Castro-Palazuelos et al. EURASIP Journal on Advances in Signal Processing (2015) 2015:100
DOI 10.1186/s13634-015-0285-z

EURASIP Journal on Advances in Signal Processing
a SpringerOpen Journal

RESEARCH Open Access

Unified commutation-pruning technique for efficient computation of composite DFTs

David E. Castro-Palazuelos[1,2*], Modesto Gpe. Medina-Melendrez[1], Deni L. Torres-Roman[2] and Yuriy V. Shkvarko[2]

Abstract

An efficient computation of a composite length discrete Fourier transform (DFT), as well as a fast Fourier transform (FFT) of both time and space data sequences in uncertain (non-sparse or sparse) computational scenarios, requires specific processing algorithms. Traditional algorithms typically employ some pruning methods without any commutations, which prevents them from attaining the potential computational efficiency. In this paper, we propose an alternative unified approach with automatic commutations between three computational modalities aimed at efficient computations of the pruned DFTs adapted for variable composite lengths of the non-sparse input-output data. The first modality is an implementation of the direct computation of a composite length DFT, the second one employs the second-order recursive filtering method, and the third one performs the new pruned decomposed transform. The pruned decomposed transform algorithm performs the decimation in time or space (DIT) data acquisition domain and, then, decimation in frequency (DIF). The unified combination of these three algorithms is addressed as the DFT_{COMM} technique. Based on the treatment of the combinational-type hypotheses testing optimization problem of preferable allocations between all feasible commuting-pruning modalities, we have found the global optimal solution to the pruning problem that always requires a fewer or, at most, the same number of arithmetic operations than other feasible modalities. The DFT_{COMM} method outperforms the existing competing pruning techniques in the sense of attainable savings in the number of required arithmetic operations. It requires fewer or at most the same number of arithmetic operations for its execution than any other of the competing pruning methods reported in the literature. Finally, we provide the comparison of the DFT_{COMM} with the recently developed sparse fast Fourier transform (SFFT) algorithmic family. We feature that, in the sensing scenarios with sparse/non-sparse data Fourier spectrum, the DFT_{COMM} technique manifests robustness against such model uncertainties in the sense of insensitivity for sparsity/non-sparsity restrictions and the variability of the operating parameters.

Keywords: Composite length discrete Fourier transform, Decimation, Decomposition, Fast Fourier transform, Pruning

1 Introduction

1.1 Motivation

Many signal processing applications require computation of the so-called pruned discrete Fourier transform (DFT), i.e., an efficient alternative to compute the required DFT when the input sequence and/or the required output sequences are smaller than the length of the full DFT (a full DFT means that all the output components are to be computed, and all the input elements are used to compute the transform); in the literature those are referred to as pruned fast Fourier transforms (FFTs) or pruned DFTs [1]. Common practical examples relate to, e.g., the least mean squared (LMS) optimal DFT-based pruned signal filtering [2], and the complexity-reduced computational implementation of the orthogonal frequency division multiplexing systems [3]. Another practical example relates to efficient implementation of the matched spatial filtering (MSF) algorithm for performing the range and azimuth data compression in unfocused of fractionally focused

* Correspondence: dcastro@gdl.cinvestav.mx
[1]Department of Electrical-Electronic Engineering, Culiacan Technological Institute, Ave. Juan de Dios Batiz S/N, Col. Guadalupe, C.P. 80220 Culiacan, Sinaloa, Mexico
[2]Telecommunications Group, CINVESTAV-IPN Campus Guadalajara, Ave. del Bosque 1145, Col. el Bajio, C.P. 45019 Zapopan, Jalisco, Mexico

synthetic aperture radar (SAR) system that both employ the pruned DFT-based MSF processing of the trajectory data signals performed in a factorized fashion in the so-called slow time and fast time data acquisition scales [4–6]. Other examples relate to DFT-based analysis of remote sensing (RS) data acquired with a variety of sensor systems, ranging from seismology [7] to multispectral radiometry [8]. Other authors as Zhu et al. in [9] proposed an algorithm for performing SAR polar format re-gridding interpolation suited for the logic-in-memory paradigm (hardware/architecture solution) and to provide the necessary design automation tool chain to implement their proposed algorithm (e.g., FFTs for image formation) in advanced silicon technology. It is important to note that a majority of real-world RS data acquisition and processing problems can be qualified as sensing in harsh environments [4–8, 10, 11] in the sense of intrinsic problem model uncertainties peculiar for such RS modalities. In a context of pruned DFTs, realistic harsh sensing scenarios are characterized by the uncertainties attributed to zero-padded input data acquisition modes with variable composite length windowing of the input and/or output Fourier transform sequences, in general cases, with non-sparse Fourier spectra [10–12]. Those specifics motivate the development of efficient pruned DFT/FFT techniques particularly adapted for computational implementation with uncertain data acquired in harsh sensing scenarios.

1.2 Related work

Traditional DFT algorithms adapted for such uncertain scenarios typically employ some pruning methods without any commutations, which prevent them from attaining the potential computational efficiency. Most of the proposals reported in the literature are based on construction of pruning modalities of specific FFT-related algorithms. Some of them prune the input of a specific FFT algorithm, others prune the output, and just a few can prune the input and output (input-output) at the same time. Markel in [1], and Skinner in [13], proposed the input pruning methods based on a radix-2 FFTs, while Yuan et al., in [14], proposed an input pruning of a split-radix FFT. The approaches of Bouguezel et al. [15] and Fan et al. [16] are applicable for output pruning a radix-2 FFTs, while the Xu's et al. [3] proposal suggests pruning the output of a split-radix FFT. In addition, Sreenivas et al., in [17], Roche, in [18], and Wang et al., in [19], developed the methods for pruning the input-output at the same time. The first one is based on a radix-2 FFT, the second one employs the split-radix FFT, and the third one performs the mixed-radix FFT, respectively. A majority of those methods are applicable only for computing DFTs with the length of a power of two that drastically restricts their applicability to general uncertain sensing scenarios.

On the other hand, a family of novel so-called sparse FFT (SFFT) algorithms adapted to computing the FFTs, when only a few Fourier spectrum coefficients of the input signal are different from zero (few largest coefficients of the Fourier transform spectrum), has been developed recently [20, 21]. The celebrated SFFT-related algorithms, so-called SFFTv1 and SFFTv2, were reported by Hassanieh et al., in [20]. Later, in [21], the improved SFFT-related versions, addressed as SFFTv3 and SFFTv4, were reported. Another algorithm that considers the Fourier spectrum sparsity restrictions is the so-called FADFT-2 reported and implemented in the AAFFT library [22]. However, the SFFT-related algorithms significantly outperform the AAFFT as it was corroborated in [20].

It is worthwhile to mention that the SFFT-related techniques are applicable *only* for the sparse sensing scenarios; e.g., referring to [20, 21], the authors exemplified the sparsity level by imposing the restriction that up to 89 % of the Fourier coefficients are zeroes or negligible, thus can be discarded. Such a restriction could be valid in a variety of data compressing applications, e.g., compression and recovery of video data not degraded by noise and/or imaging system instrumental function [20]. Nevertheless, the restriction on such sparsity is not valid for many real-world operational scenarios, e.g., processing of the RS data acquired in harsh sensing environments [4–8, 10–12]. For example, in SAR imaging of non-homogeneous scenes, e.g., urban areas, non-uniformly textured zones, etc., a majority of the Fourier transform coefficients should be considered for feature-enhanced MSF-based imaging [5, 6]; thus, an 89 % of sparsity level restriction is never a feasible model assumption.

In this paper, we are interested in developing the pruned DFT (DFTs of highly composite length) algorithms applicable for near-real-time signal processing and analysis in uncertain sensing scenarios (i.e., with non-guaranteed sparsity of the data Fourier spectra); that is why the family of the SFFT-related techniques is beyond our detailed study here. Nevertheless, for the purpose of generality, in Section 4, we perform comparative analysis of our developed methods with the SFFT under the same conditions and constraints for different combinations of the specified processing/operational parameters.

In the related literature, in which the feasible non-sparse scenarios are considered, two competing approaches for pruning the composite (no prime) length DFTs were addressed. Sorensen et al., in [23], proposed two methods to prune composite length DFTs, first one to prune the input and another one to prune the output. Next, the methodology of Medina-Melendrez et al., in [24], merges the methods developed originally in [23] to

obtain a composite structure that is capable to prune the input and/or output of a general decomposed transform at the same time. It was demonstrated, in [24], that such a computational structure could be as efficient as the one based on specific FFT algorithms [15–17]. In [24], a new methodology for decomposition over a composite length DFT has been proposed as a modification of the Sorensen's approach [23]. Furthermore, the [24] suggests, first, to perform decimation in frequency (DIF) and, second, a decimation in time (DIT). For processing of spatial data, the corresponding decimation in the space domain should be performed similarly to the DIT operation for time data processing. To avoid misunderstandings, in the rest of the paper, we will use the same abbreviation (DIT) for both processing models and consider the time data processing as a principal model. Nevertheless, all developments are directly transferable for the space data processing scenario.

Hence, the three basic stages to compute the composite length DFTs of non-sparse data encompass the input, the intermediate, and the output stages. The decomposed transform is then pruned by eliminating, from the input and output stages, additions and multiplications by zero, multiplications by one, and all other computations not needed to obtain the required Fourier transform coefficients. In [24], such the multistage decomposed and pruned transform is referred to as $\mathrm{FFT}_{\mathrm{DIF-DIT-TD}}$ (here, that method is referred as $\mathrm{DFT}_{\mathrm{DIF-DIT-Pr}}$). Nevertheless, both methods addressed in [23, 24] do not achieve the lowest attainable number of the required arithmetic operations. A possible alternative for computing few Fourier coefficients from few input elements (all non-zero, thus non-sparse) can be addressed based on the application of the second-order Goertzel algorithm [23] modified to accept the input elements in a reverse order.

1.3 Novel contributions

The main contribution of this paper consists in the development of a new alternative method for efficient computing of a composite length DFT, when the input sequence and/or the required output sequence are smaller than the length of the full transform. Our proposal guarantees the same or smaller number of arithmetic operations in comparison with the competing methods in the literature. Moreover, it manifests robustness against sparsity/non-sparsity restrictions and the variability of the operating parameters as detailed in Sections 3 and 4.

The innovative idea is to automatically commute among three modalities to implement the DFT: the direct method, the recursive method, and the pruned decomposed transform. Thus, our new proposed composite approach unifies the decomposition of the DFT with its pruning. First, we develop an alternative technique to compute the pruned decomposed transform, in which the DIT is performed at the first stage followed by the DIF. We address this method as $\mathrm{DFT}_{\mathrm{DIT-DIF-Pr}}$. An analysis of the two alternatives ($\mathrm{DFT}_{\mathrm{DIT-DIF-Pr}}$ and $\mathrm{DFT}_{\mathrm{DIF-DIT-Pr}}$) verifies that the $\mathrm{DFT}_{\mathrm{DIT-DIF-Pr}}$ requires a smaller or as maximum equal number of arithmetic operations compared with the $\mathrm{DFT}_{\mathrm{DIF-DIT-Pr}}$, so the use of the $\mathrm{DFT}_{\mathrm{DIT-DIF-Pr}}$ is strongly recommended when the decomposed and pruned transforms are required. Next, we demonstrate that our proposal requires a lower number of arithmetic operations than any of the pruning-based competing methods [3, 14, 23, 24]. Further, we demonstrate that both decomposed transforms ($\mathrm{DFT}_{\mathrm{DIF-DIT-Pr}}$ and $\mathrm{DFT}_{\mathrm{DIT-DIF-Pr}}$) can be obtained from a general decomposition methodology. Also, it manifests the robustness in sparse and non-sparse sensing scenarios (i.e., operability for an arbitrary number of consecutive input elements (L_i), the number of consecutive outputs that should be computed (L_o), and the length of the full transform (N)) in contrast to the recently developed most prominent SFFT family-related methods [20, 21] operable in sparse scenarios only.

It is noteworthy to mention that in the majority of practical computational scenarios, significant savings in the number of arithmetic operations with the proposed technique are achieved, e.g., in Section 4.1, the $\mathrm{DFT}_{\mathrm{COMM}}$ technique compared with split-radix FFT (SRFFT) algorithm produces savings of 42 to 92 %.

The rest of the paper is organized as follows: in Section 2, the general decomposition transform methodology is described and explained. An analysis of all feasible transform decomposition methods is presented next in Section 3 followed by the combinational hypotheses testing optimization-based selection of the best decomposition transform permutation modality that yields the unified commutation-pruning $\mathrm{DFT}_{\mathrm{COMM}}$ technique. In Section 4, comparisons among the developed unified commutation-pruning technique and other competing algorithms in the sense of savings in the number of required arithmetic operations are presented and featured. Also, the proposed $\mathrm{DFT}_{\mathrm{COMM}}$ method is compared in detail with the most prominent competing SFFT-related algorithms in the context of computing the DFTs in both sparse and non-sparse (harsh) sensing scenarios for different values of the operational parameters (L_i, L_o, and N). Concluding remarks in Section 5 summarize the study. The Appendix provides a pseudo-code for implementing the proposed method.

2 DFT transform decomposition

The definition of the DFT of a sequence of length N (DFT_N) is given by

$$X(k)=\sum_{n=0}^{N-1}x(n)W_N^{nk}\ \text{ for }\ k=0,1,2,...,N-1 \tag{1}$$

where $W_N^{nk}=e^{-j2\pi nk/N}$ is the kernel of the transform. Let us define L_i as the number of consecutive input elements different from zero and L_o as the number of consecutive outputs that should be computed. If N is a composite number formed by multiplications of many integer factors, the DFT_N can be decomposed into smaller DFTs. In particular, the DFT_N can be decomposed into three stages of DFTs (an input stage, an intermediate stage, and an output stage) in order to avoid the arithmetic operations involving zeros, multiplications by one, and the operations not required to compute the final outputs. Here beneath, we briefly describe such feasible decompositions. Assuming that there are two integer factors, D_{ip} and D_{op}, of N such that $N/D_{ip}D_{op}\equiv P$ is an integer, the indexes n and k can be re-expressed as

$$n=n_1+D_{op}n_2+\left(\frac{N}{D_{ip}}\right)n_3$$
$$\text{for}\ \begin{cases} n_1=0,1,2,...,D_{op}-1\\ n_2=0,1,2,...,N/D_{ip}D_{\mathrm{op}}-1\\ n_3=0,1,2,...,D_{ip}-1\end{cases} \tag{2}$$

$$k=k_1+D_{ip}k_2+\left(\frac{N}{D_{op}}\right)k_3$$
$$\text{for}\ \begin{cases} k_1=0,1,2,...,D_{ip}-1\\ k_2=0,1,2,...,N/D_{ip}D_{op}-1\\ k_3=0,1,2,...,D_{op}-1.\end{cases} \tag{3}$$

Substituting n and k in (1) by (2), (3), the original DFT_N is decomposed into

$$\begin{aligned}&X\left(k_1+D_{ip}k_2+\frac{N}{D_{op}}k_3\right)\\&=\sum_{n_1=0}^{D_{op}-1}\sum_{n_2=0}^{P-1}\sum_{n_3=0}^{D_{ip}-1}x\left(n_1+D_{op}n_2+\frac{N}{D_{ip}}n_3\right)\\&\times W_N^{\left(n_1+D_{op}n_2+\left(N/D_{ip}\right)n_3\right)\left(k_1+D_{ip}k_2+\left(N/D_{op}\right)k_3\right)}.\end{aligned} \tag{4}$$

Here, it is assumed that D_{ip} and D_{op} are chosen in such a way that $N/D_{ip}\geq L_i$ and $N/D_{op}\approx L_o$. Thus, index n_3 is always equal to zero; k_3 is near 0, hence (4) can be next rewritten as follows

$$\begin{aligned}&X\left(k_1+D_{ip}k_2+\frac{N}{D_{op}}k_3\right)\\&=\sum_{n_1=0}^{D_{op}-1}\sum_{n_2=0}^{P-1}x\left(n_1+D_{op}n_2\right)\\&\quad W_N^{\left(n_1+D_{op}n_2\right)\left(k_1+D_{ip}k_2+\left(N/D_{op}\right)k_3\right)}\\&=\sum_{n_1=0}^{D_{op}-1}\sum_{n_2=0}^{P-1}x\left(n_1+D_{op}n_2\right)\\&\quad W_N^{\left(n_1k_1+D_{op}n_2k_1+D_{ip}n_1k_2+D_{ip}D_{op}n_2k_2+n_1k_3N/D_{op}+n_2k_3N\right)}.\end{aligned} \tag{5}$$

The computation of (5) is more efficient than the direct computation of the DFT_N since the complex arithmetic operations dependent on n_3 have been pruned. The complex exponential in (5) can next be grouped in different ways, resulting in different structures for the pruned decomposed transform. The methodology of [24] suggests expressing the pruned decomposed transform as

$$\begin{aligned}&X\left(k_1+D_{ip}k_2+\frac{N}{D_{op}}k_3\right)\\&=\sum_{n_1=0}^{D_{op}-1}\left\{\sum_{n_2=0}^{P-1}\left[\ W_N^{\left(\left(n_1+D_{op}n_2\right)k_1\right)}x\left(n_1+D_{op}n_2\right)\right.\right.\\&\left.\left.\times\ W_{N/D_{ip}D_{op}}^{n_2k_2}\right]\right\}W_N^{\left(n_1\left(D_{ip}k_2+k_3N/D_{op}\right)\right)}.\end{aligned} \tag{6}$$

The pruned decomposed transform of (6) can be interpreted as follows: first, apply DIF to the DFT_N with D_{ip} as a decomposition factor, then, DIT to the resulting DFTs with D_{op} as a decomposition factor and, finally, perform the pruning. In [24], the pruned decomposed transform of (6) was addressed as an $\mathrm{FFT}_{\mathrm{DIF-DIT-TD}}$ modality, that in our notations, we refer to as $\mathrm{DFT}_{\mathrm{DIF-DIT-Pr}}$. A computational diagram of such technique (6) is presented in Fig. 1.

An alternative grouping of the complex exponentials in (5) yields

$$\begin{aligned}&X\left(k_1+D_{ip}k_2+\frac{N}{D_{op}}k_3\right)\\&=\sum_{n_1=0}^{D_{op}-1}\left\{\sum_{n_2=0}^{P-1}\left[W_N^{\left(D_{op}n_2k_1\right)}x\left(n_1+D_{op}n_2\right)W_P^{n_2k_2}\right]\right\}\\&\times W_N^{\left(n_1\left(k_1+D_{ip}k_2+k_3N/D_{op}\right)\right)}\\&=\sum_{n_1=0}^{D_{op}-1}\left\{\sum_{n_2=0}^{P-1}\left[y(n_1,n_2,k_1)W_P^{n_2k_2}\right]\right\}\\&\times W_N^{\left(n_1\left(k_1+D_{ip}k_2+k_3N/D_{op}\right)\right)}.\end{aligned} \tag{7}$$

The computing of the pruned decomposed transform (7) requires, first, application of DIT to the DFT_N with

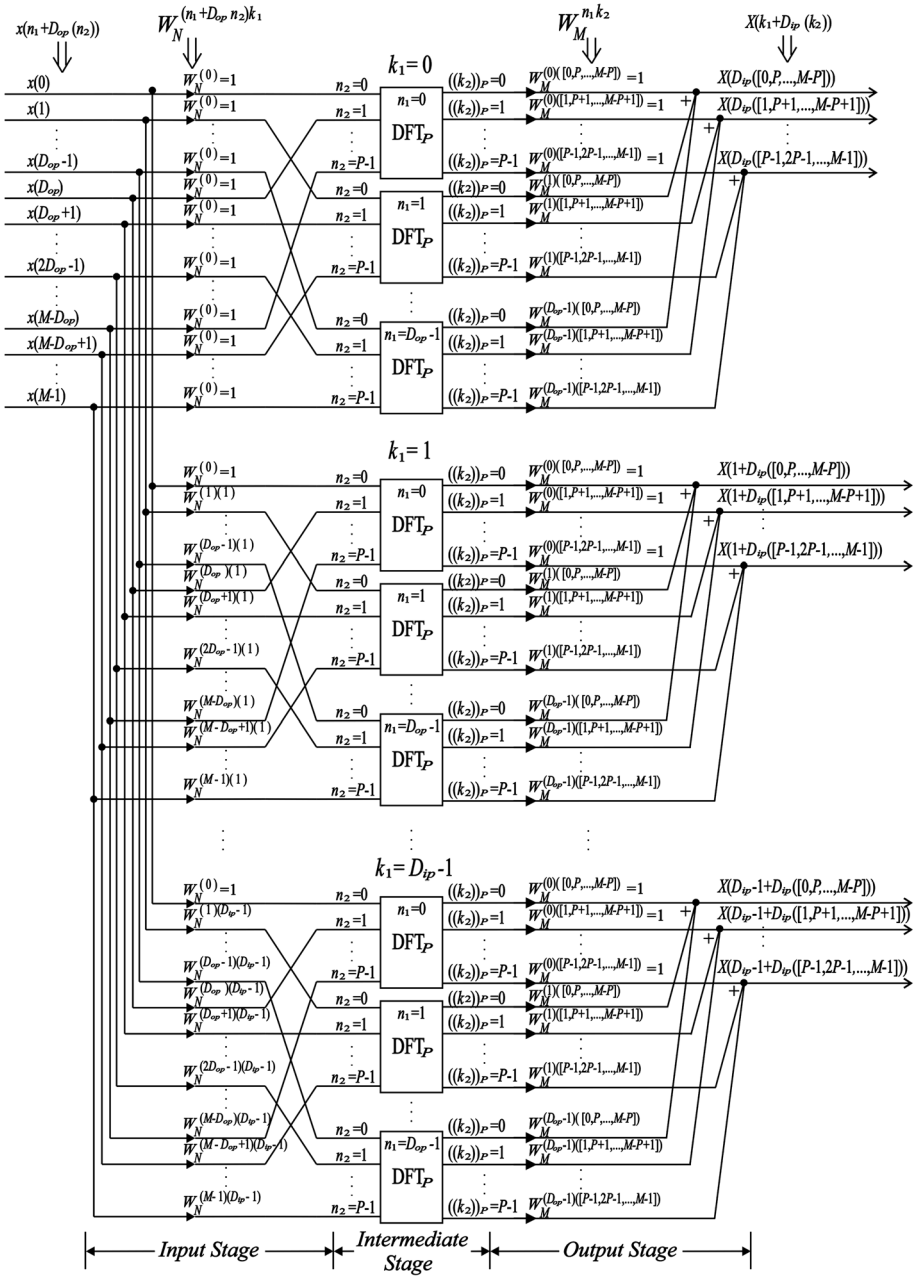

Fig. 1 General computational diagram of the $\mathrm{DFT}_{\mathrm{DIF-DIT-Pr}}$ (referred as $\mathrm{FFT}_{\mathrm{DIF-DIT-TD}}$ in [24, Fig. 1])

D_{op} as a decomposition factor and, then, application of DIF to the resulting DFTs with D_{ip} as a decomposition factor.

Hence, we refer to the pruned decomposed transform of (7) as a $\mathrm{DFT}_{\mathrm{DIT-DIF-Pr}}$ modality. A computational diagram of such the technique (7) is presented in Fig. 2.

The $\mathrm{DFT}_{\mathrm{DIT-DIF-Pr}}$ involves three processing stages: an input stage (computation of $y(n_1, n_2, k_1)$), an intermediate stage (computation of $D_{ip}D_{op}$ DFTs of length P), and an output stage (computation of the complex multiplications and additions dependent on index n_1).

3 Proposed method

Our method employs three different alternatives to compute the DFT_N: a direct method, a recursive method, and/or a pruned decomposed transform. Admissible permutations/allocations of all feasible decomposition-pruning modalities compose all possible hypotheses regarding the feasible alternative schemes for computing the composite DFTs.

All feasible commuting-pruning implementation structures are listed in Table 1. Those could be addressed as possible search "hypotheses" to be tested. Thus, the

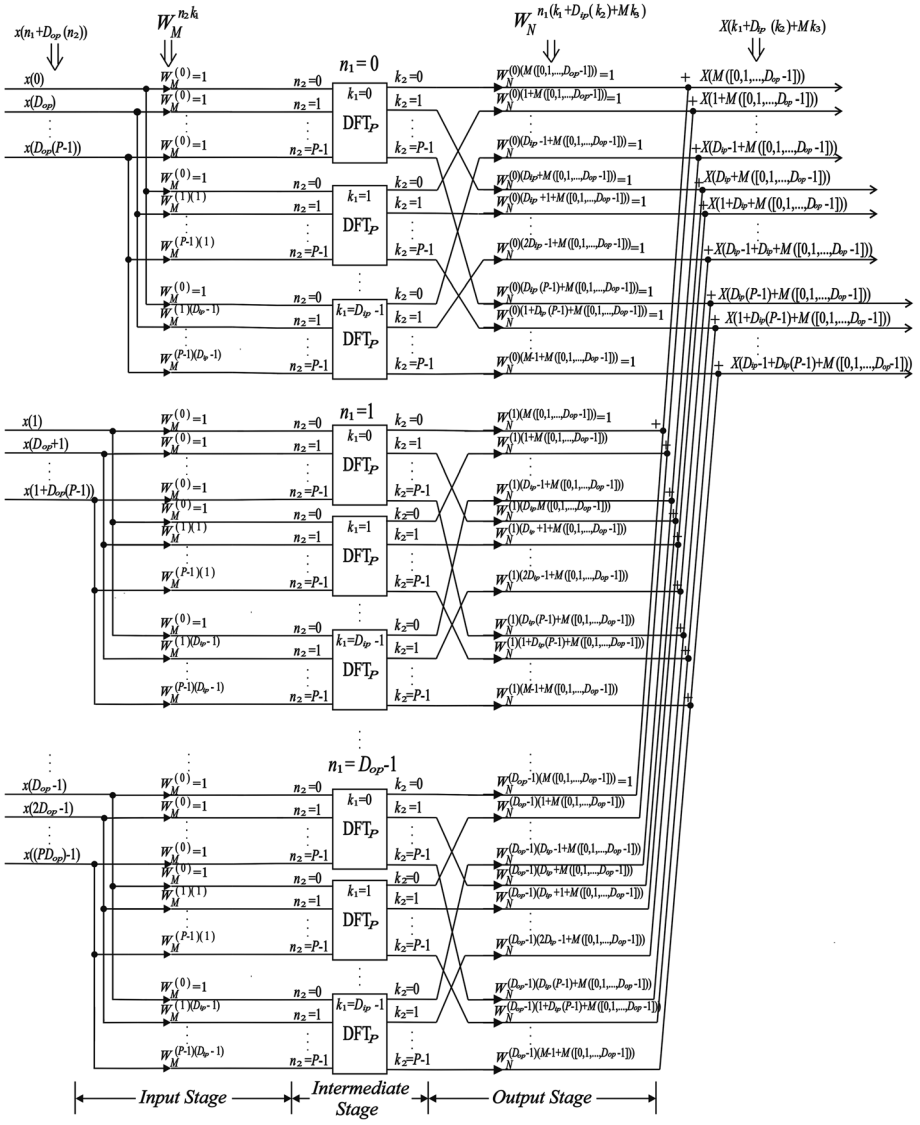

Fig. 2 General computational diagram of the $DFT_{DIT\text{-}DIF\text{-}Pr}$

problem of selection of an optimal computing-pruning implementation structure can be recast as a hypotheses testing task. All feasible hypotheses relate to formal implementation structures specified in Table 1. Four of them prescribe cascade computational implementation involving cascade combinations of structures (hypotheses H_4,..., H_7), while four others (hypotheses H_9,..., H_{12}) prescribe combinational unions of the previous hypotheses. It is important to remark that (1), (6), and (7) are the mathematical definitions of H_8, H_4, and H_5, respectively. Hence, the decision-making process that is a selection from those feasible operational prescriptions cannot be formalized as an optimization strategy for minimization of some cost function subject to relevant restrictions/constraints specified in a closed analytical form. Thus, due to the composite combinations (hypotheses over hypotheses with cascade interlaces, as in the cases of hypotheses H_9,..., H_{12}), the proper selection of the preferable implementation structure cannot be cast as an analytically tractable closed-form optimization problem. Hence, it should be treated as a test of combinations of hypotheses (hypotheses over hypotheses, as in the case of H_9,..., H_{12}), sometimes referred to as a combinational (or combinatorial-type) hypothesis testing problem [23, 24]. The global optimal solution to such a kind of problems presumes test of all feasible hypotheses in the list, making the decision in favor of the best one (in the prescribed quality measure), and rejection of all other competing hypotheses [23, 24]. In our particular case, only 12 hypotheses are admissible/feasible; thus the (global) optimal selection of the best possible implementation structure can be found simply via employing the so-called brute

Table 1 Complete list of hypotheses $\{H_h\}_{h=1}^{12}$ regarding feasible commuting-pruning implementation structures

Hypotheses	Specifications
$H_1 : y = Ax$	x = input
$H_2 : y = Bx$	y = output
$H_3 : y = Fx$	A = DIF
$H_4 : y = ABx$	B = DIT
$H_5 : y = BAx$	F = 2BF filtering
$H_6 : y = ABFx$	D = DFT_N
$H_7 : y = BAFx$	
$H_8 : y = Dx$	
$H_9 : H_4 \cup H_6$	
$H_{10} : H_5 \cup H_7$	
$H_{11} : H_9 \cup H_3 \cup H_8$	
$H_{12} : H_{10} \cup H_3 \cup H_8$	

force search over complete hypotheses list specified in Table 1.

Sorensen et al. [23] sketched how to prune the input and output of DFTs using independent allocations listed in Table 1 as H_1, H_2, and H_3 and featured in Fig. 3a. However, the authors of [23] concluded that their pruning method is less efficient than other pruning methods in the cases when both the number of input and output elements are bounded. They recommended turning to the method proposed by Sreenivas et al., in [17], i.e., to prune the input and output of a power of two length FFTs. Furthermore, an efficient input-output pruning method for a power of two length FFTs was proposed by Roche in [18].

Later, a more efficient input-output pruning method for composite length DFTs was developed in [24]. Such commuting between $H_4 \cup H_6$ leads to hypothesis H_9 as featured in Fig. 3a. In [24], such a technique was constructed as a modification of the transform decomposition proposed originally by Sorensen et al., in [23], but with extra capability to perform the input-output pruning at the same time. Additionally, the computation of each final output employs a commutation between a direct method and the 2BF filtering algorithm, i.e., the 2BF-filtering algorithm is an efficient method for computing a subset of final outputs from their decomposition transform [23, 24].

In our study, two additional feasible hypotheses are devised to perform unified commutation-pruning techniques for efficient computations of composite length DFTs (hypotheses H_{11} and H_{12}) as reported in Fig. 3b. Therefore, our proposal relates to an adaptive commuting between feasible implementation structures specified by the union of hypotheses $H_{10} \cup H_3 \cup H_8$ that is included in Table 1 as an alternative composite hypothesis H_{12}. A comparison of computational complexities related to implementation of the competing computational structures formalized by hypotheses H_9 and H_{10} (in the number of required arithmetical operations) is reported in Table 2. Also, the relevant comparisons between two other feasible structures specified by hypotheses H_{11} and H_{12} (referred here as $DFT_{COMM-DIF-DIT-Pr}$ and $DFT_{COMM-DIT-DIF-Pr}$, respectively), are reported in Tables 2 and 3 and Figs. 5a–f (in the sense of the number of required arithmetic operations).

The selection of proper permutation/allocation structure directly relates to the considered above problem of selection of an optimal commutation-pruning implementation structure casted and treated as a combinational hypotheses testing task. All feasible hypotheses $\{H_h\}_{h=1}^{12}$ relate to formal implementation structures specified in Table 1. Now, we are ready to find the best permutation/allocation structure in the sense of the imposed quality measure (in our case in the sense of the lowest possible number of required arithmetical operations).

3.1 Analysis of the hypotheses

Let us analyze, first, the pruned decomposed transform and deduce whether the direct or recursive method would be preferable. The total number of arithmetic operations ($OPER_{tot}$) required by the $DFT_{DIF-DIT-Pr}$ and the $DFT_{DIT-DIF-Pr}$ depends on the number of operations needed to be performed to implement the input stage ($OPER_{input}$), the output stage ($OPER_{output}$), and the intermediate stage ($D_{ip}D_{op}OPER_{DFTP}$), correspondingly. Thus, one could express $OPER_{tot}$ of both pruned decomposed transforms as

$$OPER_{tot} = OPER_{input} + D_{ip}D_{op}OPER_{DFTP} + OPER_{output}. \quad (8)$$

According to (8), $OPER_{tot}$ depends on L_i, L_o, N, D_{ip}, D_{op}, and the algorithm employed to implement the D_{ip}-D_{op} DFT_P blocks ($OPER_{DFTP}$).

At the input and output stages, there are multiplications by one, so those multiplications are avoided at all in our approach. Also, the multiplications by one at the input stage are also avoided depending on whether $DFT_{DIF-DIT-Pr}$ or $DFT_{DIT-DIF-Pr}$ was executed in the particular employed pruned decomposed transform modality.

If the $DFT_{DIF-DIT-Pr}$ modality is employed (see the general diagram in Fig. 1), then:

- At the input stage, the multiplications by one are excluded when $n_1 = n_2 = 0$ and $k_1 = 0$.
- Furthermore, the multiplications by one at the output stage are also avoided when $n_1 = 0$ or $k_2 = 0$.

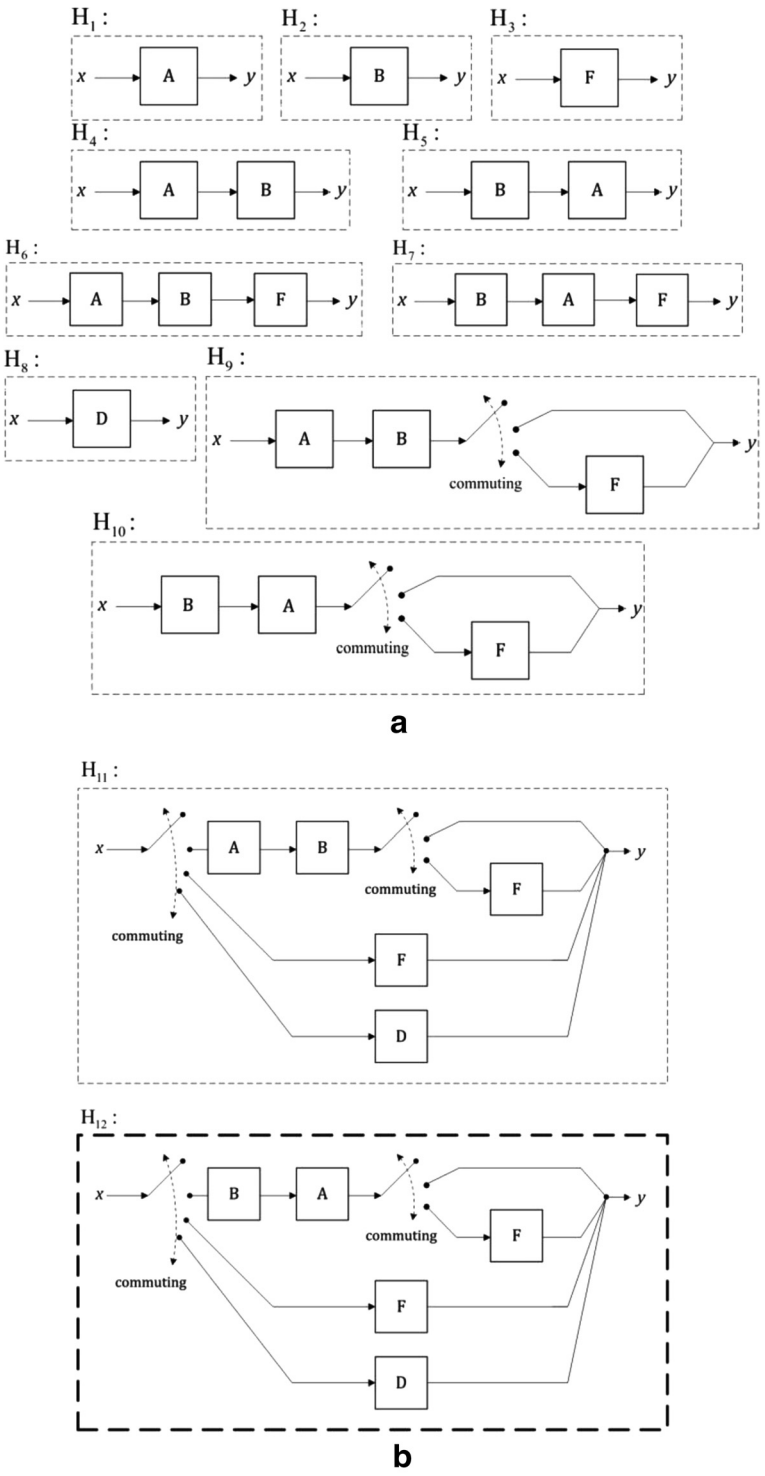

Fig. 3 Possible feasible alternative schemes for efficient computation of composite DFTs when the input and/or output sequences are smaller than the length of the full transform. **a** State of the art. **b** New feasible implementation structures. Specifications of the blocks labeled as A, B, F, and D are listed in Table 1

Table 2 Total number of arithmetic operations required to compute the input and output stages of $DFT_{DIF-DIT-Pr}$ and $DFT_{DIT-DIF-Pr}$

Pruned decomposed transform	Corresponding term in (8)	Number of required real arithmetic operations	Limiting constraints
$DFT_{DIF-DIT-Pr}$	$OPER_{input}$	$6\,(L_i-1)\,(D_{ip}-1)$	–
	$OPER_{output}$	$2L_o\,(D_{op}-1)$	$(L_o \le D_{ip})$
		$2L_o\,(D_{op}-1)+6\,(L_o-D_{ip})\,(D_{op}-1)$	$(L_o > D_{ip})$ & $(D_{op} < 4)$
		$(L_o-D_{ip})\,(2D_{op}+2)+2D_{ip}\,(D_{op}-1)+(L_o-D_{ip})\,(4D_{op}-2)$[a]	$(L_o > D_{ip})$ & $(D_{op} \ge 4)$
$DFT_{DIT-DIF-Pr}$	$OPER_{input}$	0	$(L_i \le D_{op})$
		$6\,(L_i-D_{op})\,(D_{ip}-1)$	$(L_i > D_{op})$
	$OPER_{output}$	$6\,(L_o-1)\,(D_{op}-1)+2L_o\,(D_{op}-1)$	$(D_{op} < 4)$
		$(L_o-1)\,(2D_{op}+2)+2\,(D_{op}-1)+(L_o-1)\,(4D_{op}-2)$[a]	$(D_{op} \ge 4)$

[a]Number of required operations when the 2BF filtering method is employed [23, 24]

Therefore, the $DFT_{DIF-DIT-Pr}$ modality always requires fewer complex multiplications to compute the output stage than the $DFT_{DIT-DIF-Pr}$ modality (this is reported in Tables 2 and 3).

On the other hand, if the $DFT_{DIT-DIF-Pr}$ modality is used (see Fig. 2), then:

- At the input stage, the multiplications by one are excluded when $n_2 = 0$ or $k_1 = 0$.
- Also, at the output stage, the multiplications by one are avoided when $k_1 = k_2 = 0$ or $n_1 = 0$.

Therefore, the $DFT_{DIT-DIF-Pr}$ modality always requires fewer complex multiplications at the input stage than the $DFT_{DIF-DIT-Pr}$ modality (as it is corroborated in the analysis reported in Tables 2 and 3).

The output stage of both pruned decomposed transform modalities can be computed by the direct addition of complex multiplications or a kind of recursive algorithm as those proposed in [23] (referred to as the 2BF filtering method), which reduces the number of required multiplications by about half. The number of arithmetic multiplications required by the output stage of the $DFT_{DIF-DIT-Pr}$ algorithm is equal to $4\,(L_o - D_{ip})\,(D_{op} - 1)$ when $(L_o > D_{ip})$ and $(D_{op} < 4)$. Next, the number of arithmetic multiplications is equal to $(L_o - D_{ip})\,(2D_{op} + 2)$ when $(L_o > D_{ip})$ and $(D_{op} \ge 4)$. Thus, the 2BF filtering algorithm can be effectively used to compute the output stage.

On the other hand, the number of arithmetic multiplications required to compute the output stage of the $DFT_{DIT-DIF-Pr}$ algorithm is equal to $4(L_o - 1)\,(D_{op} - 1)$ when $(D_{op} < 4)$; and the number of arithmetic multiplications is equal to $(L_o - 1)\,(2D_{op} + 2)$ when $(D_{op} \ge 4)$. Thus, the 2BF filtering algorithm can also be effectively employed to compute the output stage.

In [23], it was proven that the 2BF filtering method is more efficient than the direct addition of complex multiplications when the number of input elements is larger than 4 (when the number of input elements is equal to 4, both methods manifest the same operational complexity performances). The output stages of both pruned decomposed transforms have the same structures, so same sort of commutations is required to efficiently compute the output stage of the $DFT_{DIT-DIF-Pr}$. The expressions for $OPER_{input}$ and $OPER_{ouput}$ for the $DFT_{DIF-DIT-Pr}$ and the $DFT_{DIT-DIF-Pr}$ are listed in Table 2, where it is implicitly assumed that each complex multiplication requires six arithmetic operations (four real multiplications and two real additions), and each complex addition requires two arithmetic operations (two real additions).

The performances of the pruned decomposed transforms depend on the decomposition factors, D_{ip} and D_{op}. A simple analysis can be carried out to deduce which decomposition factors are preferable to be used. Our unified commutation-pruning method performs the decomposition of the DFT_N into three stages of smaller dimension DFTs and pruning part of those inputs that are equal to zero and/or part of those outputs that are not needed to compute the final Fourier coefficients.

Thus, the decomposed transform algorithm always selects a pair (D_{ip}, D_{op}) for which the largest DFTs could be successfully pruned, or equivalently, a pair (D_{ip}, D_{op}) for which the intermediate stage results in the smallest dimension DFTs.

Table 3 Total number of arithmetic operations required to compute the input and output stages of $DFT_{COMM-DIF-DIT-Pr}$ and $DFT_{COMM-DIT-DIF-Pr}$ modalities

Method to compute the DFT_N	Number of arithmetic operations	Limiting conditions
Direct method	$6\,(L_o-1)\,(L_i-1)+2L_o\,(L_i-1)$	$((L_i \le D_{op})\|(L_o \le D_{ip}))$ & $(L_i < 4)$
2BF filtering method	$(L_o-1)\,(2L_i+2)+2\,(L_i-1)+(L_o-1)\,(4L_i-2)$	$((L_i \le D_{op})\|(L_o \le D_{ip}))$ & $(L_i \ge 4)$
Pruned decomposed transform	OPERtot of (8) using Table 2	$(L_i > D_{op})$ & $(L_o > D_{ip})$

The DFTs of the intermediate stage have a size of $N/D_{ip}D_{op} \equiv P$, so D_{ip} and D_{op} should be chosen as large as possible. Furthermore, the values for the decomposition factors should satisfy the bound $N/D_{ip} \geq L_i$ (where, N/D_{ip} must be close to but higher than L_i) and $N/D_{op} \approx L_o$, as it was considered in the derivation of (5). Hence, the pair of decomposition factors (D_{ip}, D_{op}) closest to (N/L_i, N/L_o) that satisfy $D_{ip} \leq N/L_i$ are used by the decomposed transform algorithm, according to the proximity evaluated by its Euclidean distance.

Let us now consider the cases when the number of input elements (L_i) or the number of the required Fourier coefficients (L_o) is too small. In these cases, for the both modalities, the general diagrams presented in Figs. 2 and 1 clarify the following features of the $\mathrm{DFT}_{\mathrm{DIT-DIF-Pr}}$ and the $\mathrm{DFT}_{\mathrm{DIF-DIT-Pr}}$ algorithms, respectively.

- If $L_i \leq D_{op}$, at most one input of each DFT_P (i.e., the first one) in the intermediate stage would be applied; therefore, their P outputs would be replicas of that single input.
- For $L_o \leq D_{ip}$, only the first output of each DFT_P (this corresponds to a simple addition of the input elements) is required to compute the final Fourier coefficients.

Thus, inefficient implementations of the DFT_Ps yield the inequality-*type* constraints $Li \leq D_{op}$ or $L_o \leq D_{ip}$. In these cases, our method commutes to efficiently perform the direct computation of the DFT_N or an efficient recursive alternative (via performing the 2BF filtering technique).

Sorensen et al., in [23], proposed a method to compute a subset of the output components of their proposed specific DFT decomposition; this algorithm was referred to as a 2BF filtering method. The 2BF filtering method [23] was derived as a modification of the previously addressed Goertzel algorithm [25]. The 2BF filtering method takes advantages of the periodicity and the shifted cyclic convolution shape between the input sequence and the $W_N^{nk} = e^{-j(2\pi/N)kn}$ factor.

The transfer function $H(z)$ of a system that performs the 2BF filtering method is given by the equation

$$H(z) = \frac{z^{-1}\left(1 - z^{-1}W_N^{-k}\right)}{1 - 2\cos\left(\frac{2\pi k}{N}\right)z^{-1} + z^{-2}} \tag{9}$$

The corresponding algorithmic diagram of the second-order 2BF method is presented in Fig. 4. Thus, (9) is the mathematical definition of H_3.

The poles of the system transfer function (the roots of the polynomial in the denominator of $H(z)$) have to be evaluated L times ($n = 0, 1, 2, \ldots, L-1$), while the zeros of the system transfer function (the roots of the numerator of $H(z)$) only once. Here, L represents the number of consecutive non-zero input elements of the 2BF filter; i.e., in the opposite case, it represents the number of consecutive non-zero output components of the employed pruned decomposed transform modality ($\mathrm{DFT}_{\mathrm{DIF-DIT-Pr}}$ or $\mathrm{DFT}_{\mathrm{DIT-DIF-Pr}}$).

The computation of each pole of (9) requires two arithmetic multiplications (two real multiplications) and two arithmetic additions (two real additions). Furthermore, the computation of the zeros of (9) requires four arithmetic multiplications and four arithmetic additions only.

The Q_1 node in Fig. 4 is initialized with $f(L-1)$; therefore, the computation starts from $n = L-2$. When $n = 0$, the complex addition of the input is only required; then, the zero is computed after such a delay. Such computational organization saves two arithmetic multiplications and six arithmetic additions for finding of each required output component.

The 2BF filtering method employed to compute the output components required by the pruned decomposed transform performed by the $\mathrm{DFT}_{\mathrm{COMM-DIF-DIT-Pr}}$ or the $\mathrm{DFT}_{\mathrm{COMM-DIT-DIF-Pr}}$ algorithm can be featured as the following multistage procedure:

- The structure of the $\mathrm{DFT}_{\mathrm{DIF-DIT-Pr}}$ contains D_{ip} sets of D_{op} DFT_Ps from which the final outputs are computed (see the general diagram in Fig. 1).
- The $\mathrm{DFT}_{\mathrm{COMM-DIF-DIT-Pr}}$ algorithm employs the 2BF filtering method to implement the output stage of $\mathrm{DFT}_{\mathrm{DIF-DIT-Pr}}$ with $L = L_o$, if (($L_i > D_{op}$) & ($L_o > D_{ip}$)) & (($L_o > D_{ip}$)&($D_{op} \geq 4$)) (as featured in Tables 2 and 3). Here, the required arithmetic operations are specified as follows: the number of arithmetic multiplications are equal to $NumArithMult_{2BF} = (L_o - D_{ip})(2D_{op} + 2)$ and the number of arithmetic additions are equal to $NumArithAdd_{2BF} = 2\,D_{ip}(D_{op} - 1) + (L_o - D_{ip})(4D_{op} - 2)$.
- Furthermore, the $\mathrm{DFT}_{\mathrm{COMM-DIF-DIT-Pr}}$ algorithm employs the 2BF filtering method exclusively with

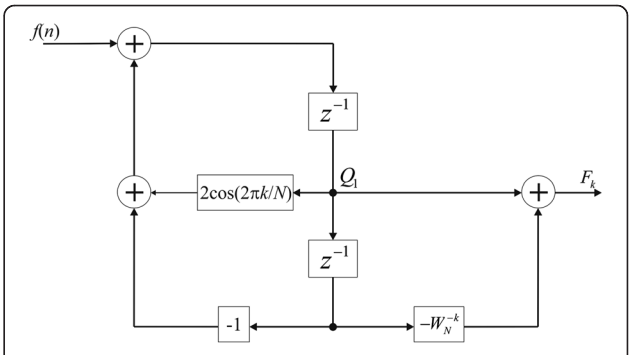

Fig. 4 Algorithmic diagram of the second order 2BF filtering method

$L = L_i$, if ($(L_i \le D_{op})$ | $(L_o \le D_{ip})$) & $(L_i \ge 4)$ (as featured in Table 3) to compute the required Fourier coefficients. Here, the required arithmetic operations are specified as follows: $NumArithMult_{2BF} = (L_o - 1)(2L_i + 2)$ and $NumArithAdd_{2BF} = 2(L_i - 1) + (L_o - 1)(4L_i - 2)$.

In contrast, the $\text{DFT}_{\text{COMM-DIT-DIF-Pr}}$ algorithm differs from the abovementioned in the following features:

- The structure of the $\text{DFT}_{\text{DIT-DIF-Pr}}$ contains D_{op} sets of D_{ip} DFT_Ps from which the final outputs are computed (as featured in Fig. 2).
- The $\text{DFT}_{\text{COMM-DIT-DIF-Pr}}$ algorithm employs the 2BF filtering method to implement the output stage of $\text{DFT}_{\text{DIT-DIF-Pr}}$ with $L = L_o$, if ($(L_i > D_{op})$ & $(L_o > D_{ip})$) & $(D_{op} \ge 4)$, (as featured in Tables 2 and 3). Here, the required arithmetic operations are specified as follows: $NumArithMult_{2BF} = (L_o - 1)(2D_{op} + 2)$, and $NumArithAdd_{2BF} = 2\ (D_{op} - 1) + (L_o - 1)(4D_{op} - 2)$.
- On the other hand, the $\text{DFT}_{\text{COMM-DIT-DIF-Pr}}$ algorithm employs the 2BF filtering method exclusively with $L = L_i$, if ($(L_i \le D_{op})$ | $(L_o \le D_{ip})$) & $(L_i \ge 4)$ (as reported in Table 3) to compute the required Fourier coefficients. Here, the required arithmetic operations are specified as follows: $NumArithMult_{2BF} = (L_o - 1)(2L_i + 2)$ and $NumArithAdd_{2BF} = 2(L_i - 1) + (L_o - 1)(4L_i - 2)$.

The computation of each input and/or output element in both cases detailed above is executed according to the diagram presented in Fig. 4. In closing, we note that the pseudo-code presented in the Appendix (see Fig. 9) contains all scripts needed to compute each Fourier coefficient employing the 2BF filtering method.

Note once again that the 2BF filtering method has to be employed if L_i is larger or equal to 4, in which case, it manifests a higher efficiency than the direct method for computing the DFT_N in (1). The total number of arithmetic operations required by our proposed method is reported in Table 3.

3.2 Selection of the permutation/allocation structure

In Fig. 5, the total number of required arithmetic operations to compute the $\text{DFT}_{\text{DIF–DIT–Pr}}$ from [24] (H_9), $\text{DFT}_{\text{COMM–DIF–DIT–Pr}}$ (H_{11}), and $\text{DFT}_{\text{COMM–DIT–DIF–Pr}}$ (H_{12}) modalities are plotted for different values of L_i and L_o for the test examples with $N = 8192$ and $N = 6561$ (It is assumed that the DFT_Ps are implemented employing the split-radix algorithm from [26] for $N = 8192$ and employing the radix-3 algorithm from [27] for $N = 6561$.) All the competing alternatives corresponding to three feasible arrangements (H_9, H_{11}, and H_{12}) in the considered permutation/allocation structure are featured in Fig. 5. The $\text{DFT}_{\text{DIF–DIT–Pr}}$ or the $\text{DFT}_{\text{DIT–DIF–Pr}}$ could be used to implement the pruned decomposed transform in the $\text{DFT}_{\text{COMM–DIF–DIT–Pr}}$ and $\text{DFT}_{\text{COMM–DIT–DIF–Pr}}$ techniques. Here, the D_{ip} and D_{op} values are the pair specified by the rough selection method (the proximity evaluated by its Euclidean distance is referred as roughDP) and those obtained by an exhaustive search method (the total numbers of operations required to implement the $\text{DFT}_{\text{DIF–DIT–Pr}}$ and the $\text{DFT}_{\text{DIT–DIF–Pr}}$ were evaluated for each possible pair of (D_{ip}, D_{op}), and, then, the pair (D_{ip}, D_{op}) with the best performance metric is selected; this selection method is referred as exhDP). Fig. 5a–f demonstrate that two commutation-pruning techniques (related to hypotheses H_{11} and H_{12}) require the same or smaller number of arithmetic operations than that specified by hypothesis H_9. Next, it is necessary to make a choice between H_{11} and H_{12}.

Graphs in Fig. 5 indicate that the number of operations required to perform our commutation-pruning technique ($\text{DFT}_{\text{COMM–DIF–DIT–Pr}}$ and $\text{DFT}_{\text{COMM–DIT–DIF–Pr}}$) with the selected decomposition factors using the roughDP method are equal to or slighty greater than those, in which the decomposition factors are specified employing exhDP. The differences correspond to the regions where the commutation conditions prescribe performing the pruned decomposed transform instead of the 2BF filtering method.

The $\text{DFT}_{\text{DIT–DIF–Pr}}$ modality requires the same or a smaller number of arithmetic operations than the competing $\text{DFT}_{\text{DIF–DIT–Pr}}$ for all the cases where the pruned decomposed transform is performed (as it follows from the data reported in Fig. 5). Since the same decomposition factors (D_{ip}, D_{op}) are used in both pruned decomposed transforms, it is sufficient to compare the number of required operations by their input and output stages ($\text{OPER}_{\text{input}} + \text{OPER}_{\text{output}}$) reported in Table 2 to distinguish which one is the most efficient. The comparison for the cases $L_i \le D_{op}$ and $L_o \le D_{ip}$ is not needed since in such scenarios, a direct or recursive method is employed instead of a pruned decomposed transform. For scenarios with $D_{op} < 4$, both pruned decomposed transforms require the same number of arithmetic operations for their execution. Otherwise, for $D_{op} \ge 4$, the execution of $\text{DFT}_{\text{DIF–DIT–Pr}}$ requires $2D_{ip}D_{op} - 8D_{ip} - 2D_{op} + 8$ more arithmetic operations than $\text{DFT}_{\text{DIT–DIF–Pr}}$ demonstrating that the latter manifests always the same or a better performance. Thus, from the combinational permutation analysis, it follows that it is always desirable to perform the $\text{DFT}_{\text{DIT–DIF–Pr}}$ when a pruned decomposed transform would be required. In the following section, an efficient implementation of that proposed unified commutation-pruning technique is detailed considering that the pruned decomposed transform is implemented using the $\text{DFT}_{\text{DIT–DIF–Pr}}$. In summary, we now resume

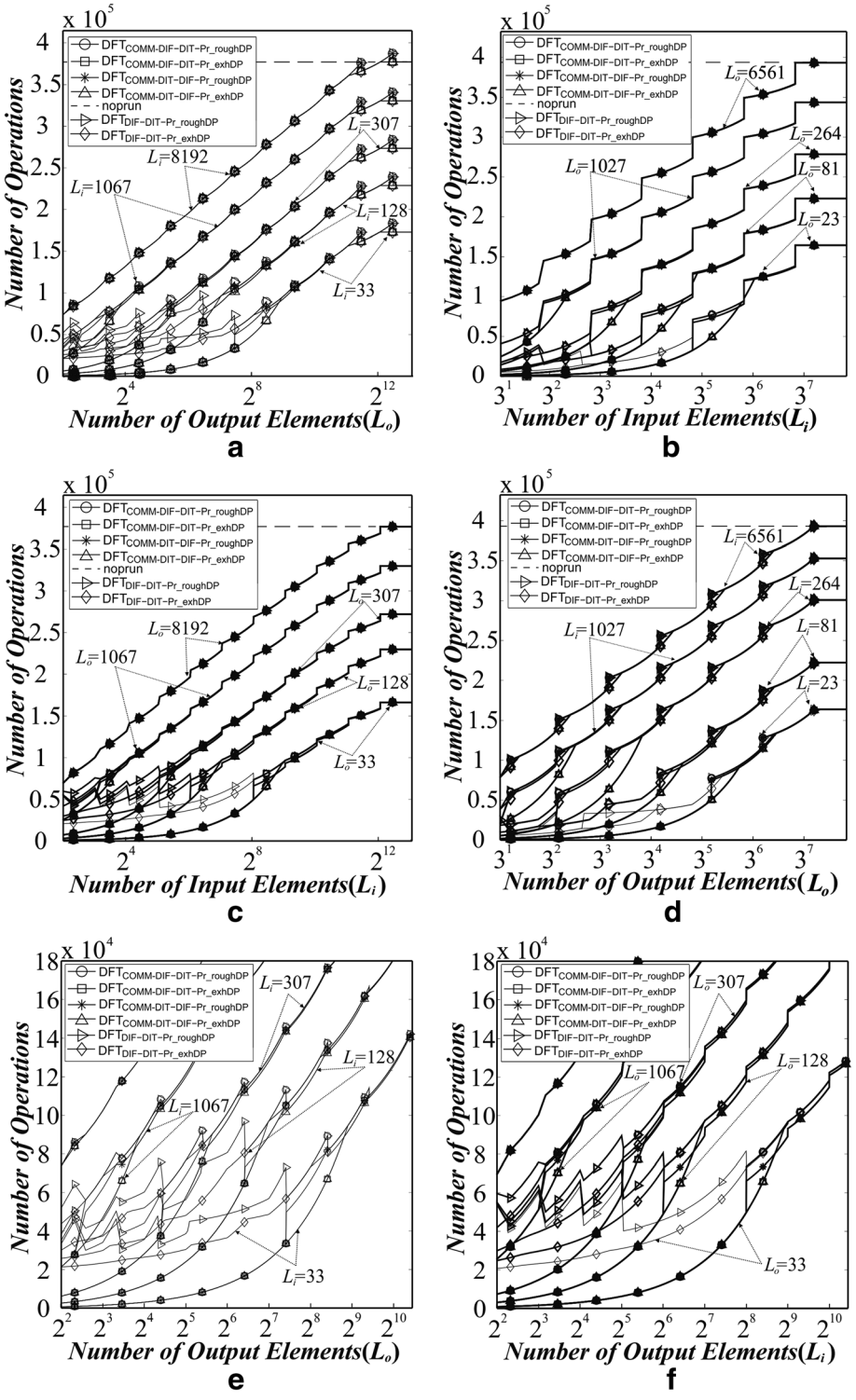

Fig. 5 Number of arithmetic operations required to compute the $DFT_{DIF\text{-}DIT\_Pr}$, the $DFT_{COMM-DIF-DIT-Pr}$, and the $DFT_{COMM-DIT-DIF-Pr}$: **a** for a constant value of L_i and different tested values of $L_o = \{1, 2, \ldots, N\}$ when $N = 8192$; **b** for a constant value of L_o and different tested values of $L_i = \{1, 2, \ldots, N\}$ when $N = 6561$; **c** for a constant value of L_o and different tested values of $L_i = \{1, 2, \ldots, N\}$ when $N = 8192$; **d** for a constant value of L_i and different tested values of $L_o = \{1, 2, \ldots, N\}$ when $N = 6561$; **e** zoom of Fig. 5a; and **f** zoom of Fig. 5c

that the performed combinational hypothesis testing-based optimal selection of the preferable computational structure of the decomposed DFTs made the decision in favor of hypothesis H_{12}; this yields the proposed $DFT_{COMM-DIT-DIF-Pr}$ method (referred further on for simplicity as DFT_{COMM}) with the highest possible computational efficiency. Being the optimal decision of the performed "brute force search" based testing of all feasible hypotheses, this method is guaranteed to be globally optimal one and thus is strongly recommended for performing the required commuting between three techniques to implement the overall composite DFT in the following arrangement mode: the direct method, the recursive method, and the pruned decomposed transform implemented via $DFT_{DIT-DIF-Pr}$.

4 Comparison with other competing algorithms

A variety of competing methods for pruning the DFTs in arbitrary (non-sparse) computational scenarios have been addressed in the literature (see [1, 3, 13–19, 23, 24]). In [24], the $FFT_{DIF-DIT-TD}$ modality (that we here refer to as $DFT_{DIF-DIT-Pr}$) was proposed as an alternative technique for pruning the input and/or the output of DFTs. That method [24] was compared with other pruning techniques reported in the literature until 2009. Comparisons of the methods proposed by Bouguezel et al. [15], Fan et al. [16], Sreenivas et al. [17], Roche [18], and the $DFT_{DIF-DIT-Pr}$ reported in [24] demonstrated that the $DFT_{DIF-DIT-Pr}$ modality requires fewer arithmetic operations than those of [15–17], while attaining the operational performances similar to that of [18]. Additionally, in Section 3, it was corroborated that our proposed DFT_{COMM} technique requires equal or less arithmetic operations than [24]. Here beneath, we compare our approach with the recently reported most prominent competing pruning methods.

4.1 Comparisons with pruning-based algorithms

The first competing algorithm for pruning the output of a SRFFT was reported in [3]. That so-called $SRFFT_{pruning}$ algorithm was developed for an implicit restriction that only a few consecutive output components (a number L equal to a power of two) are required. Fig. 6 reports the number of arithmetic operations required to perform $SRFFT_{pruning}$ in comparison with our unified DFT_{COMM} method for multiple output pruning examples using the decomposition factors (D_{ip}, D_{op}) evaluated via the roughDP method and those specified by the exhDP method, respectively.

In both cases, it is considered that the DFTs of length P required by the intermediate stage of the pruned decomposed transform have been implemented by applying the split-radix FFT, e.g., [26]. Therefore, the total number of arithmetic operations required by our proposed DFT_{COMM} method in comparison with the competing pruning-based algorithms can be found in Table 4. The savings in the number of arithmetic operations attained with the new developed DFT_{COMM} technique are reported in Tables 5 and 6.

From Fig. 6, one can deduce that our proposed DFT_{COMM} method requires fewer arithmetic operations than the competing $SRFFT_{pruning}$ method in almost all the test cases (with the only one exception for the case $L_o = N/2$ and $L_o = N/4$). Next, Tables 5 and 6 report the savings in the number of arithmetic operations attained with our DFT_{COMM} in comparison with the competing

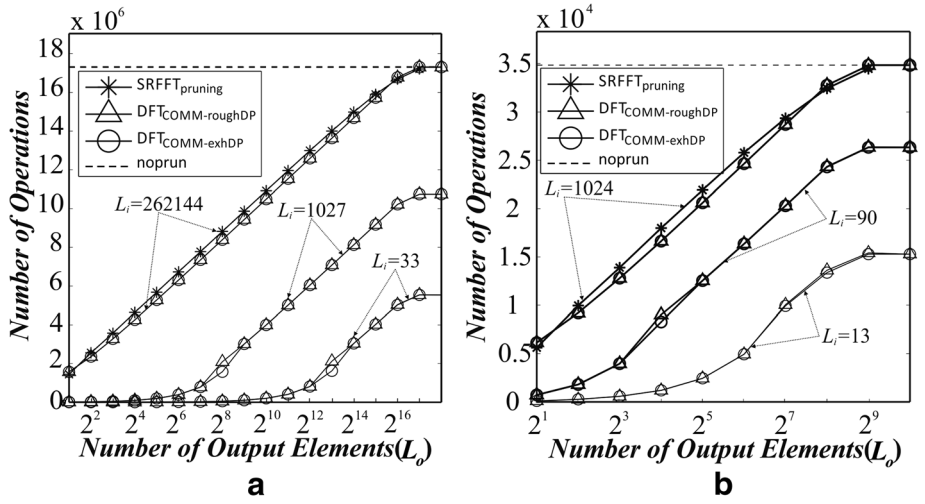

Fig. 6 Number of arithmetic operations required to perform the DFT_{COMM}, SRFFT(noprun), and the $SRFFT_{pruning}$ algorithms; parameters D_{ip} and D_{op} are selected using roughDP and exhDP methods for a constant value of L_i, and different tested values of $L_o = \{2^1, 2^2, \ldots, N\}$: **a** for $N = 262{,}144$ and $L_i = \{33, 1027, 262{,}144\}$; **b** for $N = 1024$ and $L_i = \{13, 90, 1024\}$

Table 4 Total number of arithmetic operations required to compute the $SRFFT_{pruning}$, $SRFFT_{pruning\text{-}time\text{-}shift}$, and DFT_{COMM} algorithms

Algorithm	Number of required real arithmetic operations	Conditions
$SRFFT_{pruning}$	$6\left\{\sum_{k=1}^{d}\left[N_{B(k)}N_{W(k)}\right]+\frac{1}{2}\left(N_{B(d+1)}N_{W(d+1)}\right)\right\}+2\left\{N\cdot d+\sum_{k=0}^{r-(d+1)}\left[L2^{k}\right]\right\}$	–
$SRFFT_{pruning\text{-}time\text{-}shift}$	$6\left\{\sum_{k=1}^{r-d}\left[N_{B(k)}2L\right]+\sum_{k=r-d+1}^{r-1}\left\{N_{B(k)}\left[N_{W(k)}+2\right]\right\}\right\}+2\{N(d-1)+N\}$	–
DFT_{COMM}	$6(L_o-1)(L_i-1)+2L_o(L_i-1)$	$\{(L_i \le D_{op})\vert(L_o \le D_{ip})\}\ \&\ (L_i<4)$
	$(L_o-1)(2L_i+2)+2(L_i-1)+(L_o-1)(4L_i-2)$	$\{(L_i \le D_{op})\vert(L_o \le D_{ip})\}\ \&\ (L_i \ge 4)$
	$6(L_i-D_{op})(D_{ip}-1)+D_{ip}D_{op}(4P\log_2(P)-6P+8)+6(L_o-1)(D_{op}-1)+2L_o(D_{op}-1)$	$\{(L_i>D_{op})\ \&\ (L_o>D_{ip})\}\ \&\ (D_{op}<4)$
	$6(L_i-D_{op})(D_{ip}-1)+D_{ip}D_{op}(4P\log_2(P)-6P+8)+(L_o-1)(2D_{op}+2)+2(D_{op}-1)+$	$\{(L_i>D_{op})\ \&\ (L_o>D_{ip})\}\ \&\ (D_{op}\ge 4)$

Where:
- $N \rightarrow$ Length of the full transform
- $r=\log_2(N) \rightarrow$ Number of stages
- $L=2^d \rightarrow$ Number of consecutive outputs that should be computed for $SRFFT_{pruning}$ algorithm or number of consecutive non-zero inputs for $SRFFT_{pruning\text{-}time\text{-}shift}$ algorithm
- $k \rightarrow$ Stage k
- $N_{B(k)}=\frac{2^{k-1}+(-1)^k}{3} \rightarrow k=1,2,\ldots,r \rightarrow$ Number of twiddle factor blocks
- $N_{W(k)}=2(2^{r-k}-1) \rightarrow k=1,2,\ldots,r \rightarrow$ Number of twiddle factors for each block
- $L_i \rightarrow$ Number of consecutive input elements different from zero
- $L_o \rightarrow$ Number of consecutive output that should be computed
- D_{ip} and $D_{op} \rightarrow$ Integer decomposition factors
- $N/D_{ip}D_{op} \equiv P$

SRFFT and the $SRFFT_{pruning}$ techniques. In the scenarios with $L_o=N$ and $L_i=\{2^1, 2^2,\ldots, N\}$, the DFT_{COMM} algorithm manifests 2.96 and 2.73 % savings in the number of arithmetic operations in comparison with the $SRFFT_{pruning}$ for $N=\{262,144, 1024\}$, respectively.

In other cases, from Table 5, it follows that in the scenarios with $L_i=1027$, $L_i=33$, and $L_o=\{2^1, 2^2,\ldots, N\}$, the $SRFFT_{pruning}$ method fails to deliver a result at all. Thus, from Table 5, it follows that in the cases when $L_i=N=262,144$, $L_i=1027$, $L_i=33$, and $L_o=\{2^1, 2^2,\ldots, N\}$, the DFT_{COMM} algorithm produces savings of 42.76, 75.02, and 91.35 %, respectively, in the number of arithmetic operations required to compute the composite length DFT in comparison with the competing SRFFT algorithm. Furthermore, from Table 6, it follows that in the scenarios with $L_i=90$, $L_i=13$, and $L_o=\{2^1, 2^2,\ldots, N\}$, the $SRFFT_{pruning}$ method fails to deliver a result at all. Thus, from Table 6, it follows that in the cases when $L_i=N=1024$, $L_i=90$, $L_i=13$, and $L_o=\{2^1, 2^2,\ldots, N\}$, the DFT_{COMM} algorithm produces savings of 36.48, 59.30, and 81.65 %, respectively, in the number of arithmetic operations required to compute the composite length DFT in comparison with the competing SRFFT algorithm.

Yuan et al., in [14], proposed another competing, the so-called $SRFFT_{pruning-time-shift}$ method via modifying the $SRFFT_{pruning}$ employing a time shifting approach that yields the input pruning algorithm based on the SRFFT methodology for L consecutive non-zero input elements. It is noteworthy to stress that the $SRFFT_{pruning-time-shift}$ approach implicitly assumes that lengths L and N may take values equal to the power of two only.

Figure 7 reports the number of required arithmetic operations to execute our proposed unified DFT_{COMM} method and those required by the competing pruned DFTs of [14]. These results verify that our approach requires fewer arithmetic operations than those required to perform the $SRFFT_{pruning-time-shift}$ algorithm in all the reported tests. Again, it is implicitly assumed that the DFTs of length P involved in the $DFT_{DIT-DIF-Pr}$ used by our DFT_{COMM} have been computed using the split-radix FFT [26], as reported in Table 4.

Table 5 Savings in the number of arithmetic operations attained with the DFT_{COMM} algorithm in comparison with the competing SRFFT (noprun) and $SRFFT_{pruning}$ methods for $N=262,144$

DFT_{COMM} in comparison with:	L_o	L_i	Savings
SRFFT(noprun)	$\{2^1,2^2,\ldots, N\}$	$N=2^{18}$	DFT_{COMM}, 42.76 % with output pruning
$SRFFT_{pruning}$			DFT_{COMM}, 2.96 % with output pruning
SRFFT(noprun)		1027	DFT_{COMM}, 75.02 % with input-output pruning at the same time
$SRFFT_{pruning}$			$SRFFT_{pruning}$ fails to deliver a result
SRFFT(noprun)		33	DFT_{COMM}, 91.35 % with input-output pruning at the same time
$SRFFT_{pruning}$			$SRFFT_{pruning}$ fails to deliver a result

Table 6 Savings in the number of arithmetic operations attained with the DFT_{COMM} algorithm in comparison with the competing SRFFT (noprun) and $SRFFT_{pruning}$ methods for $N = 1024$

DFT_{COMM} in comparison with:	L_o	L_i	Savings
SRFFT(noprun)	$\{2^1,2^2,\ldots, N\}$	$N = 2^{10}$	DFT_{COMM}, 36.48 % with output pruning
$SRFFT_{pruning}$			DFT_{COMM}, 2.73 % with output pruning
SRFFT(noprun)		90	DFT_{COMM}, 59.30 % with input-output pruning at the same time
$SRFFT_{pruning}$			$SRFFT_{pruning}$ fails to deliver a result
SRFFT(noprun)		13	DFT_{COMM}, 81.65 % with input-output pruning at the same time
$SRFFT_{pruning}$			$SRFFT_{pruning}$ fails to deliver a result

Next, Tables 7 and 8 report the savings in the number of arithmetic operations attained with our DFT_{COMM} in comparison with the competing SRFFT and the $SRFFT_{pruning-time-shift}$ techniques. In the scenarios with $L_o = N$ and $L_i = \{2^1, 2^2,\ldots, N\}$, the DFT_{COMM} algorithm manifests 5.11 and 8.71 % savings in the number of arithmetic operations in comparison with the $SRFFT_{pruning-time-shift}$ for $N = \{262,144, 1024\}$, respectively.

In other test cases, from Tables 7 and 8, it follows that for $L_o = \{1027, 90\}$, $L_o = \{33, 13\}$, and $L_i = \{2^1, 2^2,\ldots, N\}$, the $SRFFT_{pruning-time-shift}$ algorithm fails to deliver a result at all. Furthermore, from Table 7, it follows that in the scenarios with $L_o = \{N, 1027, 33\}$ and $L_i = \{2^1, 2^2,\ldots, N\}$, our DFT_{COMM} attains 43.26, 76.24, and 92.11 % savings for $N = 262,144$, respectively, in the number of arithmetic operations required to compute the composite length DFT. In addition, from Table 8, it follows that in the scenarios with $L_o = \{N, 90, 13\}$ and $L_i = \{2^1, 2^2,\ldots, N\}$, our DFT_{COMM} attains 38.22, 59.22, and 82.45 % savings for $N = 1024$, respectively, in the number of arithmetic operations required to compute the composite length DFT.

Note that our DFT_{COMM} always requires fewer arithmetic operations than the competing $SRFFT_{pruning}$ and $SRFFT_{pruning-time-shift}$ algorithms due to the different butterfly schemes employed to implement the split-radix FFT algorithms [26] and the unified commutation-pruning technique employed (see Section 3). The $SRFFT_{pruning}$ and $SRFFT_{pruning-time-shift}$ algorithms perform the two-butterfly scheme [26], while our $DFT_{DIT-DIF-Pr}$ algorithm employs the three-butterfly scheme to achieve a reduction in the number of arithmetic operations required to implement the DFT_P blocks. Furthermore, graphs of Fig. 6 report that the $SRFFT_{pruning}$ algorithm fail to deliver a result at all in the scenarios with L equal to N due to their algorithmic construction as reported by the authors of [3]. For this reason, this algorithm cannot present a valid value for the last test of L_o (it is simply unable to stop to prune at all). In addition, Fig. 6 reports minimal differences between the numbers of arithmetic operations attained by the DFT_{COMM} evaluated using the roughDP- or exhDP-based selection for specifying D_{ip} and D_{op}. In summary, the

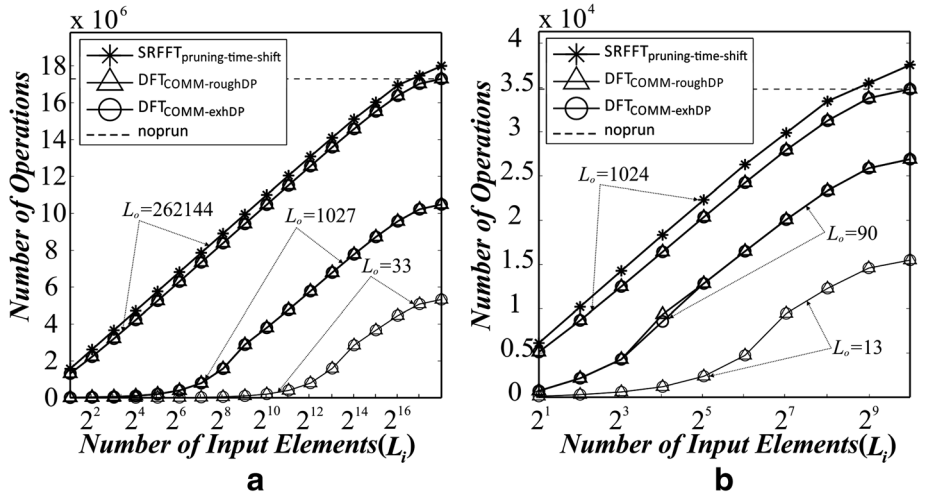

Fig. 7 Number of arithmetic operations required to perform the DFT_{COMM}, SRFFT(noprun), and the $SRFFT_{pruning}$ algorithms; parameters D_{ip} and D_{op} are selected using roughDP and exhDP methods for a constant value of L_o, and different tested values of $L_i = \{2^1, 2^2, \ldots, N\}$: **a** for $N = 262,144$ and $L_o = \{33, 1027, 262,144\}$; **b** for $N = 1024$ and $L_o = \{13, 90, 1024\}$

Table 7 Savings in the number of arithmetic operations attained with the DFT_{COMM} algorithm in comparison with the competing SRFFT (noprun) and $SRFFT_{pruning\text{-}time\text{-}shift}$ methods for $N = 262{,}144$

DFT_{COMM} in comparison with:	L_i	L_o	Saving
SRFFT(noprun)	$\{2^1,2^2,\ldots, N\}$	$N = 2^{18}$	DFT_{COMM}, 43.26 % with input pruning
$SRFFT_{pruning\text{-}time\text{-}shift}$			DFT_{COMM}, 5.11 % with input pruning
SRFFT(noprun)		1027	DFT_{COMM}, 76.24 % with input-output pruning at the same time
$SRFFT_{pruning\text{-}time\text{-}shift}$			$SRFFT_{pruning\text{-}time\text{-}shift}$ fails to deliver a result
SRFFT(noprun)		33	DFT_{COMM}, 92.11 % with input-output pruning at the same time
$SRFFT_{pruning\text{-}time\text{-}shift}$			$SRFFT_{pruning\text{-}time\text{-}shift}$ fails to deliver a result

number of arithmetic operations required to compute the $SRFFT_{pruning}$, $SRFFT_{pruning\text{-}time\text{-}shift}$, and DFT_{COMM} algorithms can be found in Table 4.

4.2 Comparison with the SFFT-related algorithms

In a context of pruned DFTs, real-world sensing scenarios are characterized by the uncertainties attributed to zero-padded input data acquisition modes with variable composite length windowing of the input and/or output Fourier transform sequences, in general cases, with non-sparse Fourier spectrum [10–12]. In contrast, the celebrated SFFT method developed and featured in [20] presumes "sparsity" of the Fourier spectrum that requires that majority of the Fourier coefficients are zeros or negligible; e.g., the authors of [20] exemplified such sparsity level at approximately 89 %, i.e., up to 89 % of the Fourier transform coefficients are to be zeroes or negligible for operability of their SFFT. Otherwise, the DFT should be specified and treated as a *non-sparse* transform.

Currently, a family of novel efficient algorithms for computing the FFTs applicable for sparse sensing scenarios when only a few Fourier transform coefficients (k_s largest coefficients of the N-length Fourier transform) of the input signal x are different from zero have been developed [20, 21], which compose a family of the so-called SFFT methods. To compute a reliable SFFT for typical high $N > 2^{10}$, the sparsity level constraint requires that majority of the Fourier coefficients are zeros [20] (or negligible to be discarded). Such model assumptions are valid, for example, in video compressing applications [20]. Therefore, if majority of the Fourier transform coefficients are supposed to be zeros or can be discarded, then efficient computing techniques from the SFFT family can be employed. The celebrated algorithms from such a family are the SFFTv1 and the SFFTv2 developed and featured in [20] where the sparsity level was exemplified at 89 % of zero (negligible) Fourier coefficients. In [21], the SFFTv3 and SFFTv4 algorithms were proposed, where some computational improvements were introduced. SFFTv3 was implemented in [28] while the program code for implementation of the SFFTv4 algorithm is not available at this time. Another competing technique for computing of the FFT of sparse (in the frequency domain) signals was addressed in [22] as the so-called FADFT-2 algorithm from the AAFFT library [22]. However, in [20, 21], it was corroborated that the SFFT-related algorithms manifest better operational performances than FADFT-2 of [22].

To perform valid test comparisons between the SFFTv1, SFFTv2, SFFTv3, and the DFT_{COMM} algorithms, those should be tested under the same conditions and constrains. Here, we use the following feasible constraints: the values of N vary as follows: $N = \{2^6, 2^7,\ldots, 2^{20}\}$ and $k_s = L_o$, where L_o represents the number of consecutive output coefficients to be calculated. In different test scenarios, the SFFTv1, SFFTv2, and SFFTv3 algorithms deliver successful results: the first of them for $N = \{2^{13}, 2^{14},\ldots, 2^{20}\}$ and $k_s = L_o = 50$, the second of them for $N = \{2^{13}, 2^{14},\ldots, 2^{20}\}$ and $k_s = L_o = 50$, and finally, the third of them for $N = \{2^{10}, 2^{11},\ldots, 2^{20}\}$ and $k_s = L_o = 50$, respectively. Furthermore, it was experimentally corroborated that the DFT_{COMM} algorithm was able to deliver efficient results in all such tested sparse scenarios, as reported in Table 9.

Table 8 Savings in the number of arithmetic operations attained with the DFT_{COMM} algorithm in comparison with the competing SRFFT (noprun) and $SRFFT_{pruning\text{-}time\text{-}shift}$ methods for $N = 1024$

DFT_{COMM} in comparison with:	L_i	L_o	Saving
SRFFT(noprun)	$\{2^1,2^2,\ldots, N\}$	$N = 2^{10}$	DFT_{COMM}, 38.22 % with input pruning
$SRFFT_{pruning\text{-}time\text{-}shift}$			DFT_{COMM}, 8.71 % with input pruning
SRFFT(noprun)		90	DFT_{COMM}, 59.22 % with input-output pruning at the same time
$SRFFT_{pruning\text{-}time\text{-}shift}$			$SRFFT_{pruning\text{-}time\text{-}shift}$ fails to deliver a result
SRFFT(noprun)		13	DFT_{COMM}, 82.45 % with input-output pruning at the same time
$SRFFT_{pruning\text{-}time\text{-}shift}$			$SRFFT_{pruning\text{-}time\text{-}shift}$ fails to deliver a result

Table 9 Comparisons of the SFFTv1, SFFTv2, SFFTv3, and DFT_{COMM} algorithms for different sizes (N) of the signal x, with $N = L_i = \{2^6, 2^7, \ldots, 2^{20}\}$ and $k_s = L_o = 50$

$L_i = N$	$k_s = L_o$	SFFTv1	SFFTv2	SFFTv3	DFT_{COMM}
$2^6 = 64$	50	*	*	*	✓
$2^7 = 128$	50	*	*	*	✓
$2^8 = 256$	50	*	*	*	✓
$2^9 = 512$	50	*	*	*	✓
$2^{10} = 1024$	50	*	*	✓	✓
$2^{11} = 2048$	50	*	*	✓	✓
$2^{12} = 4096$	50	*	*	✓	✓
$2^{13} = 8192$	50	✓	✓	✓	✓
$2^{14} = 16{,}384$	50	✓	✓	✓	✓
$2^{15} = 32{,}768$	50	✓	✓	✓	✓
$2^{16} = 65{,}536$	50	✓	✓	✓	✓
$2^{17} = 131{,}072$	50	✓	✓	✓	✓
$2^{18} = 262{,}144$	50	✓	✓	✓	✓
$2^{19} = 524{,}288$	50	✓	✓	✓	✓
$2^{20} = 1{,}048{,}576$	50	✓	✓	✓	✓

(*) The program execution is aborted
(✓) The program execution is successful

In addition, DFT computations for other sparse test scenarios with different values of N and k_s were run, in particular, for $N = L_i = \{2^{13}, 2^{14}, \ldots, 2^{17}\}$ and $k_s = L_o = \{1, 2, \ldots, k_{smax}\}$ with $k_{smax} = 11$ % of N. The test scenarios for the SFFT algorithms delivered successful results only for a few tested values of k_s. For example, the SFFTv1 algorithm is executed successfully for $N = \{2^{13}, 2^{15}\}$ and $k_s = \{1, 2, \ldots, 50\}$, for $N = 2^{14}$ and $k_s = \{1, 2, \ldots, 50\} \cup \{56, 57, \ldots, 63\}$, for $N = 2^{16}$ and $k_s = \{1, 2, \ldots, 50\} \cup \{64, 65, \ldots, 97\}$, and for $N = 2^{17}$ and $k_s = \{1, 2, \ldots, 74\}$.

The SFFTv2 algorithm is executed successfully for $N = \{2^{13}, 2^{14}, \ldots, 2^{17}\}$ and $k_s = \{1, 2, \ldots, 50\}$, while, the SFFTv3 algorithm performed successfully for $N = 2^{13}$ and $k_s = \{4, 5, \ldots, 673\}$, for $N = 2^{14}$ and $k_s = \{4, 5, \ldots, 1346\}$, for $N = 2^{15}$ and $k_s = \{4, 5, \ldots, 2692\}$, for $N = 2^{16}$ and $k_s = \{4, 5, \ldots, 5385\}$, and for $N = 2^{17}$ and $k_s = \{4, 5, \ldots, 10{,}771\}$. Furthermore, the DFT_{COMM} algorithm is executed successfully for all test cases (for $N = \{2^{13}, 2^{14}, \ldots, 2^{17}\}$ in combination with all $k_s = \{1, 5, \ldots, k_{smax}\}$, as follows from the data reported in Table 10.

Table 11 reports the absolute average errors attained with the SFFTv1, SFFTv2, SFFTv3, and DFT_{COMM} algorithms, for $N = \{2^{13}, 2^{14}, \ldots, 2^{18}\}$ and $k_s = L_o = 50$. In all test cases, the FFTW algorithm from [29] was used as a reference for computing the absolute error measures.

From the data reported in Table 11, it follows that for $N = 8192$ and $k_s = L_o = 50$, the SFFTv1 and SFFTv2 algorithms manifest very close absolute error values; in particular, the attained average absolute error values were 5.6162×10^{-5} and 5.0689×10^{-5}, respectively. However, the SFFTv3 attains a lower absolute average error values than other SFFT versions. It is noteworthy to mention that the lowest absolute average error was attained with the DFT_{COMM} algorithm at a value of 2.7642×10^{-10}.

Table 10 Comparisons of the SFFTv1, SFFTv2, SFFTv3, and DFT_{COMM} algorithms for different sizes (N) of the signal x, with $N = L_i = \{2^{13}, 2^{14}, \ldots, 2^{17}\}$ and $k_s = L_o = \{1, 2, \ldots, k_{smax}\}$ in the tested sparse scenarios with $k_{smax} \sim 11$ % of N

$L_i = N$	$1 \le k_s \le k_{smax}$	SFFTv1
2^{13}	$1 \le k_s \le 901$	$k_s = \{1, 2, \ldots, 50\}$
2^{14}	$1 \le k_s \le 1802$	$k_s = \{1, 2, \ldots, 50\} \cup \{56, 57, \ldots, 63\}$
2^{15}	$1 \le k_s \le 3604$	$k_s = \{1, 2, \ldots, 50\}$
2^{16}	$1 \le k_s \le 7208$	$k_s = \{1, 2, \ldots, 50\} \cup \{64, 65, \ldots, 97\}$
2^{17}	$1 \le k_s \le 14{,}417$	$k_s = \{1, 2, \ldots, 74\}$
$\mathbf{L_i = N}$	$\mathbf{1 \le k_s \le k_{smax}}$	**SFFTv2**
2^{13}	$1 \le k_s \le 901$	$k_s = \{1, 2, \ldots, 50\}$
2^{14}	$1 \le k_s \le 1802$	$k_s = \{1, 2, \ldots, 50\}$
2^{15}	$1 \le k_s \le 3604$	$k_s = \{1, 2, \ldots, 50\}$
2^{16}	$1 \le k_s \le 7208$	$k_s = \{1, 2, \ldots, 50\}$
2^{17}	$1 \le k_s \le 14{,}417$	$k_s = \{1, 2, \ldots, 50\}$
$\mathbf{L_i = N}$	$\mathbf{1 \le k_s \le k_{smax}}$	**SFFTv3**
2^{13}	$1 \le k_s \le 901$	$k_s = \{4, 5, \ldots, 673\}$
2^{14}	$1 \le k_s \le 1802$	$k_s = \{4, 5, \ldots, 1346\}$
2^{15}	$1 \le k_s \le 3604$	$k_s = \{4, 5, \ldots, 2692\}$
2^{16}	$1 \le k_s \le 7208$	$k_s = \{4, 5, \ldots, 5385\}$
2^{17}	$1 \le k_s \le 14{,}417$	$k_s = \{4, 5, \ldots, 10{,}771\}$
$\mathbf{L_i = N}$	$\mathbf{1 \le k_s \le k_{smax}}$	$\mathbf{DFT_{COMM}}$
2^{13}	$1 \le k_s \le 901$	$k_s = \{1, 2, \ldots, k_{smax}\}$
2^{14}	$1 \le k_s \le 1802$	$k_s = \{1, 2, \ldots, k_{smax}\}$
2^{15}	$1 \le k_s \le 3604$	$k_s = \{1, 2, \ldots, k_{smax}\}$
2^{16}	$1 \le k_s \le 7208$	$k_s = \{1, 2, \ldots, k_{smax}\}$
2^{17}	$1 \le k_s \le 14{,}417$	$k_s = \{1, 2, \ldots, k_{smax}\}$

In addition, Fig. 8 reports the absolute values of errors of the compared tested SFFTv3 and the DFT_{COMM} algorithms for $N = 8192$ and $k_s = L_o = 50$ under the same sparse computing scenarios.

On the other hand, the SFFT-related algorithms demonstrate reliable operation for specific input parameter combinations, i.e., they are dependent on the combination of the dimension N of the input signal x, and the sparsity factor k_s. In contrast, the DFT_{COMM} algorithm manifests the operational robustness in the sense that it does not subject to any of such dimensional limitation and demonstrated perfect operational performances in all tested harsh (non-sparse) computational scenarios. Furthermore, all SFFT-related algorithms are probabilistic-type techniques [20, 21], in which the desired k_s largest coefficients of the

Table 11 Average absolute errors attained in sparse scenarios with the SFFTv1, SFFTv2, SFFTv3, and DFT_{COMM} algorithms for $N = \{2^{13}, 2^{14}, \ldots, 2^{18}\}$ and $k_s = L_o = 50$

$L_i = N$	$k_s = L_o$	SFFTv1	SFFTv2	SFFTv3	DFT_{COMM}
		AbsError	AbsError	AbsError	AbsError
2^{13}	50	5.6162×10^{-5}	5.0689×10^{-5}	2.4973×10^{-5}	2.7642×10^{-10}
2^{14}	50	7.0000×10^{-4}	6.3012×10^{-4}	4.9943×10^{-5}	1.8228×10^{-9}
2^{15}	50	2.8526×10^{-4}	2.5407×10^{-4}	9.9883×10^{-5}	1.6704×10^{-8}
2^{16}	50	3.7305×10^{-4}	3.6885×10^{-4}	1.9976×10^{-4}	1.5567×10^{-7}
2^{17}	50	4.9801×10^{-7}	4.8631×10^{-7}	1.5437×10^{-7}	2.6652×10^{-10}
2^{18}	50	0.0023	0.0023	7.9905×10^{-4}	8.2515×10^{-6}

Fourier spectrum of the input sequence are reconstructed (approximated) with a *high* probability (not mandatory with probability one). In contrast, the DFT_{COMM} algorithm is a deterministic technique, and it produces more reliable and accurate results than the family of the SFFT-related algorithms (as demonstrated in Fig. 8 and Tables 10 and 11).

It is also worthwhile to note that presently (in the sparsity-guaranteed computational scenarios only), the SFFT-related algorithms outperform the DFT_{COMM} in the computational speed due to their specially devised execution parallelism [20, 21, 28]. From the family of the SFFT-related algorithms, the SFFTv3 [28] manifests the most speed-up computational performances for any input sequence dimension N and any feasible value k_s in the sparsity-guaranteed scenarios *only*; in particular, when approximately only 8.2 % (or lower number) of the Fourier coefficients of the input signal are significant, thus not discarded (as shown in Table 10). In contrast, in all comparable (sparse or non-sparse) computational scenarios, the DFT_{COMM} algorithm manifested superior accuracy performances (lower absolute error values) than those attained with the SFFT-related algorithms.

In closing, it is noteworthy to mention that in a majority of practical computational scenarios, the savings in the number of arithmetic operations achievable with the optimized unified DFT_{COMM} technique are significant. As a concluding example, refer to the test scenario with $N = 8192$ and $L_i = L_o = 307$ in which case the savings in the total number of required arithmetic operations attainable with the DFT_{COMM} algorithm in comparison with the most prominent competing split-radix FFT algorithm [3, 14, 23, 24] constitute 45 %.

5 Conclusions

We have developed a new technique that carries out an efficient computation of the DFTs of composite lengths of the input and/or output data sequences smaller than the dimension N of the full DFT/FFT. The addressed methodology unifies the commuting, filtering, and pruning paradigms yielding the new DFT_{COMM} method that outperforms the existing competing pruning-decomposition-based techniques in the sense of attainable savings in the number of required arithmetic operations.

Furthermore, our DFT_{COMM} method admits computing the DFT_P blocks at the intermediate stage of the pruned decomposed transform using any existing FFT algorithm.

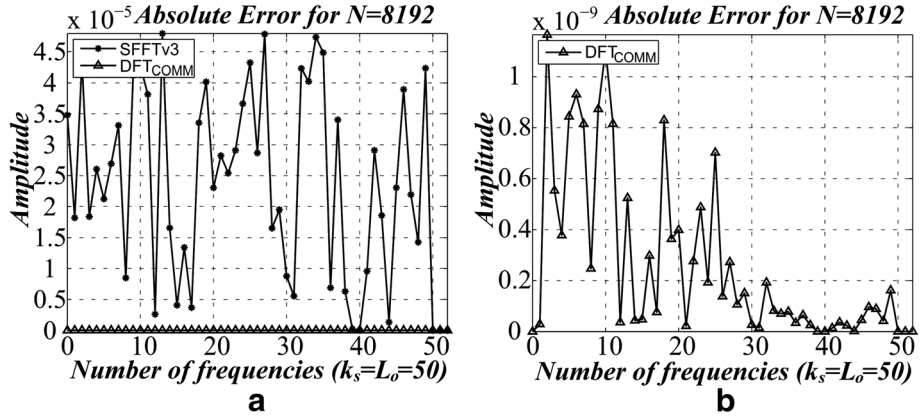

Fig. 8 Measures of absolute error values attained with the SFFTv3 and DFT_{COMM} algorithms in sparse scenarios for $N = 8192$ and $k_s = L_o = 50$: **a** SFFTv3 and DFT_{COMM}; **b** DFT_{COMM}

Based on the performed treatment of the combinational hypotheses testing-type problem regarding all feasible allocation-pruning modalities, the decision in favor of the preferable hypothesis was made that yields the proposed $\mathrm{DFT}_{\mathrm{COMM}}$ method. Being the globally optimal decision making result of testing the complete list of all feasible hypotheses, the $\mathrm{DFT}_{\mathrm{COMM}}$ method guarantees to require a fewer or at most the same number of arithmetic operations for its execution than any other of the competing pruning-decomposition-based methods reported in the literature.

In addition, we have corroborated that, in the scenarios with non-guaranteed sparsity of the data Fourier spectra, the $\mathrm{DFT}_{\mathrm{COMM}}$ method manifests better reliability and accuracy than the family of the celebrated competing SFFT-related algorithms; while in scenarios with severe Fourier spectrum non-sparsity (i.e., when the majority of the data Fourier spectrum coefficients take non-zero values, thus cannot be discarded), the $\mathrm{DFT}_{\mathrm{COMM}}$ technique always outperforms the celebrated SFFT-related algorithms because all those simply fail to execute the program code in such uncertain computational scenarios.

6 Appendix

6.1 Main function

Fig 9 presents the pseudo-code of the main function that commute among the different alternatives to compute the DFT_N ($\mathrm{DFT}_{\mathrm{COMM}}$). When the pruned decomposed transform is not required ($L_i \le D_{op}$ or $L_o \le D_{ip}$), the direct method or the 2BF filtering method could be employed. In both cases, the Fourier coefficient $X(0)$ is computed as a simple addition of the elements in the input sequence $x(n)$.

The directFourier function is used in the scenarios with $L_i < 4$ to compute the remaining Fourier coefficients ($k = 1{:}1{:}L_o - 1$). The 2BF filtering method is implemented when $L_i \ge 4$. The directFourier function carries out the addition of complex multiplications of elements in $x(n)$ by the complex exponential W_N^{nk} defined in (1). The filterFourier function computes each Fourier coefficient by implementing a recursive algorithm similar to the second-order Goertzel algorithm of [25]. In the filterFourier function, the feedback signal is multiplied by the real part of the complex exponentials W_N^{k} and, next, by the conjugate of W_N^{m}. In our modification, the array of complex exponentials W_N^{m} is pre-computed for $m = 0{:}1{:}N - 1$ and stored by duplicating in the vector W of length $2N$ ($W = [W_N^{m}, W_N^{m}]$), in such a way that W_N^{nk} and W_N^{k} could be read from it using nk and k as indexes, respectively. Accessing an element out of the vector W is impossible for these cases, as verified next. L_i is inferior than 4 (or equivalently $L_i \le 3$) when the direct method is used, thus $n \le L_i - 1 \le 2$ and $k \le L_o - 1 \le N - 1$, and consequently $nk \le 2(N-1) < 2N$. This assures that each element of W_N^{nk} can be extracted from W just via accessing the element indexed by nk. Similarly, for $k \le L_o - 1 \le N - 1$, each element of W_N^{k} is directly extracted from W accessing the element indexed by k. In order to

```
////////////////////////////////////////////////////////////////////
//Main function of the proposal to compute the DFT_N//
////////////////////////////////////////////////////////////////////
[X]=DFT(N,L_i,L_o,D_ip,D_op,x,W)
{ if ((L_i<=D_op)|(L_o<=D_ip))
  //The DFT is computed without pruning.
    //Computation of X(0)
    X(0)=x(0);
    for (n=1:1:L_i-1)
      X(0)=x(n)+X(0);
    //Computation of X(k) for 1<=k<=L_o-1
    if (L_i< 4)
      //Using the direct method
      for (k=1:1:L_o-1)
      { X(k)=directFourier(x(0:1:L_i-1),k,L_i,W);}
    else{
      //Using the filtering 2BF method
      for (k=1:1:L_o-1)
      { X(k)=filterFourier(x(0:1:L_i-1),k,L_i,W);}}}
  else{
  //The DFT_DIT-DIF-Pr algorithm is used
    y = InputStage(N,L_i,D_ip,D_op,x,W) ;
    z = IntermediateStage(N,L_o,L_i,D_ip,D_op,y,W)
    X = OutputStage(N,L_o,D_ip,D_op,P,z,W);}}
////////////////////////////////////////////////////////////////////
//Function to compute each Fourier coefficient by//
//                the direct method.                //
////////////////////////////////////////////////////////////////////
[F]=directFourier(f[0:1:L-1],k,L,W)
{ F=f(0);
  nk=k;
  for (n=1:1:L-1)
  { F=F+f(n)×W(nk);
    nk=nk+k;}}
////////////////////////////////////////////////////////////////////
//Function to compute each Fourier coefficient by//
//              the filtering 2BF method.           //
////////////////////////////////////////////////////////////////////
[F]=filterFourier(f[0:1:L-1],k,L,W)
{ t_1=f(L-1);
  t_4=t_1×(2×real(W(k)));
  for (n=L-2:-1:1)
  { t_2=t_1;
    t_1=f(n)+t_4;
    t_3=t_1×(2×real(W(k)));
    t_4=t_3-t_2;}
  t_5=f(0)+t_4;
  F=t_5-t_1×conj(W(k)) ;}}
```

Fig. 9 General computational layout of $\mathrm{DFT}_{\mathrm{COMM}}$

```
////////////////////////////////////////////////////////
//Function to generate the input stage//
////////////////////////////////////////////////////////
[y]=InputStage(N,Li,Dip,Dop,x,W)
//Generation of the elements of y(n1,n2,k1) that
//depends on x(n) for 0<=n<=Dop-1 (n1=n).
{  for (n1=0:1:Dop-1)
   {  for (k1=0:1:Dip-1)
      {  y(n1,0,k1)=x(n1);}}
//Generation of the elements of y(n1,n2,k1) that
//depends on x(n) for Dop<=n<=Li-1.
   n=Dop;
   n2Dop=Dop;
   for (n2=1:1:floor(Li-1)/Dop)-1)
   {  for (n1=0:1:Dop-1)
   //Mapping x(n) to one input of each DFTP
   //of one set of Dip DFTPS
      {  t1=x(n);
         y(n1,n2,0)=t1;
   //Multiplication of x(n) per WM^{p2×k1} and
   //mapping to the others DFTPS of the same
   //set of Dip DFTPS
         n2k1Dop=n2Dop;
         for (k1=1:1:Dip-1)
         {  y(n1,n2,k1)=W(n2k1Dop)×t1;
            n2k1Dop=n2k1Dop+n2Dop;}
         n=n+1;}
      n2Dop=n2Dop+Dop ;}
   n2=floor((Li-1)/Dop);
   for (n1=0:1:((Li-1))Dop)
   //Mapping x(n) to one input of each DFTP
   //of one set of Dip DFTPS
   {  t1=x(n);
      y(n1,n2,0)=t1;
   //Multiplication of x(n) per WM^{p2×k1} and
   //mapping to the others DFTPS of the same
   //set of Dip DFTPS
      n2k1Dop=n2Dop;
      for (k1=1:1:Dip-1)
      {  y(n1,n2,k1)=W(n2k1Dop)×t1;
         n2k1Dop=n2k1Dop+n2Dop ;}
      n=n+1;}}
//Zero padding of the elements of y(n1,n2,k1)
//for Li <=n<=N-1
   for (n1=((Li-1))Dop+1:1:Dop-1)
   {  for (k1=0:1:Dip-1)
      {  y(n1,n2,k1)=0;}}
   for (n2=floor((Li-1)/Dop)+1:1:N/DipDop-1)
   {  for (n1=0:1:Dop-1)
      {  for (k1=0:1:Dip-1)
         {  y(n1,n2,k1)=0;}}}}
```

Fig. 10 Pseudo-code of the InputStage function

```
////////////////////////////////////////////////////////
//Function to generate the output stage//
////////////////////////////////////////////////////////
[X]=OutputStage(N,Lo,Dip,Dop,z,W)
{ //Computation of X(0)
  X(0)=z(0,0,0);
  for (n1=1:1:Dop-1)
     X(0)=z(n1,0,0)+ X(0);
  //Computation of X(k) for 1<=k<=Lo-1
  if (Dop<4)
    //Using the sub-routing to compute X(k)
    //with the directFourier function (replace
    //methodFourier by directFourier).
  else{
    //Using the sub-routing to compute X(k)
    //with the filtering 2BF method (replace
    //methodFourier by filterFourier).  }}
////////////////////////////////////////////////////////////////
// Sub-routing to compute the X(k) for 1<=k<=Lo-1//
////////////////////////////////////////////////////////////////
k=1;
k2=0;
for (k1=1:1:Dip-1)
{  X(k)=methodFourier(z(0:1:Dop-1,k2,k1),k,Dop,W);
   k=k+1;}
if (Lo<=N/Dop)
{  for (k2=1:1:floor((Lo-1)/Dip)-1)
   {  for (k1=0:1:Dip-1)
      {  X(k)=methodFourier(z(0:1:Dop-1,k2,k1),k,Dop,W);
         k=k+1;}}
   k2=floor((Lo-1)/Dip);
   for (k1=0:1:((Lo-1))Dip)
      {  X(k)=methodFourier(z(0:1:Dop-1,k2,k1),k,Dop,W);
         k=k+1;}}
else{
   for (k2=1:1:N/(DipDop)-1)
   {  for (k1=0:1:Dip-1)
      {  X(k)=methodFourier(z(0:1:Dop-1,k2,k1),k,Dop,W);
         k=k+1;}}
   for (k3=1:1:((floor((Lo-1)/(N/Dop))))Dop-1)
   {   for (k2=0:1:N/(DipDop)-1)
       {   for (k1=0:1:Dip-1)
           {  X(k)=methodFourier(z(0:1:Dop-1,k2,k1),k,Dop,W);
              k=k+1;}}}
   k3=((floor((Lo-1)/(N/Dop))))Dop;
   for (k2=0:1:((floor((Lo-1)/Dip)))N/DipDop-1)
   {  for (k1=0:1:Dip-1)
      {  X(k)=methodFourier(z(0:1:Dop-1,k2,k1),k,Dop,W);
         k=k+1;}}
   k2=((floor((Lo-1)/Dip)))N/DipDop
   for (k1=0:1:((Lo-1))Dip)
   {  X(k)=methodFourier(z(0:1:Dop-1,k2,k1),k,Dop,W);
      k=k+1;}}
```

Fig. 11 Pseudo-code of the OuputStage function

avoid multiplications in the generation of the index, nk, the latter is computed by adding k to nk in each iteration of the loop n (inside the function directFourier).

In the scenarios with $L_i > D_{op}$ and $L_o > D_{ip}$, the $\mathrm{DFT}_{\mathrm{DIT-DIF-Pr}}$ is performed to compute the DFT_N. As it was explained previously, the $\mathrm{DFT}_{\mathrm{DIT-DIF-Pr}}$ is performed in three commuting stages: the input stage, the intermediate stage, and the output stage. These stages are executed in a sequential order by calling the InputStage

function, next the IntermediateStage function, and, finally, the OutputStage function.

6.2 InputStage function

The InputStage function generates the inputs to the intermediate $D_{ip}D_{op}$ DFTs of length P (DFT_Ps), resulting in an array of three dimensions $y(n_1, n_2, k_1)$. The pseudo-code for implementing the InputStage function is listed in Fig. 10. The indexes, n_1, n_2, and k_1 are varied using three nested loops ("for" instructions), in such an order that the number of accesses to each element in $x(n)$ is reduced. This is achieved by specifying k_1 for the inner loop, n_1 for the intermediate loop, and n_2 for the outer loop. With this order, once an element in $x(n_1 + D_{op}n_2)$ is loaded, all the inputs of the DFT_Ps that depend on it are generated. To minimize the required computations, the nested loops have been broken down to avoid multiplications by one and the application of if-clauses.

In order to avoid overhead in the generation of the indexes, those are generated by additions only. After the InputStage function has been executed, the intermediate stage should be called.

6.3 Intermediate stage function

The intermediate stage consists in computing D_{ip}-D_{op}DFTs of length $P = N/D_{ip}D_{op}$. This stage could be implemented with any algorithm for computing a DFT. For instance, the split-radix could be used if P is a power of two [26] or the radix-3 could be used if P is a power of three [27]. For a general case, we recommend using the FFTW (the fastest Fourier transform in the west) reported in [29] to compute the $D_{ip}D_{op}DFT_P$s since this is the most efficient algorithm for an arbitrary length DFT. The selected algorithm should be applied over each vector obtained from $y(n_1, 0 : 1 : P - 1, k_1)$ for each value of n_1 and k_1, resulting in a vector with output index k_2 that is stored in the array $z(n_1, k_2, k_1)$. This array is then processed by the OuputStage function.

6.4 OutputStage function

The OutputStage is performed to compute the final Fourier coefficients from the outputs of the D_{ip}-D_{op}DFT$_P$s stored in $z(n_1, k_2, k_1)$. This function is listed in Fig. 11. In fact, the OutputStage function performs the computation of another stage of DFTs, although with a few outputs. As previously mentioned, there are two alternatives to compute each Fourier coefficient from $z(n_1, k_2, k_1)$, using a direct computation or using the 2BF filtering method. Thus, the OutputStage function could employ the direct-Fourier or the filterFourier functions listed in the pseudo-code of Fig. 9 to compute the final Fourier coefficients.

Each Fourier coefficient depends on D_{op} inputs (obtained from $z(n_1, k_2, k_1)$ by varying n_1), so for $D_{op} < 4$, the direct method is desirable; otherwise, the 2BF filtering method is to be executed.

These nested loops should be implemented in the indicated order to specify the indexes of the final Fourier coefficients. Those indexes are obtained by increasing index k by a unit in each iteration of the loop indexed by k_1. The directFourier function utilizes the complex exponential W_N^{nk}, while the filterFourier function involves the complex exponential W_N^{k}. All elements W_N^k and W_N^{nk} are extracted from W using k and nk as indexes, respectively. In order to reduce multiple copies of data and thus to achieve an enhanced efficiency of the algorithm, it is strongly desirable to implement inline functions and passing the arrays elements by reference instead of by value.

Competing interests

The authors declare that they have no competing interests.

Acknowledgements

The authors would like to thank the anonymous reviewers for their constructive criticism and comments that helped to improve the presentation of the paper.

Received: 30 March 2015 Accepted: 11 November 2015
Published online: 26 November 2015

References

1. J Markel, FFT pruning. Audio and Electroacoustics, IEEE Transactions on **19**(4), 305, 311 (1971). doi:10.1109/TAU.1971.1162205
2. V Raghavan, KMM Prabhu, PCW Sommen, Complexity of pruning strategies for the frequency domain LMS algorithm. Signal Processing **86**(10), 2836–2843 (2006). ISSN 0165–1684, http://dx.doi.org/10.1016/j.sigpro.2005.11.015
3. Y Xu; M-S Lim, Split-radix FFT pruning for the reduction of computational complexity in OFDM based cognitive radio system, *in Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, 69-72, May 30-June 2 2010. doi: 10.1109/ISCAS.2010.5537048.
4. FM Henderson, AV Lewis (eds.), *Principles and applications of imaging radar, manual of remote sensing*, vol. 3, 3dth edn. (Willey, NY, 1998)
5. HH Barrett, KJ Myers, *Foundations of image science* (Willey, NY, 2004)
6. YV Shkvarko, Unifying experiment design and convex regularization techniques for enhanced imaging with uncertain remote sensing data–– part I: theory, part II: adaptive implementation and performance issues. IEEE Trans. Geoscience and Remote Sensing **48**(1), 82–111 (2010)
7. A Moni, CJ Bean, I Lokmer, S Rickard, Source separation on seismic data. IEEE Signal Processing Magazine **29**(3), 16–28 (2012)
8. RM Willet, MF Duarte, MA Davenport, RG Baraniuk, Sparsity and structure in hyperspectral imaging. IEEE Signal Processing Magazine **31**(1), 116–126 (2014)
9. Q Zhu, CR Berger, EL Turner, L Pileggi, F Franchetti, Polar format synthetic aperture radar in energy efficient application-specific logic-in-memory, *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2012*, pp. 1557- 1560, 25–30 March 2012. doi: 10.1109/ICASSP.2012.6288189.
10. YV Shkvarko, J Tuxpan, SR Santos, I Yaniez, *High-resolution imaging with uncertain radar measurement data: a doubly regularized compressive sensing experiment design approach,* in IEEE Intern. Symposium on Geoscience and Remote Sensing (IGRSS'2012), Munich, Germany, 6976–6970. (2012). ISBN: 978-1-46731159-51/12
11. YV Shkvarko, J Tuxpan, SR Santos, l_2-l_1 Structured descriptive experiment design regularization based enhancement of fractional SAR imagery. Signal Processing **93**, 3553–3566 (2013). http://dx.doi.org/10.1016/j.sigpro.2013.03.024
12. S Foucart, H Rauhut, *A mathematical introduction to compressive sensing* (Springer, NY-Heidelberg, 2013)
13. DP Skinner, Pruning the decimation in-time FFT algorithm, in IEEE Transactions on Acoustics, Speech and Signal Processing, **24**(2), 193–194 (1976). doi:10.1109/TASSP.1976.1162782

14. L Yuan, X Tian, Y Chen, Pruning split-radix FFT with time shift, *International Conference on Electronics, Communications and Control (ICECC), 2011*, 1581-1586, 9–11 Sept. 2011. doi: 10.1109/ICECC.2011.6066654.
15. S Bouguezel, MO Ahmad, MNS Swamy, Efficient pruning algorithms for the DFT computation for a subset of output samples, *in Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS '03*, vol.4, pp. IV-97, IV-100 vol.4, 25–28 May 2003. doi: 10.1109/ISCAS.2003.1205782.
16. C-P Fan, G-A Su, Pruning fast Fourier transform algorithm design using group-based method, Signal Processing **87**(11), 2781–2798 (2007), ISSN0165-1684, http://dx.doi.org/10.1016/j.sigpro.2007.05.012
17. TV Sreenivas, P Rao, FFT algorithm for both input and output pruning, in IEEE Transactions on Acoustics, Speech and Signal Processing, **27**(3), 291–292 (1979). doi:10.1109/TASSP.1979.1163246
18. C Roche, A split-radix partial input/output fast Fourier transform algorithm, in IEEE Transactions on Signal Processing, **40**(5), 1273, 1276 (1992). doi:10.1109/78.134493
19. L Wang, X Zhou, GE Sobelman, R Liu, Generic mixed-radix FFT pruning, in IEEE Signal Processing Letters, **19**(3), 167, 170 (2012). doi:10.1109/LSP.2012.2184283
20. H Hassanieh, P Indyk, D Katabi, E Price, 2012. Simple and practical algorithm for sparse Fourier transform, in *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms* (SODA '12) Kyoto, Japan, 17-19 Jan, 1183–1194, (2012)
21. H Hassanieh, P Indyk, D Katabi, E Price, Nearly optimal sparse Fourier transform, in *Proceedings of the forty-fourth annual ACM symposium on Theory of computing (STOC '12)*, ACM, New York, (2012), 563–578. doi:10.1145/2213977.2214029. http://doi.acm.org/10.1145/2213977.2214029
22. M Iwen, A Gilbert, M Strauss et al., Empirical evaluation of a sub-linear time sparse DFT algorithm, Communications in Mathematical Sciences **5**(4), 981–998 (2007)
23. HV Sorensen, CS Burrus, Efficient computation of the DFT with only a subset of input or output points, in IEEE Transactions on Signal Processing, **41**(3), 1184–1200 (1993). doi:10.1109/78.205723
24. M Medina-Melendrez, M Arias-Estrada, A Castro, Input and/or output pruning of composite length FFTs using a DIF-DIT transform decomposition, in IEEE Transactions on Signal Processing, **57**(10), 4124, 4128 (2009). doi:10.1109/TSP.2009.2024855
25. AV Oppenheim, RW Schafer, Discrete-time signal processing, (Prentice Hall, 2nd Edition, U.S., 1999)
26. HV Sorensen, M Heideman, CS Burrus, On computing the split-radix FFT, in IEEE Transactions on Acoustics, Speech and Signal Processing, **34**(1), 152–156 (1986). doi:10.1109/TASSP.1986.1164804
27. Y Suzuki, S Toshio, K Kido, A new FFT algorithm of radix 3,6, and 12, in IEEE Transactions on Acoustics, Speech and Signal Processing, **34**(2), 380–383 (1986). doi:10.1109/TASSP.1986.1164826
28. J. Schumacher, M. Püschel, High performance sparse fast Fourier transform, Master´s thesis, ETH Zurich, Department of Computer Science (2013).
29. M Frigo, SG Johnson, The design and implementation of FFTW3, Proceedings of the IEEE **93**(2), 216–231 (2005) doi:10.1109/JPROC.2004.840301