ELSEVIER

Signal Processing 71 (1998) 291–299

SIGNAL PROCESSING

Real-time implementation of the split-radix FFT – An algorithm to efficiently construct local butterfly modules

Pei-Chen Lo*, Yu-Yun Lee

Department of Electrical and Control Engineering, National Chiao Tung University, Taiwan, People's Republic of China

Received 16 April 1998; received in revised form 13 August 1998

Abstract

This paper presents a new technique of real-time Fourier spectral analysis based on the decimation-in-time split-radix fast-Fourier-transform (DIT sr-FFT) butterfly structure. The new algorithm (to be called the 'real-time FFT algorithm'), designed in a multi-tasking environment, efficiently utilizes the computer time. It simultaneously constructs the sr-FFT butterfly structure while the data acquisition proceeds. Hence, it provides a practical, useful approach to analyzing the on-line, time-varying Fourier spectra for the multi-channel electrophysiological signals. The authors propose a strategy of identifying the local butterfly sub-structures (modules) ready to be constructed when only a portion of the input data array is available. In addition, we develop an alternative way to analyze the computational complexity of the real-time FFT algorithm. To evaluate the efficiency of the algorithm, we calculate the number of complex arithmetic operations required to complete the rest butterfly sub-structures upon receipt of the last data point. The result shows that the efficiency of the algorithm increases with N (the FFT size). © 1998 Elsevier Science B.V. All rights reserved.

Zusammenfassung

Dieser Beitrag stellt eine neue Methode für die Echtzeit Fourier Spektralanalyse basierend auf der decimation-in-time split radix fast-Fourier-transform (DIT sr-FFT) Butterfly-Struktur vor. Der neue Algorithmus (im folgenden Echtzeit-FFT-Algorithmus genannt) eignet sich zur Anwendung in Multi-tasking Umgebungen und macht auf effiziente Weise von der verfügbaren Rechenzeit Gebrauch. Das vorgeschlagene Verfahren konstruiert die sr-FFT-Butterfly-Struktur während die Datenaufnahme erfolgt. Die besprochene Methode eignet sich daher besonders zur Echtzeitanalyse der zeitvarianten Fourierspektren von elektrophysiologischen Mehrkanalsignalen. Die Autoren schlagen eine Methode für die Ermittlung der lokalen Butterfly-Substrukturen (Module) vor, die bereits konstruiert werden können wenn erst ein Teil der zu verarbeitenden Daten vorhanden ist. Weiters wird ein alternatives Verfahren zur Analyse des Rechenaufwandes des Echtzeit FFT-Algorithmus vorgeschlagen. Um die Effizienz des Algorithmus zu bewerten wird die Anzahl der komplexwertigen arithmetischen Operationen bestimmt, die nach Empfang des letzten Datenpunktes nötig sind um die restlichen Butterfly-Substrukturen zu vervollständigen. Dieses Resultat zeigt, daß die Effizienz des Algorithmus mit N (FFT-Größe) steigt. © 1998 Elsevier Science B.V. All rights reserved.

* Corresponding author. Tel.: + 886 3 573 1667; fax: + 886 3 571 5998; e-mail: pclo@cc.nctu.edu.tw.

0165-1684/98/$19.00 © 1998 Elsevier Science B.V. All rights reserved.
PII: S0165-1684(98)00152-2

Résumé

Cet article présente une technique nouvelle pour l'analyse spectrale de Fourier en temps réel basée sur la structure papillon de la transformation de Fourier rapide de type décimation temporelle split-radix (DIT sr-FFT). Cet algorithme nouveau (appelé "algorithme FFT en temps réel") conçu pour un environment multi-tâches, utilise de manière efficiente le temps de calcul. Il construit simultanément la structure papillon du sr-FFT tandis que l'acquistion des données se poursuit. De ce fait, il conduit à une approche pratique et utile pour l'analyse en temps réel de spectres de Fourier variant dans le temps de signaux électrophysiologiques multicanaux. Les auteurs proposent une stratégie d'identification des sous structures (modules) papillon locales prêtes à être construites lorsque seule une portion du réseau de données entrantes est disponible. De plus, nous développons une voie alternative pour l'analyse de la complexité en calcul de l'algorithme de FFT en temps réel. Pour évaluer l'efficacité de l'algorithme, nous calculons le nombre d'opérations arithmétiques complexes requises pour compléter les sous-structures papillon restantes après réception de la dernière donnée. Ce résultat montre que l'efficacité de l'algorithme croît avec N (la taille de la FFT). © 1998 Elsevier Science B.V. All rights reserved.

Keywords: Split-radix FFT; Multi-channel electrophysiological signal processing; Real-time frequency analysis; Local butterfly sub-structure

1. Introduction

On-line, real-time processing of digital electrophysiological signals provides a helpful tool for intensive-care monitoring and diagnostic reference in clinics. In electroencephalograph (EEG), the rhythmicity gives a means of quantitatively describing EEG records. EEG frequencies are conveniently classified into four ranges: delta ($f < 4$ Hz), theta ($4\text{ Hz} \leqslant f < 8$ Hz), alpha ($8\text{ Hz} \leqslant f \leqslant 13$ Hz) and beta ($13\text{ Hz} < f$), which constitute the elementary patterns of EEG signals [5]. We very often require the real-time display of the frequency spectra of EEG in clinical application. In general, total number of recording channels employed in clinical application ranges from 8 to 19 channels. With the advance in digital technology and instrumentation, some EEG instruments even have 256 recording channels. To develop the algorithm capable of performing the real-time, multi-channel frequency analysis, one needs to efficiently utilize the computer time in addition to improving the computational efficiency.

Since the radix-2 FFT proposed by Cooley and Tukey [4], a variety of the FFT algorithms have been rapidly developed and implemented with software and hardware approaches [1–3,6–8,12–17, 19,20]. For example, the split-radix FFT (sr-FFT) algorithm derived by Duhamel and Hollmann [6,7] has a simple structure and an explicit theoretical basis. Their idea has been further discussed and extended in [13,15,17,19]. The FFT algorithm made classical spectral analysis practical. Short-time Fourier transform (STFT) based on FFT has been widely used in the commercially available EEG machine to characterize the time-varying frequency spectra of EEG. However, the STFT even based on FFT still requires processing time several orders of magnitude greater than real time. Most of the processing time has been wasted on acquiring the entire epoch of a (multi-channel) signal or for the I/O access to the storage medium when designing the real-time STFT algorithm based on the conventional FFT algorithm. That is, the digital computer is totally devoted to the data acquisition task without handling any arithmetic operation. Only after the entire epoch for STFT analysis is collected, the algorithm starts building the butterfly structure for the FFT computation. The conventional Cooley–Tukey FFT algorithm and the Duhamel–Hollmann sr-FFT algorithm [6,7] were both developed in this manner.

As a matter of fact, the computational time could be further reduced if we simultaneously build some local butterfly structures (to be called the 'butterfly sub-structures' or the 'butterfly modules') while the data acquisition (I/O process) is still under way. The idea is to finish building as many the butterfly sub-structures as possible with the data points on hand. This approach of implementing the FFT algorithm is to be called 'the real-time FFT

algorithm'. To make the real-time technique practicable, one requires that: (1) the butterfly modules at a given stage can be constructed from one or more butterfly modules at the previous stage, and (2) a large portion of the butterfly modules can be constructed without the last few points of the input array. Most well-known FFT computation schemes (e.g., the radix-2, radix-4, prime-factor, and split-radix FFTs) satisfy these two requirements. Among them, the sr-FFT algorithm takes advantage of the mixed radix design [6,7]. As illustrated in the following section, the sr-FFT algorithm is a mixture of the radix-2 and radix-4 FFTs. Hence, it has the advantage of better computational efficiency of the radix-4 FFT, but yet it does not require the FFT size to be an integer power of 4. This paper presents the real-time FFT algorithm, based on the sr-FFT butterfly structure, which builds the on-line butterfly sub-structures utilizing the portion of the input data array available. In addition, the authors develop an alternative way to analyze the computational complexity, which is adopted to evaluate the efficiency of the real-time FFT algorithm.

2. Duhamel–Hollmann DIT sr-FFT

According to the advantages mentioned in the previous section, the DIT sr-FFT is employed. The DIT sr-FFT structure has a horizontally reversed 'L' shape. Considering the convenient case of N an integer power of 2 ($N = 2^r$), an N-point DFT $X[k]$ of data array $x[n]$ is decomposed into one $N/2$-point DFT $X_{2n}[k]$ and two $N/4$-point DFTs $X_{4n+1}[k]$ and $X_{4n+3}[k]$. Thus, the N-point DFT becomes:

$$\begin{aligned} X[k] &= \sum_{n=0}^{N-1} x[n]W_N^{nk} \\ &= \sum_{m=0}^{(N/2)-1} x[2m]W_N^{2mk} + \sum_{m=0}^{(N/2)-1} x[2m+1]W_N^{(2m+1)k} \\ &= \sum_{m=0}^{(N/2)-1} x[2m]W_N^{2mk} + \sum_{m=0}^{(N/4)-1} x[4m+1]W_N^{(4m+1)k} \\ &\quad + \sum_{m=0}^{(N/4)-1} x[4m+3]W_N^{(4m+3)k}, \end{aligned} \tag{1}$$

where $W_N = e^{-j2\pi/N}$ and $0 \leqslant k \leqslant (N-1)$ [11]. The first term is the $N/2$-point DFT of the even-indexed data points in $x[n]$ array:

$$X_{2n}[k] = \sum_{n=0}^{(N/2)-1} x[2n]W_{N/2}^{nk}, \tag{2}$$

where $0 \leqslant k \leqslant (N/2-1)$. The last two terms in Eq. (1) represent the $N/4$-point DFTs $X_{4n+1}[k]$ and $X_{4n+3}[k]$ with the twiddle factors:

$$\begin{aligned} &\sum_{n=0}^{(N/4)-1} x[4n+1]W_N^{(4n+1)k} \\ &\quad = W_N^k \sum_{n=0}^{(N/4)-1} x[4n+1]W_{N/4}^{nk} \\ &\quad = W_N^k X_{4n+1}[k] \end{aligned} \tag{3}$$

and

$$\begin{aligned} &\sum_{n=0}^{(N/4)-1} x[4n+3]W_N^{(4n+3)k} \\ &\quad = W_N^{3k} \sum_{n=0}^{(N/4)-1} x[4n+3]W_{N/4}^{nk} \\ &\quad = W_N^{3k} X_{4n+3}[k], \end{aligned} \tag{4}$$

where $0 \leqslant k \leqslant (N/4-1)$. Hence, the algorithm is a mixture of the radix-2 and radix-4 DIT-FFTs.

Fig. 1 illustrates a local butterfly-structured module performing an L-point DFT at the $(r-j)$th stage ($L = 2^{r-j}$). The L-point DFT $G[k]$ is decomposed into one $L/2$-point DFT $G_{2n}[k]$ at the $(r-j-1)$th stage and two $L/4$-point DFTs $G_{4n+1}[k]$ and $G_{4n+3}[k]$ at the $(r-j-2)$th stage. Note that $G_{2n}[k]$ is defined for $0 \leqslant k \leqslant (L/2-1)$, and $G_{4n+1}[k]$ and $G_{4n+3}[k]$ are defined for $0 \leqslant k \leqslant (L/4-1)$. Thus the L-point DFT can be derived using the periodic property of the DFT as below:

$$G[k] = G_{2n}[k] + (W_L^k G_{4n+1}[k] + W_L^{3k} G_{4n+3}[k]), \tag{5}$$

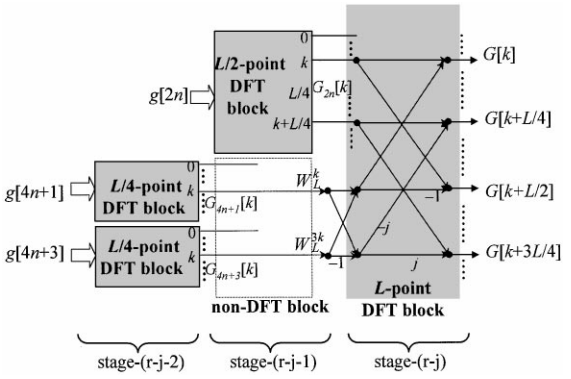

Fig. 1. A butterfly-structured module performing an L-point sr-FFT.

$$G[k+L/4] = G_{2n}[k+L/4] + W_L^{k+L/4} G_{4n+1}[k] + W_L^{3(k+L/4)} G_{4n+3}[k],$$
$$= G_{2n}[k+L/4] - j(W_L^k G_{4n+1}[k] - W_L^{3k} G_{4n+3}[k]), \quad (6)$$

$$G[k+L/2] = G_{2n}[k] + W_L^{k+L/2} G_{4n+1}[k] + W_L^{3(k+L/2)} G_{4n+3}[k],$$
$$= G_{2n}[k] - (W_L^k G_{4n+1}[k] + W_L^{3k} G_{4n+3}[k]), \quad (7)$$

$$G[k+3L/4] = G_{2n}[k+L/4] + W_L^{k+3L/4} G_{4n+1}[k] + W_L^{3(k+3L/4)} G_{4n+3}[k],$$
$$= G_{2n}[k+L/4] + j(W_L^k G_{4n+1}[k] - W_L^{3k} G_{4n+3}[k]), \quad (8)$$

where $0 \leqslant k \leqslant (L/4-1)$.

To better illustrate the structural property of the sr-FFT, Fig. 2 displays the complete butterfly structure for the 16-point DIT sr-FFT. Consider the four blocks at the second stage. The shaded regions indicate that the connecting branches in the regions are able to accomplish the local 2^2-point DFT. These regions will be called the 'DFT blocks' later. The blank region at the second stage, on the other hand, indicates the output values are not the correct DFT values. Therefore, the blank region represents the 'non-DFT block'. Further decomposition into two length-2 arrays is performed in the non-DFT block at the second stage.

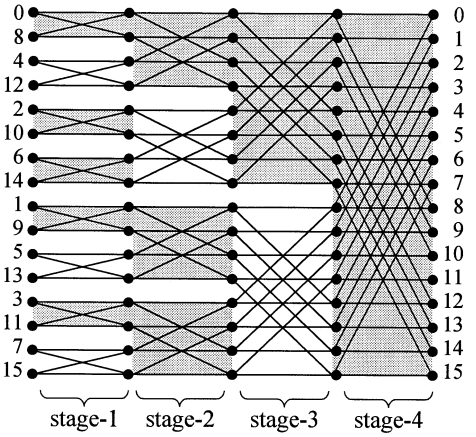

Fig. 2. The signal flow graph of a 16-point DIT sr-FFT structure.

3. Strategy of building the on-line butterfly sub-structures

The DFT basically transforms one data array into another data array of the same size. The conventional FFT, designed to obtain the complete frequency-band spectrum, hence makes an inherent assumption that the input and output sequences have the same array size. However, two situations often encountered are: (1) only a narrow frequency band of the Fourier spectrum is of interest, and (2) the input data array consists of many zero-valued data points due to zero-padding. To reduce the unnecessary arithmetic operations, a number of 'pruned FFT' algorithms were introduced [2,9,10, 12,13,16,18]. Pruning is a modification of the complete butterfly structure by removing those branches connected to the zero-valued input (input pruning) or to the unattended output samples (output pruning). The trimming of the L-shaped butterfly structure results in further reduction in the number of arithmetic operations.

The EEG is usually sampled at 200 Hz. To analyze and display the on-line, running Fourier spectra using 512-point DFT, one would spend more than 2.5 s on doing nothing except acquiring the digital EEG data. Inspired by the input pruning scheme, the authors developed the real-time FFT algorithm which builds the on-line butterfly sub-structures based on the DIT sr-FFT structure. The efficient algorithm presented in this paper utilizes the data

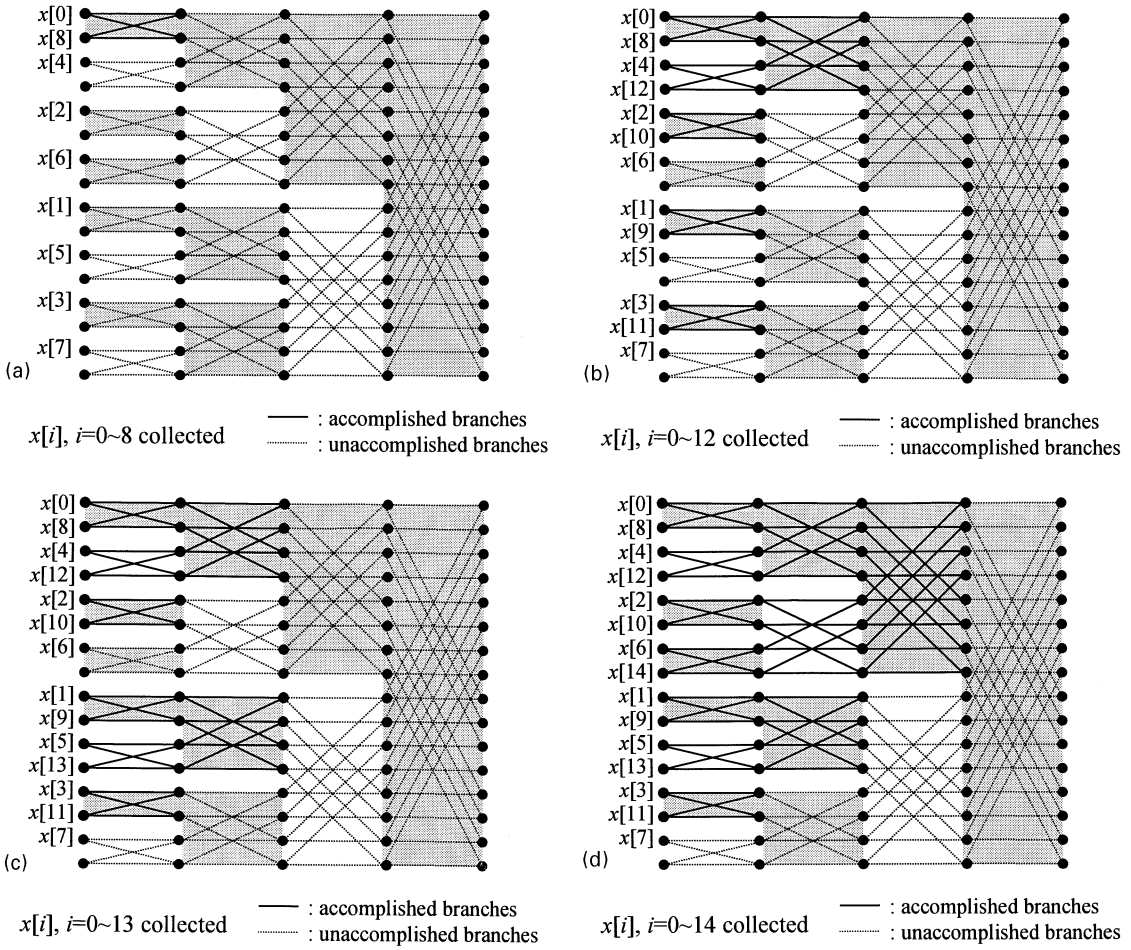

Fig. 3. The signal flow graphs showing the accomplished butterfly sub-structures upon receipt of: (a) the first 9 points, (b) the first 13 points, (c) the first 14 points, and (d) the first 15 points (assume the 16-point FFT).

points available to construct the local L-shaped butterfly modules while the EEG recording system is still acquiring the digital EEG data. In Section 5, we will show that, before the last data point comes in, the algorithm is able to accomplish more than one-half DIT sr-FFT structure when $r > 4$ (r: number of stages of the sr-FFT butterfly structure). Moreover, the accomplished portion increases with N (the FFT size, $N = 2^r$).

To describe how the algorithm works, the 16-point DIT sr-FFT structure in Fig. 2 is considered. The signal flow graphs in Fig. 3 are used to illustrate the progress of the 16-point sr-FFT butterfly structure given that the data points collected are $x[i]$, $i = 0$–8 (Fig. 3a), 0–12 (Fig. 3b), 0–13 (Fig. 3c), and 0–14 (Fig. 3d). The solid lines represent those computational branches already accomplished. The dashed lines indicate that the signal-flow branches are not formed because the last few data points are not received. Obviously, the first butterfly sub-structure is constructed when the $(N/2 + 1)$th point ($x[8]$ in Fig. 3a) is available. That is, the algorithm starts building the butterfly sub-structures only when more than one half data points are collected. Note that the last block at each stage cannot be accomplished until the last point $x[N - 1]$ is collected (Fig. 3d). Then one question arises: given that the nth $(N/2 + 1 \leqslant n \leqslant N - 1)$ position of the

input data array is filled, which butterfly substructures (modules) can be built? In the following, the authors propose a strategy of identifying these butterfly modules.

4. Identification of butterfly modules ready to be constructed

Consider an N-point DFT where $N = 2^r$. The indexes of input data array are expressed in binary form (r-bit): $x[0 \dots 0] \sim x[1 \dots 1]$. Let the first bit represent the least significant bit. There are r stages in the sr-FFT (reversed L-shaped) butterfly structure. According to the DIT FFT algorithm, the input sequence is arranged in the bit-reversed order to obtain a normal-ordered output sequence for in-place computation. Fig. 4 displays the status of the last four stages. The shaded regions represent the DFT blocks. The blank regions are the non-DFT blocks which require further decomposition into two half-length DFTs. At any given stage, a DFT block is constructed from the results of one DFT and one non-DFT blocks at the previous stage, whereas a non-DFT block is constructed from the results of two DFT blocks at the previous stage. Fig. 4 shows that, to complete at least one (DFT or non-DFT) block at the ith stage, one must have at least collected the input data point $x[n]$, where $n \geqslant 2^{r-1} + 2^{r-2} + \cdots + 2^{r-i}$. That is, the last (higher-significant) i bits of the array index must be '1'. For example, the top DFT block of the second stage of a 16-point sr-FFT can be completed when $x[1100]$ is collected. In addition, the authors

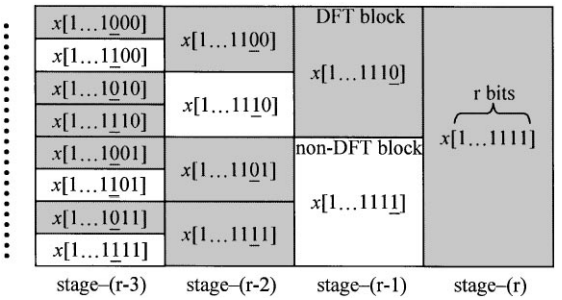

Fig. 4. The DFT (shaded) and non-DFT (blank) blocks partitioning the last four stages of an r-stage sr-FFT butterfly structure.

observed a straightforward regulation for the real-time FFT algorithm to identify the butterfly modules ready to be constructed. Consider the array index n_b (in binary format) at the $(r-j)$th stage. If the jth bit of n_b is '0', one cannot move to the $(r-j+1)$th stage. On the other hand, construction of the butterfly module can further proceed to the next higher stage, given that the jth bit is '1' at the $(r-j)$th stage. As indicated in Fig. 4, we can accomplish the top DFT block at the $(r-1)$th stage after the second block (non-DFT) at the $(r-2)$th stage is accomplished upon receipt of $x[1 \dots 11\underline{1}\,0]$. In other words, the appearance of $x[1 \dots 11\underline{1}\,0]$ enables the algorithm to construct the (2^{r-2})th block (from the top, refer to Fig. 4) of the first stage, the (2^{r-3})th block of the second stage, …, up to the (2^0)th block (the first block) of the $(r-1)$th stage. Consequently, the guideline directs the real-time FFT algorithm to the correct succeeding butterfly modules.

5. Evaluation of computational complexity

The computational complexity of the sr-FFT algorithm has been discussed in detail in [6,7,17]. In this paper, the authors develop an alternative way to analyze the number of complex multiplications and additions based on the number of DFT blocks at each stage.

At the first stage of the DIT sr-FFT butterfly structure, the algorithm requires $2 \cdot 2^{r-1} = N$ complex additions and no complex multiplication. To calculate the number of complex arithmetic operations performed at the $(r-j)$th stage $(0 \leqslant j \leqslant r-2)$, we examine the reversed L-shape module shown in Fig. 1. Note that, to accomplish the L-point DFT $(L \geqslant 4)$ at the $(r-j)$th stage, the structural schema consists of one $L/2$-point DFT block and one non-DFT block at the $(r-j-1)$th stage as well as two $L/4$-point DFT blocks at the $(r-j-2)$th stage, where $L = 2^{r-j} = N/2^j$. Suppose that $G_{2n}[k]$, $G_{4n+1}[k]$, and $G_{4n+3}[k]$ have been computed. According to Eq. (5)–(8), computing the L-point DFT $G[k]$ from $G_{2n}[k]$, $G_{4n+1}[k]$, and $G_{4n+3}[k]$ requires $2 \cdot (L/4) = 2^{r-j-1}$ complex multiplications (due to the twiddle factors in the non-DFT block) and $6 \cdot (L/4) = 3 \cdot 2^{r-j-1}$ complex additions.

Accordingly, there is no arithmetic operation to be counted in the non-DFT block. Thus, the total number of complex multiplications and additions can be obtained if we know the number of DFT blocks at each stage. There are 2^j blocks at the $(r-j)$th stage. Note that the non-DFT blocks at the $(r-j)$th stage are decomposed from the DFT blocks at the $(r-j+1)$th stage. Hence, the number of DFT blocks at the $(r-j)$th stage is

$$N_{r-j}(\mathrm{DFT}) = 2^j - N_{r-j+1}(\mathrm{DFT}),$$
$$j = 1, \ldots, r-1, \tag{9}$$

given that $N_r(\mathrm{DFT}) = 2^0 = 1$ for the last stage. It follows that

$$N_{r-j}(\mathrm{DFT}) = \sum_{l=0}^{j} (-1)^l 2^{j-1} = 2^j \cdot \left[\frac{2}{3} + \frac{1}{3}\left(-\frac{1}{2}\right)^j\right]. \tag{10}$$

Then, we obtain the total number of complex multiplications $N_{r-j}(\mathrm{MUL_c})$ and complex additions $N_{r-j}(\mathrm{ADD_c})$ at the $(r-j)$th stage:

$$N_{r-j}(\mathrm{MUL_c}) = 2^{r-j-1} \cdot N_{r-j}(\mathrm{DFT}) = 2^{r-1} \cdot \left[\frac{2}{3} + \frac{1}{3}\left(-\frac{1}{2}\right)^j\right],$$
$$0 \leqslant j \leqslant r-2, \tag{11}$$

$$N_{r-j}(\mathrm{ADD_c}) = (3 \cdot 2^{r-j-1}) \cdot N_{r-j}(\mathrm{DFT}) = (3 \cdot 2^{r-1}) \cdot \left[\frac{2}{3} + \frac{1}{3}\left(-\frac{1}{2}\right)^j\right],$$
$$0 \leqslant j \leqslant r-2. \tag{12}$$

Finally, the total number of complex multiplications $N_{\mathrm{total}}(\mathrm{MUL_c})$ is

$$N_{\mathrm{total}}(\mathrm{MUL_c}) = \sum_{j=0}^{r-2} N_{r-j}(\mathrm{MUL_c}) = 2^{r-1} \cdot \sum_{j=0}^{r-2} \left[\frac{2}{3} + \frac{1}{3}\left(-\frac{1}{2}\right)^j\right] = \frac{1}{3}(r-1)2^r + \frac{1}{9}2^r - \frac{2}{9}(-1)^{r-1} = \left(\frac{1}{3}r - \frac{2}{9}\right)N + \frac{2}{9}(-1)^r \tag{13}$$

and the total number of complex additions $N_{\mathrm{total}}(\mathrm{ADD_c})$ is

$$N_{\mathrm{total}}(\mathrm{ADD_c}) = N + \sum_{j=0}^{r-2} N_{r-j}(\mathrm{ADD_c}) = N + (3 \cdot 2^{r-1}) \cdot \sum_{j=0}^{r-2} \left[\frac{2}{3} + \frac{1}{3}\left(-\frac{1}{2}\right)^j\right] = \left(r + \frac{1}{3}\right) \cdot 2^r + \frac{2}{3}(-1)^r = \left(r + \frac{1}{3}\right)N + \frac{2}{3}(-1)^r. \tag{14}$$

Next, we calculate the number of complex multiplications and additions left to be completed upon receipt of the last point $x[2^r - 1]$.

Before $x[2^r - 1]$ is collected, only the last block of each stage is not constructed. Note that if r is an odd number, the last block of the first stage is a DFT block; otherwise, it is a non-DFT block. In addition, the last block is a DFT block every other stage. Let r be an odd number. Then, upon receipt of $x[2^r - 1]$, the algorithm starts building the last block (2-point DFT) of the first stage and proceeds to the last block of the third stage, the fifth stage, …, up to the rth stage. The number of complex multiplications involved in completing the last block of each stage is: 0 (the first stage), 2^2 (the third stage), 2^4 (the fifth stage), …, and 2^{r-1} (the rth stage). Thus, the total number of complex multiplications $N_{\mathrm{last}}(\mathrm{MUL_c})$ is:

$$N_{\mathrm{last}}(\mathrm{MUL_c}) = 0 + 2^2 + 2^4 + \cdots + 2^{r-1} = \frac{2N}{3} - \frac{4}{3},\ r\text{: odd-numbered.} \tag{15}$$

If r is an even number, the number of complex multiplications is: 2 (the second stage), 2^3 (the fourth stage), …, and 2^{r-1} (the rth stage). The total number becomes:

$$N_{\mathrm{last}}(\mathrm{MUL_c}) = 2 + 2^3 + 2^5 + \cdots + 2^{r-1} = \frac{2N}{3} - \frac{2}{3},\ r\text{: even-numbered.} \tag{16}$$

Similarly, the number of complex additions involved in completing the last block of each stage is: 2 (the first stage), $3\cdot 2^2$ (the third stage), $3\cdot 2^4$ (the fifth stage), …, and $3\cdot 2^{r-1}$ (the rth stage) for an odd-numbered r. Thus, the total number of complex additions $N_{\text{last}}(\text{ADD}_c)$ is

$$N_{\text{last}}(\text{ADD}_c) = 2 + 3\cdot 2^2 + 3\cdot 2^4 + \cdots + 3\cdot 2^{r-1} = 2N - 2,\ r\text{: odd-numbered.} \quad (17)$$

For an even-numbered r, the number of complex additions is: $3\cdot 2^1$ (the second stage), $3\cdot 2^3$ (the fourth stage), …, and $3\cdot 2^{r-1}$ (the rth stage). The total number becomes

$$N_{\text{last}}(\text{ADD}_c) = 3\cdot 2^1 + 3\cdot 2^3 + \cdots + 3\cdot 2^{r-1} = 2N - 2,\ r\text{: even-numbered.} \quad (18)$$

Eqs. (15)–(18), compared with Eqs. (13) and (14), show that both the ratios of $N_{\text{last}}(\text{MUL}_c)$ to $N_{\text{total}}(\text{MUL}_c)$ and of $N_{\text{last}}(\text{ADD}_c)$ to $N_{\text{total}}(\text{ADD}_c)$ are approximately in proportion to $2/r$. In comparison with the conventional FFT algorithms, the real-time FFT algorithm exhibits better efficiency especially for a large N (the FFT size) when applied to the on-line Fourier analysis.

The authors compared the computational time required by the real-time FFT algorithm and the algorithm provided by Matlab. The real-time FFT algorithm spent 10 s on computing 1000 times of the 1024-point FFT (on DX100-486 PC), while the Matlab spent approximately 12 s on the same task. Note that the computational time spent on building the rest butterfly sub-structures upon receipt of the last data point was approximately 2 s, that is, one-fifth of the total time required by the entire process. The result is consistent with our derivation, i.e., $N_{\text{last}}/N_{\text{total}} \approx 2/r$ (here, $r = 10$).

6. Conclusion

We have introduced a new technique of real-time implementation of the DIT sr-FFT scheme. The technique can be applied to any FFT scheme possessing modular structure with an orderly arrangement of sample points. According to both the derivation and experimental result, the number of arithmetic operations left to be accomplished after collecting all the input data is approximately $2/r$ of the total number of arithmetic operations required for computing an N-point FFT ($N = 2^r$). This fact further shows that the efficiency of the FFT algorithm can be highly improved by utilizing the spare time of CPU during the I/O process.

References

[1] M.M. Anguh, Quadtree and symmetry in FFT computation of digital images, IEEE Trans. Signal Process. 45 (12) (December. 1997) 2896–2899.
[2] S. Barash, Y. Ritov, Logarithmic pruning of FFT frequencies, IEEE Trans. Signal Process. 41 (3) (March 1993) 1398–1400.
[3] V. Boriakoff, FFT computation with systolic arrays, A new architecture, IEEE Trans. Circuit Systems – II: Analog Digital Signal Process. 41 (4) (April 1994) 278–284.
[4] J.W. Cooley, J.W. Tukey, An algorithm for machine computation of complex Fourier series, Math. Comput. 19 (1965) 297–301.
[5] R. Cooper, J.W. Osselton, J.C. Shaw, EEG Technol, 3rd ed., Ch. 6 Butterworth Inc, Woburn, MA, 1980.
[6] P. Duhamel, Implementation of 'split-radix' FFT algorithms for complex, real and real-symmetric data, IEEE Trans. Acoust. Speech Signal Process. ASSP-34, (April 1986) 285–295.
[7] P. Duhamel, H. Hollmann, Split radix FFT algorithm, Electron. Lett. 20 (1) (January 1984) 14–16.
[8] S. He, M. Torkelson, Computing partial DFT for comb spectrum evaluation, IEEE Signal Process. Lett. 3 (6) (June 1996), 173–175.
[9] J.D. Markel, FFT pruning, IEEE Trans. Audio Electroacoust. AU-19 (4) (December 1971) 305–311.
[10] K. Nagai, Pruning the decimation-in-time FFT algorithm with frequency shift, IEEE Trans. Acoust. Speech Signal Process. ASSP-34 (4) (August 1986) 1008–1010.
[11] A.V. Oppenheim, R.W. Schafer, Discrete-Time Signal Process. Ch. 9, Prentice-Hall, Englewood Cliffs, NJ, 1989.
[12] D.E. Paneras, R. Mani, S.H. Nawab, STFT computation using pruned FFT algorithms, IEEE Signal Process. Lett. 1 (4) (April 1994) 61–63.
[13] C. Roche, A split-radix partial input/output fast Fourier transform algorithm, IEEE Trans. Signal Process. 40 (5) (May 1992) 1273–1276.
[14] I.W. Selesnick, C.S. Burrus, Automatic generation of prime length FFT programs, IEEE Trans. Signal Process. 44 (1) (January 1996) 14–24.
[15] A.N. Skodras, Efficient computation of the split-radix FFT, IEE Proc. 139 (1) (February 1992) 56–60.
[16] H.V. Sorensen, C.S. Burrus, Efficient computation of the DFT with only a subset of input or output points, IEEE Trans. Signal Process. 41 (3) (March 1993) 1184–1200.

[17] H.V. Sorensen, M.T. Heideman, C.S. Burrus, On computing the split-radix FFT, IEEE Trans. Acoust. Speech Signal Process. ASSP-34 (1) (February 1986) 152–156.

[18] T.V. Sreenivas, P.V.S. Rao, High resolution narrow-band spectra by FFT pruning, IEEE Trans. Acoust. Speech Signal Process. ASSP-28 (2) (April 1980) 254–257.

[19] P.R. Uniyal, Transforming real-valued sequences: fast Fourier versus fast Hartley transform algorithms, IEEE Trans. Signal Process. 42 (11) (November 1994) 3249–3254.

[20] J. You, S.S. Wong, Serial-parallel FFT array processor, IEEE Trans. Signal Process. 41 (3) (March 1993) 1472–1476.