Digital Signal Processing 25 (2014) 275–279

Contents lists available at ScienceDirect

Digital Signal Processing

www.elsevier.com/locate/dsp

Recursive sliding discrete Fourier transform with oversampled data

A. van der Byl *, M.R. Inggs

Department of Electrical Engineering, University of Cape Town, Rondebosch 7701, South Africa

ARTICLE INFO

Article history:
Available online 17 October 2013

Keywords:
Discrete Fourier transform
Recursive discrete Fourier transform
Sliding discrete Fourier transform
Running Fourier transform
Spectral updating

ABSTRACT

The Discrete Fourier Transform (DFT) has played a fundamental role for signal analysis. A common application is, for example, an FFT to compute a spectral decomposition, in a block by block fashion. However, using a recursive, discrete, Fourier transform technique enables sample-by-sample updating, which, in turn, allows for the computation of a fine time–frequency resolution. An existing spectral output is updated in a sample-by-sample fashion using a combination of the Fourier time shift property and the difference between the most recent input sample and outgoing sample when using a window of finite length. To maintain sampling-to-processing synchronisation, a sampling constraint is enforced on the front–end hardware, as the processing latency per input sample will determine the maximum sampling rate. This work takes the recursive approach one step further, and enables the processing of multiple samples acquired through oversampling, to update the spectral output. This work shows that it is possible to compute a fine-grained spectral decomposition while increasing usable signal bandwidths through higher sampling rates. Results show that processing overhead increases sub-linearly, with signal bandwidth improvement factors of up to 6.7× when processing 8 samples per iteration.

1. Introduction

The Discrete Fourier Transform (DFT) has played a fundamental role for signal analysis. Traditionally, samples are acquired and processed in a block-processing fashion, where the number of samples required per spectral update is a function of the desired spectral resolution. The result is a delayed output with the time resolution determined by the block capture and processing rate.

To improve time and frequency resolution, we need to either capture and process the block data at a higher rate, or, alternatively, utilise recursive Fourier transform methods [1–4]. Adopting Recursive Sliding discrete Fourier Transform techniques enables sample-by-sample updating with the flexibility of allowing the computation of finer time–frequency resolution. The sample-by-sample updating uses the Fourier time shift property to update an existing spectral output using the most recent input sample. To maintain sampling-to-processing synchronisation, a sampling constraint is enforced on the front–end hardware. The processing latency per input sample will determine the maximum sampling rate permitted to allow an updated output in a single sample period [5].

This work takes this technique one step further, permitting multiple samples gained through sampling rates higher than those permitted for single-input synchronisation, however still achieving synchronous processing (albeit with a marginal penalty). This in turn shows that it is possible to increase the sampling rate with the aim to increase usable signal bandwidths, and still achieve a fine time–frequency decomposition. Section 2 discusses the Recursive DFT and highlights previous work in the field including error correction for finite-bit arithmetic. Section 3 discusses the use of multiple samples for spectral updating, and discusses the costs and benefits of using higher sampling rates.

2. Recursive sliding discrete Fourier transform

The method employed in this study for DFT computation is the Recursive DFT (RDFT), where the RDFT is based on the principle of updating a current output $F[u]$ as new data is added to the input sequence. The addition of a new data point does not imply that the input sequence has to grow in size (from size N to $N+p$, where p is one for a single new input to the sequence), but rather imposes the constraint that a window function of size N is required, which shifts to include the new sample $(N+p)$, and removes the outgoing sample $(N+p-N=p)$.

If the output is known a priori, an update can be computed by utilising the Fourier shift theorem and computing a DFT on the new data that has been added, while removing the influence the outgoing sample had on the output (for a window of size N). The computational cost for the recursive DFT technique is far lower than the FFT $(O(n\log_2 n))$ [1,6], and is shown to improve by $\log_2 n$ [2].

The recursive technique is based on the work of Sherlock [2] and Kamei [7] and computes an updated $F[u]$ by removing the

* Corresponding author.
E-mail addresses: a.vanderbyl@uct.ac.za (A. van der Byl), Michael.Inggs@uct.ac.za (M.R. Inggs).

1051-2004/$ – see front matter
http://dx.doi.org/10.1016/j.dsp.2013.10.008

contribution of the outgoing sample and adding the contribution of the incoming sample using N length window [1,3,8].

Let:

$$f_{out} = f[0] \quad \text{be the outgoing sample} \tag{1}$$

$$f_{in} = f[N] \quad \text{be the incoming sample} \tag{2}$$

The new output can be computed using:

$$F_{new}[u] = \left[F_{current}[u] + \frac{f_{in} - f_{out}}{N} \right] W_N^u \tag{3}$$

where the complex exponential is defined in a more compact form:

$$\exp\left[j\frac{2\pi u}{N} \right] = W_N^u \tag{4}$$

Using this formula, an updated output can be computed for each DFT point based on the difference between the incoming and outgoing samples. A further advantage of this technique is each DFT point can compute an update independently (select frequencies of interest can be computed if desired), and only minimal data (new sample) needs to be transferred to the processing elements used for each DFT point. An FFT could be performed prior to using the recursive DFT (if block data available), or given a data sequence $f[n]$, the output is already known at time $t = 0$ ($F[u] = 0$ $\forall u$). Prior to any data entering the system, it can be assumed that the resulting output $F[u]$ is zero, and therefore can be used as the initial state for the recursive DFT. At the point where $t = N$, the resulting output matches the DFT output $F[u]$ for window of length N. Further samples can now be added and the resulting updated output computed. If the recursive DFT were implemented sequentially, the cost would be in the order of $O(N^2)$, however if concurrency were exploited by using many smaller processing elements, the cost reduces to $O(N)$, for computing a DFT of length N, if N processing elements are used [9].

3. Multi-sample updating

The work discussed in Section 2 only considers a single sample input per iteration (real or complex) based on a sampling rate with period T_s (and defined by the minimal processing time required to compute an update based on a single new sample). Processing of data assumes that the period (T_s) between input samples matches the processing latency of the underlying system computing the update to ensure no loss of information.

It would be beneficial to explore the possibility to capture multiple data samples at a scaled sampling rate ($\frac{T_s}{k}$) within the processing time, and present an updated spectra based not on one sample, but rather on k samples, where k represents a sampling rate scaling factor ($k \in Z$). Eq. (3) expresses a single sample based update in terms of the Fourier time shift property and the current DFT output vector $F_{current}[u]$ for point u. If a higher sampling rate were used, instead of f_{out} and f_{in} representing a single sample, they would represent multiple samples acquired during the processing latency inherent in computing Eq. (3). The total sample pairs represented by f_{out} and f_{in} are stipulated by the value of k, and are processed in the same manner, except for the value of the complex exponential (which is determined as a function of the time shift).

Computing an update using two sets of samples ($k = 2$) involves shifting in two new samples (f_{in_1} and f_{in_2}), and shifting out two samples (f_{out_1} and f_{out_2}) (required to maintain a constant window length). The difference between the incoming and outgoing sample pairs requires multiplication by $W_N^{(s+1)u}$ where $s = 0, 1$ ($s \in Z$), respectively, followed by summation. s will take on two values in this example, as two samples are shifted in and out, and each difference pair requires multiplication by a different phase. Re-writing Eq. (3) for any k:

$$F_{new}[u] = \left[F_{current}[u] \right] W_N^{ku} + \sum_{s=0}^{k-1} \left[\frac{f_{in_{(k-s)}} - f_{out_{(k-s)}}}{N} \right] W_N^{(s+1)u} \tag{5}$$

Two key points should be noted from Eq. (5). Firstly, the existing computed spectrum is now multiplied by a complex exponential influenced by the shifting parameter k. Secondly, the incoming and outgoing samples are handled in pairs, and multiplied by a complex exponential determined relative to the shift the pair of samples experienced. It should also be noted that when computing this step, the inherent parallelism can be exploited minimising the additional overhead required if processing resources permit it. The computation in Eq. (5) is suitable for both finite-bit and floating-point arithmetic, however the use of error correction would be needed to maintain a constant error rate when implementing finite-bit arithmetic. The following section details this inclusion.

3.1. Error correction

The recursive DFT as expressed in Eq. (3) allows for frequent spectral updating when a single sample is added, however, computational errors can accumulate if implemented with finite-bit arithmetic. The error is produced as a result of a quantisation and arithmetic round-off in the complex exponentials used per point, as well as a round-off used in the storage and computation of the DFT update. Furthermore, the error grows without bound due to the recursive nature of the algorithm [3,10].

It is possible to model and track errors as they develop, allowing on-the-fly dynamic error correction per point [11,5]. The correction vector (E_u) for a single sample shift is expressed in Eq. (6) for iteration l at DFT point u [5]:

$$E_u[l+1] = \sigma_u F_{current}[l] + \sigma_u \left[\frac{f_{in} - f_{out}}{N} \right] + W_N^u E_u[l] \tag{6}$$

where:

$$\sigma_u = \hat{W}_N^u - W_N^u \tag{7}$$

and:

W_N^u is the complex twiddle factor and
$\hat{W}_N^u$ is the finite-bit approximation of W_N^u
$F_{current}$ is the current DFT point output
f_{out}, f_{in} are the outgoing and incoming samples

Modifying Eq. (6) to handle multiple input sample pairs produces:

$$E_u[l+1] = \sigma_{ku} F_{current}[l] + \sum_{s=0}^{k-1} \sigma_{(s+1)u} \left[\frac{f_{in_{(k-s)}} - f_{out_{(k-s)}}}{N} \right] + W_N^{ku} E_u[l] \tag{8}$$

where:

$$\sigma_{ku} = \hat{W}_N^{ku} - W_N^{ku} \tag{9}$$

and

$$\sigma_{(s+1)u} = \hat{W}_N^{(s+1)u} - W_N^{(s+1)u} \tag{10}$$