A Mathematical Framework for Parallel Computing of Discrete-Time Discrete-Frequency Transforms in Multi-Core Processors

Pablo Soto-Quiros

Escuela de Matemáticas, Instituto Tecnológico de Costa Rica, Apdo. 159-7050, Cartago, Costa Rica

Published online: 1 Nov. 2014

Abstract: This paper presents a mathematical framework for a family of discrete-time discrete-frequency transforms in terms of matrix signal algebra. The matrix signal algebra is a mathematics environment composed of a signal space, a finite dimensional linear operators and special matrices where algebraic methods are used to generate these signal transforms as computational estimators. The matrix signal algebra contribute to analysis, design and implementation of parallel algorithms in multi-core processors. In this work, an implementation and experimental investigation of the mathematical framework are performed using MATLAB® with the Parallel Computing Toolbox™. We found that there is advantage to use multi-core processors and a parallel computing environment to minimize the high execution time. Also, speedup and efficiency increases when the number of logical processor and length of the signal increase. Moreover, a superlinear speedup is obtained in this experimental investigation.

Keywords: DFT, matrix signal algebra, superlinear speedup

1 Introduction

In signal processing, an important aspect of the study of a signal is understanding how its frequency varies with time [1, 2]. The time-frequency analysis was developed to aid get this information using time-frequency representations of a signal, through of time-frequency transforms [2, 3]. Time-frequency transforms can represent a signals over a time-frequency plane. These transforms combine time-domain and frequency-domain analyses to yield a picture of the temporal localization of a signals spectral components. They may also serve for signal synthesis, coding and processing [1, 3].

A computational implementation of time-frequency transforms is performed using discrete periodic signals and discrete-time discrete-frequency (DT-DF) transforms. A signal is a discrete periodic signal if it completes a pattern within a measurable time frame, called a period and repeats that pattern over identical subsequent periods. Examples of DT-DF transforms are the discrete ambiguity function (DAF) [4], the discrete short-time Fourier transforms (DSTFT) [5], the discrete Zak transform (DZT) [6], the discrete chirp-Fourier transform (DCFT) [7], the modified discrete chirp-Fourier transform (MDCFT) [8] and the new discrete chirp-Fourier transform (NDCFT) [9]. These transforms have several applications in engineering: waveform designs [10], time-frequency representations of audio [5], Gabor expansions and Weyl-Heisenberg frames [11], radar signal processing as estimator of range and velocity parameters of the moving object [2, 12] and synthetic aperture radar (SAR) and inverse SAR imaging [8]. Many implementations of these DT-DF transforms have been studied and developed in [2, 6, 7, 8, 9, 12, 13], but very few developed a parallel computing (see, e.g., [13, 14]).

In this paper, we present a new general mathematical framework for all DT-DF transforms mentioned above (DAF, DSTFT, DZT, DCFT, MDCFT, NDCFT). This mathematical framework is different to others implementations because express each DT-DF transform in terms of a matrix signal algebra, which is a mathematics environment composed of a signal space, finite dimensional linear operators and special matrices, where algebraic methods are used to generate these signal transforms as computational estimators [12]. This matrix signal algebra contributes to analysis, design and

* Corresponding author e-mail: jusoto@itcr.ac.cr
implementation of parallel algorithms. Thus, an implementation and experimental investigation of this mathematical framework are performed using MATLAB® with the Parallel Computing Toolbox™ in a computer with multi-core processors.

The present paper is organized as follows. In Section 2, we define the matrix signal algebra and we explain some applications to parallel computing. In Section 3, we explain the different types of DT-DF transforms to use in this paper. Furthermore, we develop a mathematical framework of DT-DF transforms in terms of the matrix signal algebra. In Section 4, we explain an implementation and experimental investigation of this mathematical framework using parallel computing in multi-core processors with MATLAB®. Finally, in Section 5, we present some conclusions.

Throughout the paper, the following notation is used. Throughout the paper, the following notation is used. $\mathbb{Z}_N = \{0, 1, \ldots, N-1\}$ is the additive group $\mathbb{Z}$ of integers modulo $N$, $\mathbb{C}^{M\times N}$ is the matrix space of $M$ rows and $N$ columns with complex numbers entries and $\mathbb{C}^N = \mathbb{C}^{N\times 1}$. The rows and columns of $A \in \mathbb{C}^{M\times N}$ are indexed by elements of $\mathbb{Z}_M$ and $\mathbb{Z}_N$, respectively. $A(m,n), A(m,\cdot)$, $A(\cdot, n), A^T$ represent entry $(m,n)$, row $m$, column $n$, conjugate matrix and transpose matrix of $A$, respectively. $I_N \in \mathbb{C}^{N\times N}$ and $I_N \in \mathbb{C}^N$ are identity matrix and one vector, respectively.

2 Matrix Signal Algebra

We define the matrix signal algebra as a mathematics environment composed of a signal space, finite dimensional linear operators and special matrices where algebraic methods are used to generate algorithms in signal processing area.

Let $A,B \in \mathbb{C}^{M\times N}, C \in \mathbb{C}^{P\times Q}$ and $\{A_n\}_{n \in \mathbb{Z}_N}$ such that $A_n \in \mathbb{C}^{M\times P}$. Some spaces, operators and matrices associated to the matrix signal algebra are the following:

- The space of discrete periodics signals, $\mathbb{P}^2(\mathbb{Z}_N)$, is the set of $C$-valued signals on $\mathbb{Z}_N$. Moreover, $x \in \mathbb{P}^2(\mathbb{Z}_N)$ if and only if $x \in \mathbb{C}^N$ [15]. This space corresponds to signals with finite energy and $N$-periodic sequences, i.e., for each $k_1 \in \mathbb{Z}$, $x(k_1) = x(k_2)$, where $k_2 \in \mathbb{Z}_N$ and $k_1 \equiv k_2 \mod N$.

- The Hadamard product of $A$ and $B$ is defined as $A \odot B \in \mathbb{C}^{M\times N}$ such that

$$\left( A \odot B \right)(m,n) = A(m,n) \cdot B(m,n).$$

The Hadamard product is also known as pointwise or coordinatewise product.

- The Kronecker product of $A$ and $C$ is defined as $A \otimes C \in \mathbb{C}^{MP\times NQ}$ such that

$$A \otimes C = \begin{pmatrix} A(0,0)C & \cdots & A(0,N-1)C \\ \vdots & \ddots & \vdots \\ A(M-1,0)C & \cdots & A(M-1,N-1)C \end{pmatrix}.$$ It replaces every entry $(m,n)$ of $A$ by the matrix $A(m,n)C$. In the special case $A = I_N$, it is called parallel operation [16].

- Let $N = RS$. The stride permutation matrix is defined as $L_S^N \in \mathbb{C}^{N\times N}$ such that it permutes the elements of the input signal $x \in \mathbb{C}^N$ as $mr+n \rightarrow mR+n, m \in \mathbb{Z}_R$ and $n \in \mathbb{Z}_S$ [16, 17]. This matrix permutation governs the data flow required to parallelize a Kronecker product computation [16].

- The vec operator, $\text{vec} : \mathbb{C}^{M\times N} \rightarrow \mathbb{C}^{MN}$, transforms a matrix into a vector, by stacking all the columns of this matrix one underneath the other. On the other hand, the vec inverse operator, $\text{vec}^{-1} : \mathbb{C}^{MN} \rightarrow \mathbb{C}^{M\times N}$, transforms a vector of dimension $MN$ into a matrix of size $M \times N$. $\text{vec}^{-1}$ is related to the stride permutation matrix: $\text{vec}^{-1}(L_S^Nv) = (\text{vec}^{-1}(v))^T$, for $v \in \mathbb{C}^N$.

- The accumulation operator of matrices, $\bigcup : \prod_{n \in \mathbb{Z}_N} \mathbb{C}^{M\times P} \rightarrow \mathbb{C}^{MP\times P}$ with $M = \sum_{n \in \mathbb{Z}_N} M_n$, is defined as

$$\bigcup_{n \in \mathbb{Z}_N} A_n = \begin{pmatrix} A_0 \\ \vdots \\ A_{N-1} \end{pmatrix}.$$ The following examples illustrate how the matrix signal algebra contributes to analysis, design and implementation of parallel algorithms.

Example 2.1. Let $A \in \mathbb{C}^{R\times M}, x \in \mathbb{C}^{RN}$ and $y \in \mathbb{C}^{MN}$. We consider the matrix operation $x \odot \left( I_N \otimes A \right) y$. This matrix operation can be decomposed as follows:

$$\begin{pmatrix} x_0 \\ \vdots \\ x_{N-1} \end{pmatrix} \odot \begin{pmatrix} A \\ \vdots \\ A \end{pmatrix} \begin{pmatrix} y_0 \\ \vdots \\ y_{N-1} \end{pmatrix} = \begin{pmatrix} x_0 \odot Ay_0 \\ \vdots \\ x_{N-1} \odot Ay_{N-1} \end{pmatrix},$$

where $x_m \in \mathbb{C}^R$ and $y_m \in \mathbb{C}^M$. The matrix operation $x \odot \left( I_N \otimes A \right) y$ can be divided into $N$ sub-operations $x_m \odot Ay_m$, for $m \in \mathbb{Z}_N$. The structure operation of $x \odot \left( I_N \otimes A \right) y$ allows an implementation using parallel computing, because each $x_m \odot Ay_m$ is computed independently.

Example 2.2. Matrix signal algebra is using to compute the discrete Fourier Transform (DFT) [16, 18, 19]. The DFT of $x \in \mathbb{P}^2(\mathbb{Z}_N)$ is represented as $F_x : \mathbb{Z}_N \rightarrow \mathbb{C}$ such that $F_x(k) = \frac{1}{\sqrt{N}} \sum_{n \in \mathbb{Z}_N} x(n) w_n^k$, where $w_n = e^{2\pi ink \ mod N}$. The matrix representation of DFT of $x$ is $F_x = \frac{1}{\sqrt{N}} F_N x$, where $F_N \in \mathbb{C}^{N\times N}$ such that $F_N(m,n) = w_m^n$. If $N = RS$, then the matrix formalism can be used to express $F_N$ as factorizations of matrices using operators and matrices from matrix signal algebra [16, 18]:

$$F_N = \frac{1}{\sqrt{N}} I_S^R \left( I_R \otimes F_S \right) L_S^N T_S^R \left( I_S \otimes F_R \right) L_S^N.$$ Here, $T_S^R$ is a diagonal matrix containing the twiddle factors. This factorization of $F_N$ is the recursive general-radix decimation in time Cooley-Tukey FFT for $N = RS$. In addition, this representation of $F_N$ allows the implementation using parallel computing [17].
explained above, or matrix form. Both representations of DT-DF transform allow to develop a fast algorithm, but the matrix representation permits an implementation using parallel computing.

Let \( T_x \in \mathbb{C}^{N \times N} \) the matrix representation of DT-DF transforms, such that \( T_x(m,k) = \mathcal{T}_x(m,k) \). The following result represents \( T_x \) in terms of matrix signal algebra.

**Theorem 3.2.1.** Let \( x \in \mathcal{F}(\mathbb{Z}_N) \). Then

\[
T_x = \frac{1}{\sqrt{N}} A \otimes \mathcal{R}_{N} \left\{ L_N^2 \left( I_N \otimes F_N \left( h \otimes (I_N \otimes x) \right) \right) \right\},
\]

where \( h \in \mathbb{C}^{N^2} \) such that \( h = \bigcup_{m \in \mathbb{Z}_N} [H(m,:)\,]^T \).

**Proof.** Let \( z = L_N^2 \left( I_N \otimes F_N \left( h \otimes (I_N \otimes x) \right) \right) \). This vector can be expressed as

\[
z = L_N^2 \bigcup_{m \in \mathbb{Z}_N} s_m,
\]

where \( s_m \in \mathbb{C}^N \), such that \( s_m = F_N \left( [H(m,:)\,]^T \otimes x \right) \).

Applying the \( \mathcal{R}_{N} \) operator in (3), we obtain

\[
\mathcal{R}_{N} \{z\} = \mathcal{R}_{N} \left\{ L_N^2 \bigcup_{m \in \mathbb{Z}_N} s_m \right\} = \left( \mathcal{R}_{N} \left\{ \bigcup_{m \in \mathbb{Z}_N} s_m \right\} \right)^T.
\]

Let \( S \in \mathbb{C}^{N \times N} \) such that \( S = \mathcal{R}_{N} \left\{ \bigcup_{m \in \mathbb{Z}_N} s_m \right\} \). Then

\[
\mathcal{R}_{N} \{z\}(m,k) = S^T(m,k) = s_m(k) = \sum_{m \in \mathbb{Z}_N} x(m)H(m,k)\omega_N^{-mk}.
\]

Finally, if we make the Hadamard product of \( A \) and \( S^T \), then we obtain

\[
\frac{1}{\sqrt{N}} (A \otimes S^T)(m,k) = \frac{1}{\sqrt{N}} A(m,k) \cdot S^T(m,k) = \frac{1}{\sqrt{N}} A(m,k) \cdot s_m(k) = T_x(m,k)
\]
Fig. 1 shows a model of DT-DF transforms using the matrix signal algebra. We can observe $N$ independent processes, making this approach a parallel operation. Now, using the property $\mathcal{R}_{N,N} \{ I_N^2 \mathbf{v} \} = (\mathcal{R}_{N,N} \{ \mathbf{v} \})^T$, equation (2) can write as

$$T_x = \frac{1}{\sqrt{N}} A \odot \mathcal{R}_{N,N} \{ (I_N \odot F_N) \ (h \odot (I_N \odot x)) \}^T. \quad (4)$$

The computational complexity of $\mathcal{R}_{N,N} \{ I_N^2 \mathbf{v} \}$ and $(\mathcal{R}_{N,N} \{ \mathbf{v} \})^T$ can be implemented linearly; thus, the equations (2) and (4) are computationally similar.

The following algorithm shows the implementation of equation (4).

**Algorithm 1: DT-DF Transform Algorithm**

**Require:** $x \in \mathbb{C}^N$

**Ensure:** $T_x \in \mathbb{C}^{N \times N}$

1. **for** $m \leftarrow 0 : N - 1$
2. $h \leftarrow [H(m, :)][T]
3. $v_1 \leftarrow x \odot h$
4. $v_2 \leftarrow \frac{1}{\sqrt{N}} F_N v_1$
5. $T_x(:, m) \leftarrow [A(m, :)][T] \odot v_2$
6. **end for**
7. $T_x \leftarrow (T_x)^T$

Steps 2-5 are independent in each iteration, therefore the above algorithm allows parallel computation. Also, in the case $A(m, n) = 1$, for all $m, n \in \mathbb{Z}_N$, the Hadamard product of Step 5 can be omitted.

### 4 Implementation and Experimental Investigation

**4.1 General Information**

The investigations have been carried out on multi-core processors computer of Instituto Tecnológico de Costa Rica (Costa Rica Institute Technology). The computer consists of 4 two-processor units (8 logical processors) with Intel® Core™ i7-3632QM CPU processor, system clock of 2.20 GHz and 8 GB of RAM.

In this experiment, we do the implementation and testing of Algorithm 1 for all DT-DF transforms defined above. We use a chirp signal $x \in I^2(\mathbb{Z}_N)$ such that $x(n) = \omega^{-25n^2 - 30n} + \omega^{-5n^2 - 63n}$ as experimental signal. We select a chirp signal because the time-frequency plane is a natural representation space for chirps signals and, therefore it is a signal frequently used in DT-DF transforms [21]. For the DAF, we use the same chirp signal $x$ as echo signal and, for the DSTFT, we use a discrete Hamming signal $w \in I^2(\mathbb{Z}_N)$ as the discrete window function, where it is defined as $w(n) = 0.54 - 0.46 \cos(2\pi n/(N - 1))$.

The implementation of Algorithm 1 to compute each DT-DF transform is performed using MATLAB®. MATLAB® provides two main ways to take advantage of multicore and multiprocessor computers: built-in multithreading and parallelism using MATLAB® workers.

---

1. If discrete echo signal is the same signal $x$, then DAF is called discrete cross-ambiguity function [2].
We use parallelism using MATLAB® workers. We can run multiple MATLAB® workers (MATLAB® computational engines) on a multi-core computer to execute applications in parallel, with the Parallel Computing Toolbox™. This approach allows more control over the parallelism than with built-in multithreading [22]. With programming constructs such as parallel for-loops (parfor) and batch, we write the parallel MATLAB programs of the mathematical framework for DT-DF transforms.

4.2 Results and Discussion

The computational performance analysis of Algorithm 1 is evaluated using the metrics speedup (or acceleration) and efficiency. Let $T_1$ the execution time of the sequential algorithm and $T_p$ the execution time of the parallel algorithm, where $p$ is the number of logical processors. The speedup is the ratio between the execution times of sequential and parallel implementations, and it is a value typically between 1 and $p$. It is represented by the formula $S = T_1 / T_p$. The efficiency is determined by the ratio between the speedup and the number of processing elements, and it is a value typically between 0 and 1. It is represented by the formula $E = T_1 / (p T_p)$. When $S > p$ and $E > 1$, it is called superlinear speedup.

Fig. 2 shows the execution time $T_s$, in seconds $s$, of Algorithm 1 as a function of $N$, where $N$ is the size of signal of each DT-DF transform. In this figure, it is observed that there is significant reduction in the parallel execution time of each DT-DF transform. For example, to compute DAF with a chirp signal of size $N = 8192$ produce a time of serial execution $T_1 = 178.845$ s. But, using parallel computing, we obtain $T_2 = 82.339$ s (43.04% of $T_1$), $T_4 = 28.310$ s (15.82% of $T_1$) and $T_8 = 11.700$ s (6.54% of $T_1$). This shows the advantage of using multi-core processors and a parallel computing environment to minimize the high execution time in each DT-DF transform. This is due because parallel computing is a form of computation in which many calculations are made simultaneously and sequentially.

Table 2: Speedup of Algorithm 1

<table>
<thead>
<tr>
<th>DT-DF Transform</th>
<th>$p$</th>
<th>$N = 256$</th>
<th>$N = 512$</th>
<th>$N = 1024$</th>
<th>$N = 2048$</th>
<th>$N = 4096$</th>
<th>$N = 8192$</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAF</td>
<td>2</td>
<td>2.267</td>
<td>1.255</td>
<td>2.878</td>
<td>3.751</td>
<td>2.172</td>
<td>2.141</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1.921</td>
<td>1.206</td>
<td>2.592</td>
<td>5.335</td>
<td>6.318</td>
<td>5.661</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>2.593</td>
<td>1.660</td>
<td>4.291</td>
<td>11.146</td>
<td>15.140</td>
<td>15.288</td>
</tr>
<tr>
<td>DSTFT</td>
<td>2</td>
<td>0.975</td>
<td>2.109</td>
<td>3.543</td>
<td>2.177</td>
<td>1.935</td>
<td>2.186</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.921</td>
<td>2.191</td>
<td>2.685</td>
<td>6.457</td>
<td>4.844</td>
<td>6.854</td>
</tr>
<tr>
<td>DZT</td>
<td>2</td>
<td>2.443</td>
<td>1.445</td>
<td>2.213</td>
<td>2.402</td>
<td>2.047</td>
<td>2.885</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2.035</td>
<td>1.257</td>
<td>2.644</td>
<td>4.571</td>
<td>4.497</td>
<td>7.318</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>2.670</td>
<td>1.779</td>
<td>4.701</td>
<td>10.555</td>
<td>11.269</td>
<td>12.465</td>
</tr>
<tr>
<td>DCFT</td>
<td>2</td>
<td>1.670</td>
<td>1.692</td>
<td>1.780</td>
<td>2.072</td>
<td>1.617</td>
<td>1.993</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2.102</td>
<td>2.895</td>
<td>3.123</td>
<td>2.805</td>
<td>2.584</td>
<td>2.820</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>1.867</td>
<td>4.432</td>
<td>8.263</td>
<td>7.761</td>
<td>7.599</td>
<td>8.355</td>
</tr>
<tr>
<td>MDCFT</td>
<td>2</td>
<td>2.009</td>
<td>1.680</td>
<td>1.895</td>
<td>1.653</td>
<td>2.238</td>
<td>2.315</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1.996</td>
<td>2.181</td>
<td>3.266</td>
<td>3.017</td>
<td>3.789</td>
<td>3.251</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>2.853</td>
<td>3.967</td>
<td>8.173</td>
<td>6.178</td>
<td>7.120</td>
<td>8.764</td>
</tr>
<tr>
<td>NDCFT</td>
<td>2</td>
<td>2.325</td>
<td>1.678</td>
<td>1.581</td>
<td>1.968</td>
<td>2.306</td>
<td>2.408</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1.998</td>
<td>3.200</td>
<td>2.638</td>
<td>2.811</td>
<td>2.673</td>
<td>3.080</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>2.899</td>
<td>3.649</td>
<td>6.480</td>
<td>8.017</td>
<td>7.810</td>
<td>8.136</td>
</tr>
</tbody>
</table>

$N$ is length of the signal and $p$ is the number of logical processor.

Tables 2 and 3 represent speedup and efficiency of Algorithm 1 obtained from the experimental chirp signal with each DT-DF transform. In Table 2, it is observed that the acceleration of most DT-DF transforms increases when $p$ increases, regardless of the value of $N$. Moreover, we obtain superlinear speedup in about 42% of simulations and most of it is obtained when $N$ increases. It indicate that speedup increases and superlinear speedup is obtained when $p$ and $N$ increase, using Algorithm 1 and MATLAB® with the Parallel Computing Toolbox™ in a computer with similar characteristic to those used in this paper. Superlinear speedup is not common in parallel computing. A few researches obtain a superlinear speedup in its parallel implementation (see, e.g., [27,28]). Some research mentioned various reasons for superlinear speedup: cache effect resulting from the different memory hierarchies of a modern compute [27], the termination time can be reduced when several searches are executed at the same time or the efficient utilization of resources by multiprocessors [29].

Now, Table 3 shows increasing values of efficiency with the increase of $p$ of most DT-DF transforms. For all DT-DF transforms, we obtain an efficiency above of 23% in the range $256 \leq N \leq 1024$ and an efficiency above of
57% in the range $2048 \leq N \leq 8192$. In special case $N = 8192$, we obtain an efficiency above 70%. Furthermore, we obtain an efficiency above 100% of 42% of simulations and an efficiency above 80% in 60% of simulations. It indicates a good efficiency to calculate DT-DF transforms using Algorithm 1 and MATLAB® with the Parallel Computing Toolbox™ in a computer with multiple-core processors:

- there is advantage to use multi-core processors and a parallel computing environment to minimize the high execution time (for DAF, we obtain $T_1 = 178.845$ s, $T_2 = 82.339$ s, $T_4 = 28.310$ s and $T_6 = 11.700$ s),
- speedup increases and superlinear speedup is obtained when the number of logical processors $p$ and length of the signal $N$ increase (42% of simulations),
- a good efficiency too is obtained when $p$ and $N$ increase (above 80% in 60% of simulations).

### Acknowledgement

The author wish to thank Vicerrectoría de Investigación y Extensión of Instituto Tecnológico de Costa Rica, Prof. Roger Moya (Instituto Tecnológico de Costa Rica) for useful suggestions of improving the presentation of the paper and Prof. Domingo Rodríguez (University of Puerto Rico, Mayagüez Campus) for insightful discussions and useful suggestions of improving the presentation of the paper.

### References


---

Pablo Soto-Quiros

works in Department of Mathematics at Instituto Tecnológico de Costa Rica since 2009 and University of Costa Rica since 2012. He received a Bachelor degree in Computer Assisted Math Teaching from Instituto Tecnológico de Costa Rica in 2007 and a M.Sc. in Applied Mathematics from University of Puerto Rico at Mayaguez, Puerto Rico, USA, in 2012. His main research interests are applied and numerical harmonic analysis, signal processing, linear programming, numerical linear algebra and numerical analysis.