CONSTRAINTS ON THE SIMD VECTORIZATION OF RADIX-2 AND MIXED-RADIX FFTS

Peter Westermann, Hartmut Schröder

CAS Lab, Technische Universität Dortmund
Otto-Hahn-Str. 4, 44221 Dortmund, Germany
{peter.westermann,hartmut.schroeder}@tu-dortmund.de

ABSTRACT

Single instruction, multiple data (SIMD) signal processors for wireless communications require efficient vectorized algorithms for radix-2 and mixed-radix Fast Fourier Transforms (FFTs). Especially, mixed-radix FFTs are challenging for a processor that operates on power-of-two length vectors. We analyze the vectorization of pure radix-2 and mixed-radix FFTs and demonstrate that both FFTs have different constraints for an efficient vectorization. The radix-2 FFT can be efficiently vectorized if the FFT length is at least twice the vector length while the mixed-radix FFT requires the FFT length to be a multiple of the squared vector length.

1. INTRODUCTION

In recent years, the development of signal processing architectures for wireless communications shifted from ASICs towards programmable processors. This approach offers better flexibility, especially for multi-standard radio systems, improves hardware reuse, and enables to share information between different processing stages. However, especially baseband signal processing is a computationally demanding, hard real-time problem. Next to that, the algorithmic complexity is still increasing as novel techniques are introduced and data rates increase.

As many regular baseband algorithms feature an innate degree of data level parallelism, Single Instruction Multiple Data (SIMD) vector processing appears to be a viable approach to satisfy the demands of real-time processing. Hence, several parallel baseband processor architectures like Linköping University's Single Instruction Multiple Tasks (SIMT) architecture [10, 11], Sandbridge's Sandblaster [6], NXPs Embedded Vector Processor (EVP) [15], and University of Michigan's Software On-Demand Architecture (SODA) [8, 9] have been proposed.

One important task in baseband signal processing is the Fast Fourier Transform (FFT), which is the main part of Orthogonal Frequency Division Multiple Access (OFDMA) and Single Carrier Frequency Division Multiple Access (SC-FDMA) systems. While FFT sizes for OFDMA are mostly powers of two, SC-FDMA in UMTS LTE [13] requires FFTs based on powers of two, three, and five; this is especially challenging on a SIMD architecture operating on power-of-two length vectors. Hence, we analyzed the SIMD vectorization of radix-2 and mixed-radix FFTs. Our goal was to determine constraints on the ratio between FFT length and SIMD vector length for an efficient implementation on a SIMD signal processor for baseband processing.

Based on a mathematical notation using the Kronecker matrix product (Section 2), different FFT algorithms are discussed. The contributions of this paper are as follows:

- Using definitions of vectorizable formulas, we demonstrate that the pure radix-2 FFT can be efficiently vectorized if the FFT size is at least twice the vector length V (section 3). Furthermore, we show that mixed-radix FFTs that contain a factor of $2 \cdot V$ and other non-power-of-two factors cannot be efficiently vectorized.
- We present a general FFT algorithm decomposition for $N_{\mathrm{DFT}} = V^2 \cdot M$ that allows an efficient vectorization (section 4). We decompose the FFT into three stages. Two stages perform V-point FFTs and one stage performs an M-point FFT. The M-point FFT contains all non-power-of-two factors and is completely independent of the vector length.
- We analyze the permutation operations in our algorithms and discuss requirements for the permutation unit of a SIMD processor (section 5). Our focus is on the width of the required permutation network (e.g. one data vector or two data vectors).
- We illustrate our findings with performance results for the Embedded Vector Processor (EVP)[15] in section 6 as an extension of the results published in [16].

2. MATRIX REPRESENTATION OF THE FFT

The Discrete Fourier Transform (DFT) and its fast variants may be expressed by matrix-vector multiplications. For example, the Fourier transform of an N-point data vector $\mathbf{x} \in \mathbb{C}^N$ may be written as

$$\mathbf{y} = \mathbf{W}_N \cdot \mathbf{x}.$$

In the following, we denote all elements (k,l) of a matrix $\mathbf{A}$ as $[\mathbf{A}](k,l)$; a single matrix element is symbolized by a_{kl}. The elements of the N-point DFT-matrix are defined as

$$[\mathbf{W}_N](k,l) = \omega_N^{kl}$$

with $\omega_N = exp(-2\pi i/N)$. Any non-prime DFT-matrix may be decomposed into smaller-sized DFTs using a formalism based on Kronecker products and permutation matrices (see e.g. [1, 5, 14]). The Kronecker or tensor product of two matrices $\mathbf{A}$ and $\mathbf{B}$ is defined as follows:

$$\mathbf{A} \otimes \mathbf{B} = \left(a_{k,l} \cdot \mathbf{B}\right)$$

For example, a Kronecker product of a 2×2 matrix $\mathbf{A}$ with an arbitrary matrix $\mathbf{B}$ results in the following matrix:

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11} \cdot \mathbf{B} & a_{12} \cdot \mathbf{B} \\ a_{21} \cdot \mathbf{B} & a_{22} \cdot \mathbf{B} \end{bmatrix}$$

Of special interest are Kronecker products with a $q \times q$ identity matrix $\mathbf{I}_q$. As an example, the Kronecker product $\mathbf{A} \otimes \mathbf{I}_q$ implies that the transformation defined by matrix $\mathbf{A}$ is applied to blocks of q consecutive elements in parallel.

Further, the matrix conjugation of $\mathbf{A}$ by $\mathbf{X}$ can be described by:

$$\mathbf{X}^T \cdot \mathbf{A} \cdot \mathbf{X} = \mathbf{A}^{\mathbf{X}} \tag{1}$$

Additionally, we define a $pq \times pq$ stride permutation matrix by the following equation:

$$\left[\mathbf{P}_q^p\right](j,k) = \begin{cases} 1 & \text{for } j = r \cdot p + s,\ k = s \cdot q + r, \\ 0 & \text{otherwise} \quad (0 \le r < q, 0 \le s < p) \end{cases}$$

A multiplication with permutation matrix $\mathbf{P}_q^p$ results in a stride by q reordering of elements [7], i.e. the elements of input vector $\mathbf{x} = \left(x_0, x_1, \ldots, x_{pq-1}\right)^T$ are reordered as $\mathbf{y} = \left(x_0, x_q, x_{2q}, \ldots, x_{pq-q}, x_1, x_{q+1}, \ldots, x_{pq-1}\right)^T$.

Equation (2) defines the basic structure of an FFT algorithm using Kronecker products and permutation matrices. The equation decomposes a pq-point DFT into a p-point DFT block, a multiplication by a diagonal twiddle factor matrix $\mathbf{D}_q^p$ (see (3)), a permutation operation, and a q-point DFT block.

$$\mathbf{W}_{pq} = \left(\mathbf{W}_q \otimes \mathbf{I}_p\right) \cdot \mathbf{P}_q^p \cdot \mathbf{D}_q^p \cdot \left(\mathbf{W}_p \otimes \mathbf{I}_q\right) \tag{2}$$

$$\left[\mathbf{D}_q^p\right](j,k) = \begin{cases} \omega_{pq}^{s \cdot m} & \text{for } j = k = s \cdot q + m, \\ 0 & \text{otherwise} \quad (0 \le m < q, 0 \le s < p) \end{cases} \tag{3}$$

Different FFT algorithms may be derived from (2) by repeatedly decomposing into smaller DFTs and by formula manipulations using mathematical identities. Some useful identities are defined by the following rules; others may be found in [5, 14]. Here, we assume that $\mathbf{A}$ is a $p \times p$ matrix and $\mathbf{B}$ and $\mathbf{C}$ are $q \times q$ matrices.

$$\mathbf{P}_q^p \cdot (\mathbf{A} \otimes \mathbf{B}) = (\mathbf{B} \otimes \mathbf{A}) \cdot \mathbf{P}_q^p \tag{4}$$

$$(\mathbf{B} \otimes \mathbf{I}_r) \cdot (\mathbf{C} \otimes \mathbf{I}_r) = (\mathbf{B} \cdot \mathbf{C}) \otimes I_r \tag{5}$$

$$\mathbf{P}_q^{pr} = \left(\mathbf{P}_q^p \otimes \mathbf{I}_r\right) \cdot \left(\mathbf{I}_p \otimes \mathbf{P}_q^r\right) \tag{6}$$

$$\mathbf{P}_{qr}^p = \left(\mathbf{I}_q \otimes \mathbf{P}_r^p\right) \cdot \left(\mathbf{P}_q^p \otimes \mathbf{I}_r\right) \tag{7}$$

$$\mathbf{A} \otimes (\mathbf{B} \cdot \mathbf{C}) = (\mathbf{A} \otimes \mathbf{B}) \cdot (\mathbf{A} \otimes \mathbf{C}) \tag{8}$$

3. VECTORIZATION OF THE RADIX-2 FFT

Below, we first introduce basic vectorizable formulas that represent operations on a SIMD signal processor [5]. Using this notation, we describe the SIMD vectorization of short radix-2 FFTs. Next, we extend the short radix-2 FFT to the mixed-radix case.

3.1 Vectorizable Formulas for Operations on a SIMD DSP

A SIMD signal processor operates on long data vectors. SIMD processing units operate on all elements of data vectors in a uniform manner; arbitrary element access is usually not supported. Using the Kronecker product, we can describe SIMD operations on vectors of length V by the following formula:

$$\mathbf{A} \otimes \mathbf{I}_V \tag{9}$$

Here, $\mathbf{A}$ is an arbitrary matrix. The vectorization can be done by generating scalar code for the linear transform defined by $\mathbf{A}$ and replacing all operations with vector operations on V elements. Any multiplication with a diagonal matrix is also a vectorizable formula; e.g. products with twiddle factor matrices $\mathbf{D}_q^p$ may be vectorized by replacing scalar multiplications with multiplications with complex twiddle factor vectors.

As most signal processing for wireless communications is done on complex I/Q signals, many processors support complex multiplication and complex data types (e.g. [12, 15]). Hence, in the following, we assume that all matrices are complex. Support of complex operations effectively halves the vector length as consecutive elements are treated as real and imaginary parts of a single value. If a processor does not support complex operations, the FFT requires a further transformation of interleaved complex data to block-interleaved complex data [5] with block-size V (i.e. real and imaginary parts are stored in adjacent data vectors).

Next to SIMD processing units, a SIMD signal processor usually also contains a permutation unit that enables permutations of vector elements. Most permutation units allow computing permutations of elements of a *single* vector or a *pair* of vectors. Examples for such permutations are given in (10). A more detailed analysis of the required permutations for FFTs and the effect on the layout of a permutation unit is carried out in section 5.

$$\mathbf{P}_2^2 \otimes \mathbf{I}_{\frac{V}{2}}, \quad \mathbf{P}_4^2 \otimes \mathbf{I}_{\frac{V}{4}}, \quad \mathbf{P}_b^a \otimes \mathbf{I}_{\frac{V}{ab}}, \quad \mathbf{I}_{\frac{V}{ab}} \otimes \mathbf{P}_b^a \tag{10}$$

Another example, the permutation $\mathbf{P}_V^V$ on V data vectors, can be carried out in $log_2(V)$ permutation stages with each stage permuting elements of pairs of vectors. A special case of (9) is described by the following basic vectorizable formula:

$$\mathbf{P}_b^a \otimes \mathbf{I}_V \tag{11}$$

Formula (11) defines a permutation of complete vectors. In this case, no permutation operations are needed - only the addressing of data vectors needs to be adjusted.

3.2 Short Radix-2 FFT

If we want to perform the FFT decomposition described by (2) on vectors of data, we have to be able to write all smaller sized FFT stages as Kronecker products with $V \times V$ identity matrices (e.g. $\mathbf{W}_k \otimes \mathbf{I}_V$). Furthermore, permutations should be either permutations of complete vectors or vectorizable permutations on pairs of vectors. For the pure radix-2 FFT, the first claim only allows FFT sizes that are at least twice the vector length. A DFT defined by $\mathbf{W}_{2 \cdot V}$ may be decomposed by applying (2) repeatedly:

$$\begin{aligned} \mathbf{W}_{2 \cdot V} &= (\mathbf{W}_V \otimes \mathbf{I}_2) \cdot \mathbf{P}_V^2 \cdot \mathbf{D}_V^2 \cdot (\mathbf{W}_2 \otimes \mathbf{I}_V) \\ &= \left(\mathbf{W}_{\frac{V}{2}} \otimes \mathbf{I}_4\right) \cdot \left(\mathbf{P}_{\frac{V}{2}}^2 \otimes \mathbf{I}_2\right) \cdot \left(\mathbf{D}_{\frac{V}{2}}^2 \otimes \mathbf{I}_2\right) \\ &\quad \cdot (\mathbf{W}_2 \otimes \mathbf{I}_V) \cdot \mathbf{P}_V^2 \cdot \mathbf{D}_V^2 \cdot (\mathbf{W}_2 \otimes \mathbf{I}_V) \\ &= \left(\mathbf{W}_{\frac{V}{4}} \otimes \mathbf{I}_8\right) \cdot \left(\mathbf{P}_{\frac{V}{4}}^2 \otimes \mathbf{I}_4\right) \cdot \left(\mathbf{D}_{\frac{V}{4}}^2 \otimes \mathbf{I}_4\right) \\ &\quad \cdot (\mathbf{W}_2 \otimes \mathbf{I}_V) \cdot \left(\mathbf{P}_{\frac{V}{2}}^2 \otimes \mathbf{I}_2\right) \cdot \left(\mathbf{D}_{\frac{V}{2}}^2 \otimes \mathbf{I}_2\right) \\ &\quad \cdot (\mathbf{W}_2 \otimes \mathbf{I}_V) \cdot \mathbf{P}_V^2 \cdot \mathbf{D}_V^2 \cdot (\mathbf{W}_2 \otimes \mathbf{I}_V) \end{aligned} \tag{12}$$

The decomposition in (12) leads to the self-sorting decimation-in-frequency FFT [14]; the full decomposition may be written as:

$$\mathbf{W}_{2 \cdot V} = \prod_{i=0}^{\log_2(V)} \left(\left(\mathbf{P}_{2^i}^2 \otimes \mathbf{I}_{V/2^i}\right) \left(\mathbf{D}_{2^i}^2 \otimes \mathbf{I}_{V/2^i}\right) (\mathbf{W}_2 \otimes \mathbf{I}_V)\right) \tag{13}$$

Here, all FFT butterfly stages operate on full vectors ($\mathbf{W}_2 \otimes \mathbf{I}_V$). Furthermore, the FFT requires $\log_2(V)$ permutation stages on pairs of vectors.This is the minimum number of permutation stages to perform the necessary reordering of vector elements. Bigger radix-2 FFTs do not require further vector element permutations, but introduce permutations of full vectors. This can be seen by replacing all occurrences of V in equation (13) with $N \cdot V$ ($N \in 2^n$).

3.3 Short Mixed-radix FFT

An extension of the short radix-2 FFT for $2 \cdot V$ elements can be done by adding a further factor X that contains all non-power-of-two factors (i.e. X is not divisible by two). A decomposition of $\mathbf{W}_{2 \cdot V \cdot X}$ using (2) and (13) leads to the following equation:

$$\mathbf{W}_{2 \cdot V \cdot X} = \left(\prod_{i=0}^{\log_2(V)} \mathbf{T}_i\right) \cdot \mathbf{P}_{2 \cdot V}^X \cdot \mathbf{D}_{2 \cdot V}^X \cdot (\mathbf{W}_X \otimes \mathbf{I}_{2 \cdot V})$$

$$\mathbf{T}_i = \left(\mathbf{P}_{2^i}^2 \otimes \mathbf{I}_{\frac{VX}{2^i}}\right) \cdot \left(\mathbf{D}_{2^i}^2 \otimes \mathbf{I}_{\frac{VX}{2^i}}\right) \cdot (\mathbf{W}_2 \otimes \mathbf{I}_{V \cdot X}) \tag{14}$$

A closer look at the radix-2 stages defined by (14) shows that all butterfly stages operate on full vectors. However, the Kronecker product with $\mathbf{I}_{\frac{VX}{2^i}}$ in the permutation stage means that permutations are not done on elements of full vectors, but on blocks of X elements. For example, if we set $V = 8$ and $X = 3$, the required permutations will be $\left(\mathbf{P}_2^2 \otimes \mathbf{I}_{12}\right)$, $\left(\mathbf{P}_4^2 \otimes \mathbf{I}_6\right)$, and $\left(\mathbf{P}_8^2 \otimes \mathbf{I}_3\right)$. None of these permutations can be efficiently vectorized. Furthermore, the permutation $\mathbf{P}_{2 \cdot V}^X$ also cannot be vectorized. Hence, mixed-radix FFTs that only contain a factor $2 \cdot V$ cannot be efficiently mapped on SIMD vector operations.

4. GENERAL MIXED-RADIX FFT

As shown in the previous section, mixed-radix FFT sizes that can be written as $N_{\text{DFT}} = 2 \cdot V \cdot X$ cannot be efficiently vectorized as permutations do not adhere to vector boundaries. In the following, we will develop an FFT algorithm that solves this issue.

4.1 Algorithm Derivation

Our FFT algorithm is designed for $N_{\text{DFT}} = V \cdot M \cdot V$. Here, the factor M may take on arbitrary values and contains all non-power-of-two factors. We start by applying (2) twice to get a decomposition of $\mathbf{W}_{VMV}$ into three factors:

$$\begin{aligned}\mathbf{W}_{VMV} &= (\mathbf{W}_{VM} \otimes \mathbf{I}_V) \cdot \mathbf{P}^V_{MV} \cdot \mathbf{D}^V_{MV} \cdot (\mathbf{W}_V \otimes \mathbf{I}_{VM}) \\ &= (\mathbf{W}_V \otimes \mathbf{I}_M \otimes \mathbf{I}_V) \cdot \left(\mathbf{P}^M_V \otimes \mathbf{I}_V\right) \cdot \left(\mathbf{D}^M_V \otimes \mathbf{I}_V\right) \\ &\quad \cdot (\mathbf{W}_M \otimes \mathbf{I}_V \otimes \mathbf{I}_V) \cdot \mathbf{P}^V_{MV} \cdot \mathbf{D}^V_{MV} \cdot (\mathbf{W}_V \otimes \mathbf{I}_{VM})\end{aligned}$$

Using (4) and (5), we next move the permutation defined by $\mathbf{P}^M_V \otimes \mathbf{I}_V$ to the left. Then, we again apply (4) to move $\mathbf{P}^V_{MV}$.

$$\begin{aligned}\mathbf{W}_{VMV} &= \left(\mathbf{P}^M_V \otimes \mathbf{I}_V\right) \cdot (\mathbf{I}_M \otimes \mathbf{W}_V \otimes \mathbf{I}_V) \cdot \left(\mathbf{D}^M_V \otimes \mathbf{I}_V\right) \\ &\quad \cdot (\mathbf{W}_M \otimes \mathbf{I}_V \otimes \mathbf{I}_V) \cdot \mathbf{P}^V_{MV} \cdot \mathbf{D}^V_{MV} \cdot (\mathbf{W}_V \otimes \mathbf{I}_{VM}) \\ &= \underbrace{\left(\mathbf{P}^M_V \otimes \mathbf{I}_V\right) \cdot (\mathbf{I}_M \otimes \mathbf{W}_V \otimes \mathbf{I}_V) \cdot \mathbf{P}^V_{MV}}_{\mathbf{T}_{V_1}} \\ &\quad \cdot \underbrace{\left(\mathbf{I}_V \otimes \mathbf{D}^M_V\right) \cdot (\mathbf{I}_V \otimes \mathbf{W}_M \otimes \mathbf{I}_V)}_{\mathbf{T}_M} \cdot \underbrace{\mathbf{D}^V_{MV} \cdot (\mathbf{W}_V \otimes \mathbf{I}_{VM})}_{\mathbf{T}_{V_2}} \\ &= \mathbf{T}_{V_1} \cdot \mathbf{T}_M \cdot \mathbf{T}_{V_2}\end{aligned}$$

The formulas labeled as $\mathbf{T}_M$ and $\mathbf{T}_{V_2}$ can already be directly mapped on vector operations. However, $\mathbf{T}_{V_1}$ contains a not directly vectorizable permutation $\mathbf{P}^V_{MV}$ that needs to be further decomposed.

Using (7), we can split $\mathbf{P}^V_{MV}$ into two permutations. Next, we reorder operations using (8) and simplify using (1).

$$\begin{aligned}\mathbf{T}_{V_1} &= \left(\mathbf{P}^M_V \otimes \mathbf{I}_V\right) \cdot (\mathbf{I}_M \otimes \mathbf{W}_V \otimes \mathbf{I}_V) \cdot \mathbf{P}^V_{MV} \\ &= \left(\mathbf{P}^M_V \otimes \mathbf{I}_V\right) \cdot (\mathbf{I}_M \otimes \mathbf{W}_V \otimes \mathbf{I}_V) \\ &\quad \cdot \left(\mathbf{I}_M \otimes \mathbf{P}^V_V\right) \cdot \left(\mathbf{P}^V_M \otimes \mathbf{I}_V\right) \\ &= \left(\mathbf{I}_M \otimes \left((\mathbf{W}_V \otimes \mathbf{I}_V) \cdot \mathbf{P}^V_V\right)\right)^{\left(\mathbf{P}^V_M \otimes \mathbf{I}_V\right)}\end{aligned}$$

To emphasize similarities, we transform $\mathbf{T}_{V_2}$ to the same principle structure, leading to:

$$\begin{aligned}\mathbf{W}_{VMV} &= \left(\mathbf{I}_M \otimes \left((\mathbf{W}_V \otimes \mathbf{I}_V) \cdot \mathbf{P}^V_V\right)\right)^{\left(\mathbf{P}^V_M \otimes \mathbf{I}_V\right)} \\ &\quad \cdot \left(\mathbf{I}_V \otimes \mathbf{D}^M_V\right) \cdot (\mathbf{I}_V \otimes \mathbf{W}_M \otimes \mathbf{I}_V) \\ &\quad \cdot \mathbf{D}^V_{MV} \cdot (\mathbf{I}_M \otimes \mathbf{W}_V \otimes \mathbf{I}_V)^{\left(\mathbf{P}^V_M \otimes \mathbf{I}_V\right)}\end{aligned}$$

4.2 Algorithm Discussion

The algorithm defined by matrices $\mathbf{T}_M$, $\mathbf{T}_{V_1}$, and $\mathbf{T}_{V_2}$ performs all FFT stages on complete data vectors. The algorithm also only requires one single permutation of vector elements, defined by $\mathbf{P}^V_V$. This permutation can be carried out by $\log_2(V)$ stages with permutation operations on pairs of vectors as in the radix-2 case described in section 3.2. If the V-point DFT in $\mathbf{T}_{V_1}$ is further decomposed into a series of radix-2 FFT stages, the $\log_2(V)$ permutation stages can be merged with the $\log_2(V)$ radix-2 butterfly stages to enable a better schedule of the operations on the processor. We omitted the decomposition to enhance readability.

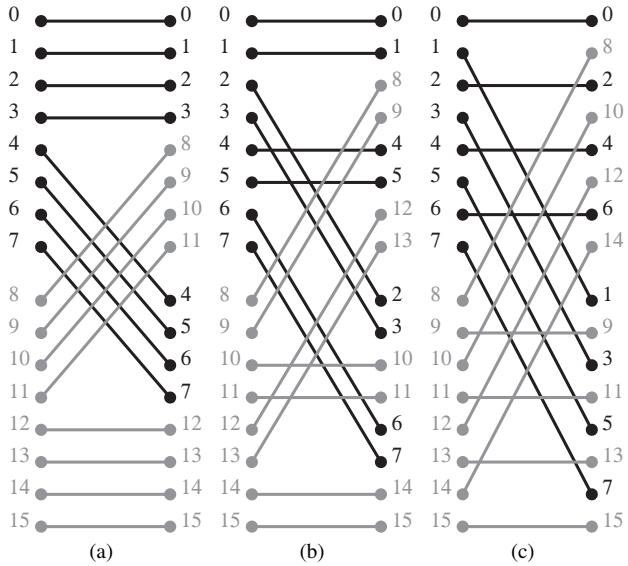

Figure 1: Basis permutations on pairs of vectors for vector length $V = 8$. Elements of the first and second input vectors are printed in black respectively gray.

Next to maximizing opportunities for vectorization and minimizing the overhead for permutations, the FFT algorithm has some major advantages regarding an efficient implementation on a SIMD vector processor. The V-point FFT blocks defined by $\mathbf{T}_{V_1}$ and $\mathbf{T}_{V_2}$ are mostly independent of M. M only influences the number of FFTs, the twiddle factors, and permutations of complete vectors. The vector permutations are defined by $\left(\mathbf{P}^V_M \otimes \mathbf{I}_V\right)$:

$$\begin{aligned}&(\mathbf{I}_M \otimes \mathbf{W}_V \otimes \mathbf{I}_V)^{\left(\mathbf{P}^V_M \otimes \mathbf{I}_V\right)} = \\ &\left(\mathbf{P}^M_V \otimes \mathbf{I}_V\right) \cdot (\mathbf{I}_M \otimes \mathbf{W}_V \otimes \mathbf{I}_V) \cdot \left(\mathbf{P}^V_M \otimes \mathbf{I}_V\right)\end{aligned}$$

The first permutation effects that every M-th data vector is selected as input for one FFT-stage, the second permutation stores the results at the correct positions. For the concrete algorithm implementation, this means that only pointer updates are dependent on M. Hence, it is possible to implement optimized programs for V-point FFTs once and later adjust for different values of M simply by changing parameters.

The M-point FFT stage defined by matrix $\mathbf{T}_M$ always operates on complete vectors. Therefore, a further decomposition into smaller DFTs has no impact on the vectorization, no matter how the decomposition is done. This simplifies the development of algorithms for different FFT sizes.

5. PERMUTATIONS FOR SIMD FFTS

The mixed-radix FFT algorithm requires one permutation of vector elements defined by $\mathbf{P}^V_V$. This permutation can be carried out in $\log_2(V)$ stages; each stage contains a permutation on a pair of vectors. Figure 1 shows the permutations that are required for vector length $V = 8$. For greater vector lengths, the principle structure of the permutation operations stays the same.

All required permutations share some common characteristics: First, half of the elements of each output vector are copied directly from the corresponding input vector. Second, the other half of the elements of each output vector is generated by permuting elements of the other input vector using butterfly permutations. The butterfly permutations in each stage exchange elements that differ in one bit position, starting with the most significant bit in Figure 1(a) and ending with the least significant bit in Figure 1(c).

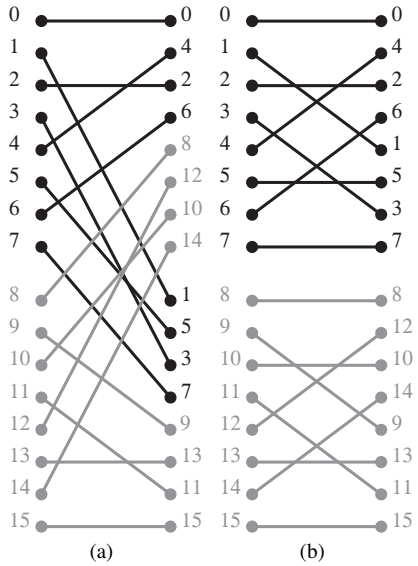

Figure 2: Additional permutation for last stage of 16-point FFT with $V = 8$. (a) shows the complete permutation on pairs of vectors, (b) can be used after the permutation in Figure 1(c) to perform the same operation on processors that only support permutations of one vector.

These characteristics of the required permutations mean that, while the stages actually define permutations on pairs of vectors, all permutations may as well be carried out on a permutation unit that supports only permutations on one input vector. In this case one permutation stage on a pair of vectors is split into two operations. Each operation performs a masked butterfly permutation, i.e. a vector mask defines elements that are not permuted, but instead copied from a second input vector.

The permutations in our radix-2 algorithm defined by equation (13) can be transformed into a similar structure as the permutations for the general mixed-radix case. If this is done, the last permutation stage (Figure 1(c) in the mixed-radix case) has to be replaced by a more complex permutation on pairs of vectors. All other permutations stay unchanged. As an example, Figure 2(a) depicts the last permutation stage required for a 16-point FFT with vector length $V = 8$. The new permutation operation cannot be split into two masked permutations on one vector. On a processor without support of permutations of pairs of vectors, the last stage can be realized by first applying the permutation in Figure 1(c) and then the permutation in Figure 2(b). Hence, an additional permutation stage is required compared to a processor with permutations on pairs of vectors.

Our findings may be summarized as follows: All permutations required for our FFT algorithms can be realized using a butterfly network for permutations. Furthermore, most permutations on pairs of vectors can be split into permutations on one vector. Only the short radix-2 FFT contains one permutation stage that cannot be directly split into two permutations on one input vector.

6. RESULTS ON THE EMBEDDED VECTOR PROCESSOR

We implemented our FFT algorithms on the EVP [15]. The DFT sizes have been selected from the range of DFTs required for SC-FDMA in UMTS LTE [13]; a more detailed analysis of the performance of this application on the EVP can be found in [16].

6.1 The Embedded Vector Processor

The EVP operates on 256-bit vectors and supports complex operations on 16+16 bit elements (16 bit real part, 16 bit imaginary part). Hence, the vector length is eight. The processor contains a vector arithmetic logic unit (ALU) and a vector multiply-accumulate (MAC) unit. For permutations, the EVP contains a crossbar permutation network that operates on one input vector. All operations on the EVP can execute conditionally on an element-by-element basis using a vector mask.

The processor operates at 300 MHz and supports a very long instruction word (VLIW) instruction set. For the FFT, this means that one multiplication, one addition or subtraction, and one permutation can be done in the same cycle — if all operations work on independent data. The EVP's theoretical peak performance is 30 GOP/s.

The EVP is programmed in the EVP-C language, which combines the C language for scalar operations with data types for data vectors and explicit vector operations.

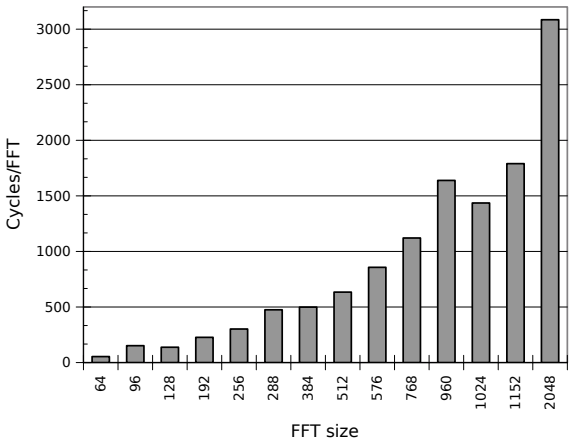

Figure 3: Clock cycles per FFT for different FFT sizes on the EVP (8-way complex vectors)

6.2 FFT Results

Figure 3 displays the performance of various FFT sizes on the EVP measured in clock cycles per FFT. The FFT sizes may be organized in four groups:

- Pure power-of-two FFTs with $N_{\mathrm{DFT}} \geq V^2$.
- Smaller power-of-two FFTs, i.e. the 16-point and 32-point FFTs.[1]
- Mixed-radix FFTs containing multiples of 3 and/or 5 with $N_{\mathrm{DFT}} = M \cdot V^2$.
- Mixed-radix FFTs with $N_{\mathrm{DFT}} \neq M \cdot V^2$, i.e. the 96-point and 288-point FFTs.

To illustrate the impact of the vectorization on the performance, Figure 4 displays the FFT performance in a normalized format in pseudo giga operations per second (pseudo GOP/s). We approximate the number of real-valued operations for an N_{DFT}-point FFT as $5 \cdot N_{\mathrm{DFT}} \log_2(N_{\mathrm{DFT}})$ based on [5]. Pseudo GOP/s then can be calculated as $5N_{\mathrm{DFT}} \log_2(N_{\mathrm{DFT}})/t$ with t measuring the runtime of one FFT in nano seconds.

Especially Figure 4 shows that FFT sizes, which do not permit the usage of our algorithms (i.e. the 96-point and 288-point FFTs), are quite inefficient compared to the other FFTs. To vectorize these FFTs it was necessary to introduce several additional permutation stages manually.

The differences between mixed-radix and pure power-of-two FFTs for $N_{\mathrm{DFT}} = M \cdot V^2$ in both figures can be explained by the different complexity of basic FFT butterfly stages. One radix-2 butterfly stage occupies the vector ALU and vector MAC unit for one

[1] The 16-point and 32-point FFTs have been omitted in Figure 3 as the cycle counts are too small to be distinguishable.

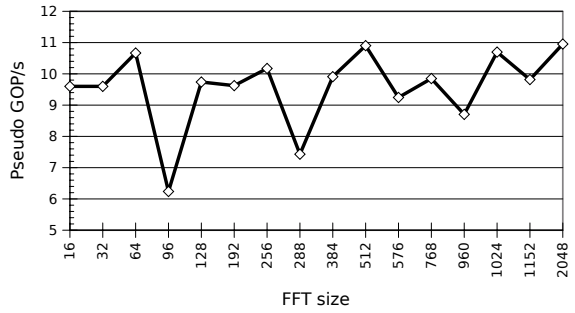

Figure 4: Pseudo GOP/s for different FFT sizes on the EVP (8-way complex vectors)

clock cycle per vector (including twiddle factor multiplications). In contrast, a radix-3 stage requires 2.33 cycles per vector on the vector MAC unit and a radix-5 stage 3.6 cycles per vector.

The increased performance of the 64-point FFT compared to the smaller power-of-two FFTs has two causes: First, the EVP only supports permutations on one data vector. For radix-2 FFT sizes smaller than the squared vector length V^2, processors that only support single vector permutations require one additional permutation stage (see section 5). Second, bigger FFTs allow capitalizing on VLIW, i.e. additions/subtractions, multiplications, and permutations of independent data vectors can be done in parallel. For small FFTs, there are not enough independent data vectors in each FFT stage to fully benefit from VLIW.

Differences in the performance of FFT sizes from the same class can be attributed to an unequal segmentation of the FFT: Due to a limited register file size, big FFTs must be split into segments of two or three FFT stages on a limited input data size (e.g. 8 vectors).

7. RELATED WORK

Franchetti and Püschel [2, 3, 5, 4] implemented an FFT compiler for short vector SIMD extensions of modern general purpose microprocessors as part of the SPIRAL library generator. They derive an efficient FFT algorithm for a given FFT size and target processor by automatically generating multiple algorithms, represented by mathematical formulas. These algorithms are translated into programs, executed, and timed. Searching the design space of mathematical formulas for the FFT, a suitable algorithm can be found. The key components of the framework are a set of rewriting rules and a set of vectorizable formulas [5]; our algorithm is based on the same rewriting rules.

In [3], a mixed-radix FFT algorithm for FFT sizes that are a multiple of the squared vector length is proposed. Besides that Franchetti and Püschel decompose complex operations into real-valued operations, the main difference to our algorithm is the partitioning of vector element permutations in the FFT stages. The complexity of the permutation stages is the same as in our algorithm.

8. CONCLUSION

We analyzed the problem of mapping radix-2 and mixed-radix FFTs on SIMD signal processors that support SIMD operations on vectors with V elements and permutation operations on single vectors or pairs of vectors. Our analysis shows that a radix-2 FFT can be efficiently vectorized on a SIMD signal processor if the FFT length is at least twice the vector length V. Mixed-radix FFTs can be efficiently vectorized if the FFT length is a multiple of the squared vector length V^2.

Both types of FFTs require $\log_2(V)$ permutation stages on pairs of vectors. In the mixed-radix case, all permutations on pairs of vectors can be converted into pairs of permutations on single vectors. In the radix-2 case, the last permutation stage cannot be split into permutations on single vectors. Hence, in this case, a permutation unit that directly supports permutations on pairs of vectors is beneficial.

The performance analysis of different FFT sizes on the EVP validates our results for the requirements on the FFT length for the radix-2 and mixed-radix FFT.

REFERENCES

[1] R. Bellman. *Introduction to Matrix Analysis*. McGraw-Hill, 1960.

[2] F. Franchetti and M. Püschel. A SIMD vectorizing compiler for digital signal processing algorithms. In *IPDPS '02: Proceedings of the 16th International Parallel and Distributed Processing Symposium*, 2002.

[3] F. Franchetti and M. Püschel. Short Vector Code Generation for the Discrete Fourier Transform. In *International Parallel and Distributed Processing Symposium (IPDPS'03)*, 2003.

[4] F. Franchetti and M. Püschel. SIMD vectorization of non-two-power sized FFTs. In *Proceedings of International Conference on Acoustics, Speech, and Signal Processing (ICASSP) 2007*, 2007.

[5] F. Franchetti, Y. Voronenko, and M. Püschel. A rewriting system for the vectorization of signal transforms. *High Performance Computing for Computational Science - VECPAR 2006*, 4395/2007:363–377, 2007.

[6] J. Glossner and D. Iancu. The Sandbridge SB3011 SDR platform. In *Proceedings of the Symposium on Trends in Communications (SympoTIC06)*, Bratislava, Slovakia, 2006.

[7] J. Granata, M. Conner, and R. Tolimieri. Recursive Fast Algorithms and the Role of the Tensor Product. *IEEE Trans. Signal Processing*, 40(12):2921–2930, Dez. 1992.

[8] Y. Lin, H. Lee, M. Woh, Y. Harel, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner. SODA: A Low-power Architecture For Software Radio. In *Proc. 33rd Intl. Symposium on Computer Architecture (ISCA)*, 2006.

[9] Y. Lin, H. Lee, M. Woh, Y. Harel, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner. SODA: A High-performance DSP Architecture for Software-Defined Radio. *IEEE Micro*, 27(1):114–123, Jan/Feb 2007.

[10] A. Nilsson and D. Liu. Multi-standard support in SIMT programmable baseband processors. In *Proc. of the Swedish System-on-Chip Conference (SSoCC)*, 2006.

[11] A. Nilsson and D. Liu. Area Efficient Fully Programmable Baseband Processors. In *Embedded Computer Systems: Architectures, Modeling, and Simulation, 7th International Workshop, SAMOS 2007, Samos, Greece, July 16–19, 2007. Proceedings*, 2007.

[12] A. Nilsson, E. Tell, and D. Liu. A programmable SIMD-based multi-standard rake receiver architecture. In *European Signal Processing Conference, EUSIPCO*, 2005.

[13] Technical Specification Group Radio Access Network. TS 36.212 Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding (Release 8). Technical Report V8.1.0, 3rd Generation Partnership Project, Nov. 2007.

[14] C. Temperton. Self-sorting mixed-radix fast Fourier transforms. *Journal of Computational Physics*, 52(1):1–23, Oct 1983.

[15] K. van Berkel, F. Heinle, P. P. E. Meuwissen, K. Moerman, and M. Weiss. Vector processing as an enabler for software-defined radio in handheld devices. *EURASIP Journal on Applied Signal Processing*, 16:2613–2625, 2005.

[16] P. Westermann, G. Beier, H. Ait-Harma, and L. Schwoerer. Developing FFTs for SC-FDMA on the Embedded Vector Processor. In *Proceedings of the 13th International OFDM-Workshop (InOWo'08)*, 2008.