 authors:

Yuheng He
Ruhr-Universität Bochum
5 PUBLICATIONS **26** CITATIONS
SEE PROFILE

Klaus Hueske
Technische Universität Dortmund
24 PUBLICATIONS **67** CITATIONS
SEE PROFILE

J. Gotze
Technische Universität Dortmund
66 PUBLICATIONS **463** CITATIONS
SEE PROFILE

Edmund Coersmeier
Task9 GmbH
21 PUBLICATIONS **61** CITATIONS
SEE PROFILE

A Matrix-Vector Based Approach to FFT Implementations

Yuheng He[1], Klaus Hueske[2], Jürgen Götze[2], Edmund Coersmeier[3]
[1] Institute for Integrated Systems, Ruhr University Bochum, 44780 Bochum, Germany
[2] TU Dortmund University, Information Processing Lab, Dortmund, Germany
[3] Task9 GmbH, 44799 Bochum, Germany
Yuheng.He@is.ruhr-uni-bochum.de

Abstract— *Today Discrete Fourier Transforms (DFTs) are applied in various radio standards based on OFDM (Orthogonal Frequency Division Multiplex). To achieve a high computational speed with low power consumption, specialized Fast Fourier Transform (FFT) engines are used in mobile devices. However, in face of the Software Defined Radio (SDR) development, more general (parallel) processor architectures are often desirable, which are not necessarily tailored to FFT computations. Therefore, alternative approaches are required to reduce the complexity of the DFT. Starting from a matrix-vector based description of the FFT idea, we will present different factorizations of the DFT matrix, which allow a reduction of the complexity. The resulting complexity lies between the original DFT and the minimum FFT complexity. The computational complexities of these factorizations and their suitability for implementation on different processor architectures are investigated.*

Keywords— *FFT, multi-core, SDR, matrix-vector, OFDM*

I. Introduction

Recently multi-carrier modulation techniques, such as OFDM, have received great attention in high-speed data communication systems and have been selected for several communication standards (e.g. WLAN, DVB, WiMax and LTE). A central baseband signal processing task required in OFDM transceivers is the Fourier transform, which is generally realized as FFT [1]. To meet the real-time processing requirements digital signal processors (DSPs) with optimized structures, e.g. native support of butterfly operations, or specialized FFT processors can be used [2].

With the progress in processor performance Software Defined Radio (SDR) approaches for mobile terminals are gaining momentum [3]. Different SDR architectures are discussed: One concept is based on heterogeneous architectures that are built of different types of processors, e.g. DSPs, general purpose processors or hardware accelerators. In this concept the FFT can be implemented as a reconfigurable accelerator to achieve high data throughput, e.g. [4]. Another approach is a homogeneous architecture with massive parallel structures. This can be a single instruction multiple data (SIMD) vector processor or a multi-core architecture [5][6].

In terms of mathematical operations the FFT is the most efficient way to compute the DFT. However, to yield a high flexibility of the platform the used architectures are not necessarily optimized for butterfly structures, which leads to the question: Are there any efficient procedures, which reduce the computational complexity of the DFT significantly (as far as possible to the FFT complexity), but at the same time are more tailor-made to specific parallel hardware architectures than the FFT?

In this paper a matrix-vector based formulation of the FFT algorithm [7] is derived from the respective matrix factorizations of the DFT matrix. Due to the structure of the butterfly matrices and the matrix factorizations, the matrix-vector based FFT allows several separations of the resulting matrix product. These separations have an increased computational complexity compared to the FFT, but due to their mathematical structure they allow more flexible implementations on different architectures (e.g. multi core processors or vector processors). A similar mathematical description is used in [8] to analyze the performance of automatically generated code on different parallel architectures.

The paper is organized as follows: In Section II we will clarify the definition and notation of the DFT. A short repetition of the FFT idea and the corresponding factorization is described in Section III. In Section IV different separations of the butterfly matrices are presented, which lead to specific algorithmic structures and different computational complexities. Implementation issues of these separations and their suitability for different hardware architectures are discussed in Section V. Conclusions are given in Section VI.

II. Discrete Fourier Transform (DFT)

The sequence of N (with $N = 2^m, m \in \mathbb{N}$) complex numbers $x_1, \cdots, x_N$ is transformed into the sequence of N complex numbers $y_1, \cdots, y_N$ by the DFT according to

$$y_t = \sum_{n=1}^{N} x_n w_N^{(t-1)(n-1)}, \quad t = 1, \cdots, N, \tag{1}$$

where $w_N = e^{-j\frac{2\pi}{N}}$ is the primitive N-th root of unity. The DFT can also be formulated as a matrix vector product [7]. With

$$\mathbf{x}_N = [x_1 \cdots x_N]^T \in \mathbb{C}^N \tag{2}$$

and

$$\mathbf{y}_N = [y_1 \cdots y_N]^T \in \mathbb{C}^N, \tag{3}$$

the DFT can be described as matrix vector product

$$\mathbf{y}_N = \mathbf{F}_N \cdot \mathbf{x}_N, \tag{4}$$

where $\mathbf{F}_N \in \mathbb{C}^{N \times N}$ is the DFT-Matrix with elements $f_{nt} = w_N^{(t-1)\cdot(n-1)}$ and $n, t = 1, \cdots, N$.

III. Matrix-Vector Based FFT

To give a repetition of the known radix-2 FFT algorithms [1], this section will show the connection between the matrix $\mathbf{F}_N$ and the matrix $\mathbf{F}_{N/2}$, which allows us to compute an N-point DFT from a pair of $(N/2)$-point DFTs. We introduce a so-called permutation matrix $\mathbf{P}_N$ of size $(N \times N)$, which performs an odd-even ordering of the rows (multiplied from the left by $\mathbf{P}_N$) or columns (multiplied from the right by $\mathbf{P}_N^T$) of a matrix. Furthermore $\mathbf{P}_N^T \cdot \mathbf{P}_N = \mathbf{I}_N$ holds. The following factorizations and separations are shown for the decimation in time (DIT) FFT. They can be easily transferred to the decimation in frequency (DIF) FFT.

A. FFT-Matrix Decomposition

Applying $\mathbf{P}_N^T$ to the right of the DFT-matrix $\mathbf{F}_N$ yields

$$\mathbf{F}'_N = \mathbf{F}_N \cdot \mathbf{P}_N^T = \begin{bmatrix} \mathbf{F}_{\frac{N}{2}} & \mathbf{D}_{\frac{N}{2}} \cdot \mathbf{F}_{\frac{N}{2}} \\ \mathbf{F}_{\frac{N}{2}} & -\mathbf{D}_{\frac{N}{2}} \cdot \mathbf{F}_{\frac{N}{2}} \end{bmatrix}, \tag{5}$$

where the odd indexed columns of $\mathbf{F}_N$, forming the left half of $\mathbf{F}'_N$, are composed of $\mathbf{F}_{N/2}$, while the even indexed columns, forming the right half of $\mathbf{F}'_N$, are composed of $\mathbf{D}_{N/2} \cdot \mathbf{F}_{N/2}$ and $-\mathbf{D}_{N/2} \cdot \mathbf{F}_{N/2}$. This matrix can be decomposed into the product of two matrices, i.e.

$$\mathbf{F}'_N = \begin{bmatrix} \mathbf{I}_{\frac{N}{2}} & \mathbf{D}_{\frac{N}{2}} \\ \mathbf{I}_{\frac{N}{2}} & -\mathbf{D}_{\frac{N}{2}} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{F}_{\frac{N}{2}} & \mathbf{0} \\ \mathbf{0} & \mathbf{F}_{\frac{N}{2}} \end{bmatrix}, \tag{6}$$

where $\mathbf{D}_{N/2}$ is a diagonal matrix with elements $d_{nn} = w_N^{(n-1)}$ and $n = 1, \cdots, N/2$. The first matrix represents FFT butterfly operations, while the second matrix contains two independent DFTs of length $N/2$. Note that the second matrix is a block diagonal matrix.

B. Matrix-Vector Based Computation of the FFT

To apply the modified DFT $\mathbf{F}'_N$ to an input vector $\mathbf{x}_N$, a permutation of $\mathbf{x}_N$ is required, i.e.

$$\mathbf{P}_N \cdot \mathbf{x}_N = \begin{bmatrix} \mathbf{x}_o \\ \mathbf{x}_e \end{bmatrix}, \tag{7}$$

with $\mathbf{x}_o \in \mathbb{C}^{\frac{N}{2}}$ the odd and $\mathbf{x}_e \in \mathbb{C}^{\frac{N}{2}}$ the even indexed part of $\mathbf{x}_N$. Introducing $\mathbf{y}_1, \mathbf{y}_2 \in \mathbb{C}^{\frac{N}{2}}$ we can formulate the DFT as follows:

$$\begin{aligned} \mathbf{y}_N &= \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{bmatrix} = \mathbf{F}_N \cdot \mathbf{x}_N = \mathbf{F}_N \cdot \underbrace{\mathbf{P}_N^T \cdot \mathbf{P}_N}_{\mathbf{I}_N} \cdot \mathbf{x}_N \qquad (8) \\ &= \begin{bmatrix} \mathbf{I}_{\frac{N}{2}} & \mathbf{D}_{\frac{N}{2}} \\ \mathbf{I}_{\frac{N}{2}} & -\mathbf{D}_{\frac{N}{2}} \end{bmatrix} \begin{bmatrix} \mathbf{F}_{\frac{N}{2}} & \mathbf{0} \\ \mathbf{0} & \mathbf{F}_{\frac{N}{2}} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{x}_o \\ \mathbf{x}_e \end{bmatrix} \end{aligned}$$

Defining $\mathbf{y}_o = \mathbf{F}_{\frac{N}{2}} \cdot \mathbf{x}_o$ and $\mathbf{y}_e = \mathbf{F}_{\frac{N}{2}} \cdot \mathbf{x}_e$ we obtain:

$$\mathbf{y}_1 = \mathbf{y}_o + \mathbf{D}_{\frac{N}{2}} \cdot \mathbf{y}_e = \mathbf{y}_o + \mathbf{d}_{\frac{N}{2}} \cdot *\mathbf{y}_e \tag{9}$$

$$\mathbf{y}_2 = \mathbf{y}_o - \mathbf{D}_{\frac{N}{2}} \cdot \mathbf{y}_e = \mathbf{y}_o - \mathbf{d}_{\frac{N}{2}} \cdot *\mathbf{y}_e \tag{10}$$

Here $\mathbf{d}_{\frac{N}{2}}$ is the vectorized diagonal of $\mathbf{D}_{\frac{N}{2}}$ and $(\cdot *)$ denotes component-wise vector multiplication. Taking into account that the two products in Eq. (9) and Eq. (10) are identical, altogether $\frac{N}{2}$ multiplications and $2 \cdot \frac{N}{2}$additions are required. It is easy to see that a DFT of size N can be executed by two DFTs of size $\frac{N}{2}$. For $N = 2^m$ this process can be recursively repeated $m = \log_2 N$ times. This leads to an overall complexity of $\frac{N}{2} \log_2 N$ multiplications and $N \log_2 N$ additions, which represents the complexity of the FFT.

C. Factorization of the DFT matrix

To obtain alternative factorizations of the DFT matrix, the number of recursions described in the previous section can be modified. Introducing a new parameter k we can define the number of performed recursions as $m-k$ with $N = 2^m$ giving the size of the initial DFT matrix. An example for $m = 4$ and $k = 2$ is given below.

Example 1:

$$\boldsymbol{y}_N = \underbrace{\begin{bmatrix} \boldsymbol{I}_8 & \boldsymbol{D}_8 \\ \boldsymbol{I}_8 & -\boldsymbol{D}_8 \end{bmatrix}}_{\boldsymbol{B}_2} \cdot \underbrace{\begin{bmatrix} \boldsymbol{I}_4 & \boldsymbol{D}_4 & 0 & 0 \\ \boldsymbol{I}_4 & -\boldsymbol{D}_4 & 0 & 0 \\ 0 & 0 & \boldsymbol{I}_4 & \boldsymbol{D}_4 \\ 0 & 0 & \boldsymbol{I}_4 & -\boldsymbol{D}_4 \end{bmatrix}}_{\boldsymbol{B}_1} \cdot \underbrace{\begin{bmatrix} \boldsymbol{F}_{\frac{N}{4}} & 0 & 0 & 0 \\ 0 & \boldsymbol{F}_{\frac{N}{4}} & 0 & 0 \\ 0 & 0 & \boldsymbol{F}_{\frac{N}{4}} & 0 \\ 0 & 0 & 0 & \boldsymbol{F}_{\frac{N}{4}} \end{bmatrix}}_{\boldsymbol{F}_B} \cdot \boldsymbol{x}_p$$

For arbitrary m and k the resulting block submatrices in $\mathbf{F}_B$ are of size $(2^k \times 2^k)$ and can be executed in parallel as independent DFTs (or FFTs). With the length of the input vector $\mathbf{x}_N$ given as $N = 2^m$ the number of block submatrices in $\mathbf{F}_B$ equals $2^m/2^k = 2^{m-k}$.

After computing the 2^{m-k} independent DFTs of $\mathbf{F}_B$, $m-k$ butterfly matrices $\mathbf{B}_i$ have to be applied to finish the calculation. The entire DFT is then given as

$$\mathbf{y}_N = \mathbf{B}_{m-k} \cdot \mathbf{B}_{m-k-1} \cdots \mathbf{B}_1 \cdot \mathbf{F}_B \cdot \mathbf{x}_P, \tag{11}$$

with $\mathbf{x}_p$ the $(m-k)$-times permuted input vector $\mathbf{x}_N$.

D. Construction of Butterfly Matrices

Increasing the number of recursions $m-k$ will also increase the number of butterfly matrices $\mathbf{B}_i$. While the independent DFT submatrices in $\mathbf{F}_B$ can be processed independently, the butterfly matrices can be treated in different ways:

- As sequence of matrix multiplications
$$\prod_{i=0}^{m-k-1} \mathbf{B}_{m-k-i}$$
- As one matrix multiplication with the butterfly product matrix
$$\mathbf{B} = \prod_{i=0}^{m-k-1} \mathbf{B}_{m-k-i}$$

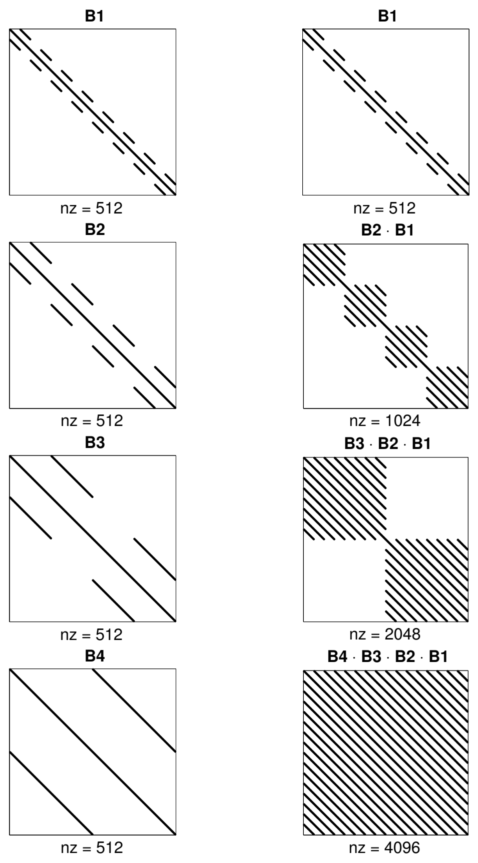

Fig. 1. The single butterfly matrices $\mathbf{B}_i$ on the left hand side; their products on the right hand side.

- As partly combined matrix multiplication with the products

$$\prod_{i=0}^{p_1} \mathbf{B}_{m-k-i} \prod_{i=p_1-1}^{p_2} \mathbf{B}_{m-k-i} \prod_{i=p_r-1}^{m-k-1} \mathbf{B}_{m-k-i}$$

  with arbitrarily chosen integer numbers p_l fulfilling $(1 < p_1 < p_2 < \cdots < p_r < m-k)$.

Example 2: The following example illustrates the construction of the butterfly matrices: For the given parameters $m = 8$ and $k = 4$ already $2^{m-k} = 2^4 = 16$ DFTs are executed by $\mathbf{F}_B$. The structure of the remaining butterfly matrices $\mathbf{B}_i$ is depicted in Figure 1 on the left hand side (non-zero elements are depicted as lines). The product of the butterfly matrices for increasing i is shown on the right hand side. The matrices are all of dimension $2^8 \times 2^8$, the variable nz in the figure shows the number of non-zero elements. Focusing on the butterfly product matrix in the bottom right, we can find that only $nz = 4096$ of all $2^8 \cdot 2^8 = 65536$ elements are non-zero. This means, every 16-th diagonal of the butterfly product matrix is non-zero, since the other multiplications have already been performed by the block diagonal matrix $\mathbf{F}_B$. In general, every 2^k-th diagonal is non-zero. This regular sparse structure can be exploited to reduce the computational effort of the matrix-vector multiplications.

IV. Complexity

In this section we will compare different separations of the butterfly matrix in terms of computational complexity, which will be measured as the number of required multiplications.

The direct application of the DFT matrix to an input vector requires $(N-1) \cdot (N-1)$ multiplications, i.e. the product of all rows and columns, except the first row/column, which is all ones:

$$M_{DFT} = N^2 - 2N + 1 = 2^m \cdot 2^m - 2 \cdot 2^m + 1$$

The FFT is based on butterfly operations, which require $N/2 = 2^{m-1}$ multiplications each (see section III-B). Since we have m butterfly matrices, we obtain

$$M_{FFT} = m \cdot 2^{m-1} = \frac{N}{2} \log_2 N.$$

In the following we will examine the complexities of four different separations, which lie between M_{DFT} and M_{FFT}.

A. Separation A

We use $(\mathbf{B}_{m-k} \cdots \mathbf{B}_1 \cdot \mathbf{F}_B)$, where the submatrices of $\mathbf{F}_B$ are executed as independent DFTs and the butterfly matrices are separately executed one by one. The number of the DFT blocks in $\mathbf{F}_B$ is 2^{m-k} and each DFT needs $2^k \cdot 2^k - 2 \cdot 2^k + 1$ multiplications. The number of butterfly matrices is $m-k$ and each butterfly needs 2^{m-1} multiplications. This leads to

$$M_A = (m-k) \cdot 2^{m-1} + (2^k \cdot 2^k - 2 \cdot 2^k + 1) \cdot 2^{m-k}.$$

B. Separation B

We use $\mathbf{B} \cdot \mathbf{F}_B$, where the submatrices of $\mathbf{F}_B$ are again executed as independent DFTs, and $\mathbf{B}$ is the butterfly product matrix. Since only every 2^k-th diagonal is occupied, the number of required multiplications for $\mathbf{B}$ is $(2^m - 1) \cdot (\frac{2^m}{2^k} - 1)$. The number of multiplications for $\mathbf{F}_B$ is identical to separation A, such that

$$M_B = (2^m - 1) \cdot (\frac{2^m}{2^k} - 1) + (2^k \cdot 2^k - 2 \cdot 2^k + 1) \cdot 2^{m-k}.$$

C. Separation C

We again use $\mathbf{B} \cdot \mathbf{F}_B$, but now the submatrices of $\mathbf{F}_B$ are executed as independent FFTs. There are $k \cdot 2^{k-1}$ multiplications required for each FFT, and totally there are 2^{m-k} FFT blocks, so the number of multiplications is given as

$$M_C = (2^m - 1) \cdot (\frac{2^m}{2^k} - 1) + k \cdot 2^{k-1} \cdot 2^{m-k}.$$

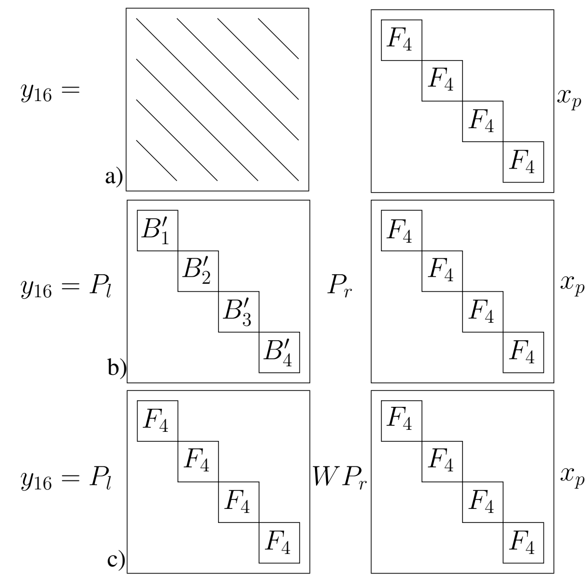

Fig. 2. Parallel FFT following Separation D.

D. Separation D

A special case of separation C is separation D, which describes a partly parallel implementation of the original FFT for the case $k = m/2$. The factorization is given as

$$\mathbf{y}_N = \mathbf{P}_l \cdot \mathbf{F}_B \cdot \mathbf{W} \cdot \mathbf{P}_r \cdot \mathbf{F}_B \cdot \mathbf{x}_p, \tag{12}$$

with $\mathbf{W}$ a twiddle factor diagonal matrix and $\mathbf{P}_l$ and $\mathbf{P}_r$ permutation matrices that sort the rows and columns of a matrix, such that $\mathbf{P}_l \cdot \mathbf{F}_B \cdot \mathbf{W} \cdot \mathbf{P}_r = \mathbf{B}$ is the butterfly product matrix. To clarify this, an example is given in Figure 2 for $m = 4$ and $k = 2$.

Example 3: In a) we can see the graphical representation of example 2, with the butterfly product matrix $\mathbf{B}$ on the left and the block FFT matrix $\mathbf{F}_B$ on the right hand side. By sorting rows and columns of $\mathbf{B}$ we can transform the sparse diagonal matrix $\mathbf{B}$ to a block diagonal matrix, as shown in b). This leads to a matrix based notation of the Jeon-Reeves FFT algorithms [9]. As the blocks $(\mathbf{B}'_1, \cdots, \mathbf{B}'_4)$ are not identical, constant twiddle factors are extracted from the matrices, which transfers the block submatrices to DFT matrices, as shown in c). The extracted twiddle factors are collected in $\mathbf{W}$.
The computational complexity of this separation is determined by the number of FFTs $(2 \cdot 2^k)$ and the additional twiddle factor multiplications $(N = 2^m)$:

$$M_D = 2 \cdot 2^k \cdot k \cdot 2^{k-1} + 2^m = 2^m(1 + m/2) \tag{13}$$

The number of required multiplications for separations A-D are given in Figure 3 for $m = 8$ and varying k as multiples of M_{FFT}. The upper bound is defined by the DFT complexity, the lower bound shows the FFT complexity. Note that separation D is only defined for $k = m/2 = 4$.

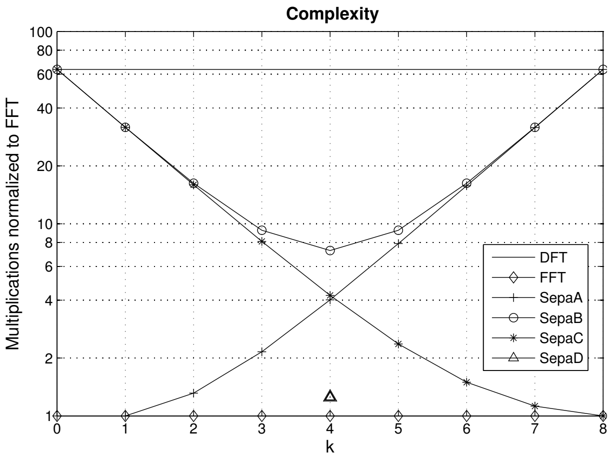

Fig. 3. Complexities for DFT, FFT and various separations.

V. Implementation Issues

Here we will discuss possible implementations of the presented separations on different hardware architectures. The discussion is of a qualitative character, i.e. implementation details like memory access, interconnection complexity or inter processor communication time are not considered here.

A. Separation A

Matrix-vector processors as special kind of SIMD (Single Instruction Multiple Data) architectures are discussed as possible platforms for SDR [10]. As an example we will show an implementation of separation A using a general systolic array structure for matrix multiplications, like depicted in Figure 4.

With $m = 5$ and $k = 2$ we have to perform 8 DFTs of length 4 in $\mathbf{F}_B$. To obtain the matrix-matrix structure we separate the input vector $\mathbf{x}_p$ into 8 parts $(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_8)$ of length 4, which means that all $\mathbf{F}_4 \cdot \mathbf{x}_j$ with $j = 1, ..., 8$ can be computed independently. The multiple matrix-vector products can then be described as matrix-matrix product $\mathbf{F}_4 \cdot \mathbf{X}_p$ with $\mathbf{X}_p = [\mathbf{x}_1\ \mathbf{x}_2 \cdots \mathbf{x}_8]$.

The remaining butterfly operations are performed according to Eq. (9) and Eq. (10). The even numbered columns of the product $\mathbf{F}_4 \cdot \mathbf{X}_p$ are multiplied component-wise by $\mathbf{d}_4$ and then added/subtracted to the corresponding odd numbered columns. For example, the second column is multiplied by $\mathbf{d}_4$ and then added/subtracted to the first column, resulting in the intermediate vectors $\mathbf{y}_{11}$ and $\mathbf{y}_{21}$ that are stored in the registers of the first and second column of the processor array. In the next step these intermediate results are multiplied by $\mathbf{d}_8$ and added/subtracted to the other intermediate results. After $m-k$ steps the registers contain the Fourier transform of $\mathbf{x}$.

The computational complexity for this approach is only slightly increased for small values of k (compare Figure 3), but a high speed partly parallel Fourier transformation can be realized.

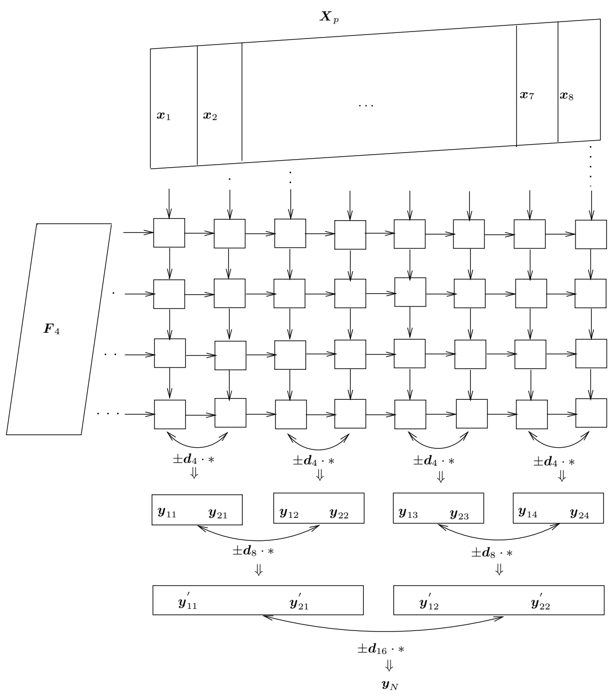

Fig. 4. Matrix-vector processor based implementation structure.

B. Separation B

For generic matrix-vector architectures the DFT can be described as simple matrix-vector product using separation B. In this case the number of required multiplications is increased considerably compared to the FFT, but nevertheless, it is interesting to see the significant decrease of the number of multiplications (e.g. for $k = 3, 4, 5$ in Figure 3) compared to the DFT, just by separating the DFT matrix into a product of the two matrices $\mathbf{B}$ and $\mathbf{F}_B$.

C. Separation C

To match the real time requirements in SDR systems, general purpose processors are often combined with hardware accelerators, like e.g. for the Fourier transform. As the flexibility of these accelerators regarding to the DFT size might be limited, separation C can be used to map DFTs of any size onto these engines. For example in a systems with FFT accelerators of size 64 and a required FFT length of 256 (this corresponds to $m = 8$ and $k = 6$), the block DFTs in $\mathbf{F}_B$ can be realized on the accelerators while the remaining 2 butterfly operations are executed as a simple matrix vector multiplication on a generic host processor. This leads to a major flexibility regarding the DFT size of the used accelerators in context of the required DFT size of the specific application. Furthermore existing architectures with fixed length FFT accelerators can be easily reused.

Depending on the difference of m and k, the mathematical complexity is increased (compare Figure 3). For the example the number of required multiplications is increased by a factor 1.5, but the 4 FFTs of size 64 can be executed in parallel, if more than one accelerator is available in the system.

D. Separation D

The FFT proposed by Jeon and Reeves is especially suitable for multi processor architectures [9]. However the extraction of the twiddle factors in separation D also enables the use of homogeneous parallel FFT cores in an SDR system, as the resulting block submatrices are all identical. This means an N-point DFT can be separated into $2\sqrt{N}$ FFTs of size $\sqrt{N}$. The first $\sqrt{N}$ FFTs are computed using the specialized processor cores, the results are then permuted and multiplied by $\mathbf{W}$. The remaining $\sqrt{N}$ blocks are again computed using the FFT cores.

VI. Conclusions

There is no optimum FFT implementation that fits to all hardware architectures. Each architecture has its specific properties, which makes an adaptation of the algorithms necessary. The presented separations of the DFT matrix provide a flexibility in terms of different algorithmic structures. Depending on the architecture, the most suitable separation can be chosen, e.g. separation A for matrix processors or separation C for architectures using FFT hardware accelerators. Depending on the chosen parameters the computational complexity in terms of multiplications is only slightly increased, however, some architectures may benefit from the different algorithmic structure (e.g. addressing, bus communication, memory transfers). The behavior of the presented separations on existing hardware architectures is a topic for further research.

References

[1] J.W. Cooley, J.W. Tukey: An algorithm for the machine calculation of complex Fourier series, *Mathematics of Computation*, vol. 19, pp. 297-301, 1965.

[2] B.S. Son, B.G. Jo, M.H. Sunwoo, S.K. Yong: A high-speed FFT processor for OFDM systems, *Proc. of the IEEE International Symposium on Circuits and Systems (ISCAS2002)*, Vol. 3, pp. 281-284, May 2002.

[3] M. Dillinger, K. Madani, N. Alonistioti: Software Defined Radio - Architectures, Systems and Functions, *Wiley Series in Software Radio*, John Wiley and Sons Ltd., 2003.

[4] B. Heyne, J. Götze: A Pure Cordic Based FFT for Reconfigurable Digital Signal Processing. *12th European Signal Processing Conference (Eusipco2004)*, Vienna, September 2004.

[5] U. Berthold, A.-R. Riehmeier, F.K. Jondral: Spectral partitioning for modular software defined radio, *59th Vehicular Technology Conference (VTC2004)*, 2004.

[6] K. Hueske, J. Geldmacher, J. Götze, E. Coersmeier: Multi Core Processing for Software Radio Channel Decoder, *5th Karlsruhe Workshop on Software Radios (WSR 2008)*, Karlsruhe, Germany, March 2008.

[7] C. Van Loan: Computational Frameworks for the Fast Fourier Transform, *Frontiers in applied mathematics*, Society for Industrial and Applied Mathematics, Philadelphia, 1992.

[8] F. Franchetti et al.: Discrete Fourier Transform on Multicore, *IEEE Signal Processing Magazine*, Vol.26, pp. 90-102, November 2009.

[9] C.H. Jeon, A.P. Reeves: The fast Fourier transform on a multicluster computer architecture, *Proceedings of the 19th Annual Hawaii International Conference on System Sciences*, pp. 103-110, 1986.

[10] M. Schoenes, S. Eberli et al.: A novel SIMD DSP architecture for software defined radio, *Proc. of the 46th IEEE International Midwest Symposium on Circuits and Systems (MWSCAS2003)*, Vol. 3, pp. 1443-1446, December 2003.