Spectrum Sample Calculation of Discrete, Aperiodic and Finite Signals Using the Discrete Time Fourier Transform (DTFT)

Julio Cesar Taboada-Echave(✉), Modesto Medina-Melendrez, and Leopoldo Noel Gaxiola-Sánchez

Division of Research and Postgraduate Studies, Tecnológico Nacional de México, Instituto Tecnológico de Culiacán, Culiacán, SIN, Mexico
{m17170010, modestogmm}@itculiacan.edu.mx, drgaxiolasanchez@gmail.com

Abstract. A method for the calculation of spectrum samples of discrete, aperiodic and finite signals based on the DTFT is proposed. This method is based on a flexible discretization of the frequency variable that could produce equidistant, sparse or unique spectrum samples. It is implemented in a GPU platform as a Matrix-Vector product, being able to be applied on modern HPC systems. As a result, a general use tool is developed for the frequency analysis that achieves execution times in a linear relation with the length of the vector to be processed and the number of samples required. Finally, it is shown that the required execution time for the computation of equally spaced spectrum samples is competitive to the achievements of other tools for frequency analysis based on sequential execution.

Keywords: High performance computing · DTFT · Fourier analysis · GPU

1 Introduction

In digital signal processing, the transformation and analysis of information is fundamental. An alternative to this is the harmonic analysis also called Fourier analysis, which can be performed with a set of tools that enable a different interpretation of information by transforming it from time or space domain to frequency domain. Frequency domain analysis is based on the premise that all signals can be represented by the sum of different harmonic components, made up of sinuses and cosines. Tools for Fourier analysis include the Fourier Series (CTFS) for analyzing continuous and periodic signals; the Fourier Transform (CTFT) for continuous and aperiodic signals; the Discrete Time Fourier Series (DTFS) for discrete and periodic signals; and the Discrete Time Fourier Transform (DTFT) for discrete and aperiodic signals [1]. The inconvenience of DTFT for the analysis of discrete and aperiodic signals is that the resulting spectrum is a continuous function. The Fourier Discrete Transform (DFT), whose definition is derived from the DTFS or the DTFT according to several authors [2, 3], represents the first choice tool for the analysis of discrete, aperiodic and finite signals. The Fast Fourier Transform (FFT) [4], besides the direct version of the DFT, is currently the most widely used algorithm for spectral analysis. The use of the DFT has as a restriction that the spectrum samples that are obtained correspond to equidistant elements forming a vector of equal length as the input sequence. To avoid such restriction in the use of DFT, some authors propose pruning operations, who allows removing samples not required [5–8], or filling with zeros (zero-padding), that allows to increase the number of samples in the frequency domain. These operations allow the application of techniques for the recovery of windows in the frequency or zoom operations. For applications where the detection of single sample or sparse samples is required, Goertzel's algorithm represents a valid option [9]. Even though the previous alternatives may overcome different requirements for frequency analysis, none of these represents a versatile alternative for general use because all are based or are variants of the FFT algorithm. This publication proposes a versatile method for frequency analysis of discrete, aperiodic and finite signals, this is based on the computation of specific DTFT samples and is implemented on a parallel processing unit. The direct implementation of the proposed method has an order of $O(N^2)$ to process a single dimension signal or an order of $O(N^3)$ to process a two dimension signal. This order of complexity makes the implementation prohibitive, unless a high performance platform is chosen for parallel processing. At current time, Graphic Processing Units (GPU) had been used successfully as parallel processing units, thus implementations of the direct version of the DFT has been done based on GPU [6, 10] with good results, especially when the sequence to be analyzed has a prime length size [11]. The proposed method is implemented in a parallel structure based on GPU, where each of the processing elements individually calculates a sample of the spectrum required by the user.

2 Calculation of the DTFT for Discrete, Aperiodic and Finite Signals

This section shows the basis for the reinterpretation of the DTFT as the right tool for the analysis of discrete, aperiodic and finite signals instead of the DFT.

A. Adaptation of DTFT.

The DTFT is positioned as the most suitable tool when Fourier analysis of discrete, aperiodic and finite signals is required, and its definitions is given by

$$X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n} \tag{1}$$

The disadvantage of the DTFT is that, despite processing a discrete and aperiodic input, the frequency variable ω is in the periodic continuous domain with a period of 2π. To adapt its implementation in modern computing systems, it is possible to discretize the continuous variable. In the case that the selected samples of the continuous variable are equidistant and of an equal number than the length of the input sequence, their implementation results equivalent to the DFT. It is important to note that each discrete element of the continuous variable corresponds to a unique sample of the spectrum. With these conditions in mind, a not necessarily invertible implementation can be obtained to compute versatile DTFT samples.

For the definition of the DTFT sampling is necessary to introduce a selection vector $\omega[k]$ in substitution of the continuous variable ω, thus, the synthesis equation can be rewritten as

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\omega[k]n} \quad 0 \leq k \leq K-1 \tag{2}$$

where $\omega[k]$ represents the vector with the K required discretized frequencies and k represents its index. The vector $x[n]$ of length N represents the input sequence of a discrete and aperiodic signal, turning the equation into a finite summation.

B. Calculation of the DTFT as a matrix-vector product.

For the implementation of Eq. (2), it is convenient to arrange it in the form of a Matrix-Vector product, as observed in Eq. (3). In this way, a direct implementation in a parallel system is possible. The matrix consists of evaluations of the kernel transform, where each k row is formed by evaluations of the kernel for the frequency defined by $\omega[k]$.

$$X[k] = \begin{bmatrix} X[0] \\ X[1] \\ \vdots \\ X[K-1] \end{bmatrix} = \begin{bmatrix} 1 & e^{-j\omega[0]} & \cdots & e^{-j\omega[0](N-1)} \\ 1 & e^{-j\omega[1]} & \cdots & e^{-j\omega[1](N-1)} \\ \vdots & \vdots & \cdots & \vdots \\ 1 & e^{-j\omega[k-1]} & \cdots & e^{-j\omega[k-1](N-1)} \end{bmatrix} \begin{bmatrix} X[0] \\ X[1] \\ \vdots \\ X[N-1] \end{bmatrix} \tag{3}$$

The implementation in a parallel system of Eq. (3) can divide the task of computing a sample of the spectrum by each processing unit, individually. The computation of the sample within the thread is done by an iterative process of adding multiplication between the kernel transformed and the input sequence. The execution time of each execution thread is directly related to the length N of the input sequence. As result, $2N$ complex operations are performed on each thread.

A convenient way for its implementation in high performance computing equipment (HPC) is using GPU architectures, especially considering the multi-core characteristics of modern GPUs. A valid approach is to calculate in each core of the GPU one of the output samples, where each thread makes an iterative pass over all the input elements multiplying them with the kernels and carrying out their addition. For the calculation of the transform is necessary to determine the kernel values, which are calculated individually within each thread using the mathematical functions library of the GPU.

3 GPU Implementation

In order to perform the implementation in a parallel system, the Nvidia GPU platform was chosen. Currently, multiple HPC systems rely their computing power in GPU units [12], for instance, supercomputers based on Tesla units [13]. One of the advantages to choose a GPU as a hardware accelerator platform is that GPUs are available even in consumer grade computers, especially in high-performance computers. The GPU devices that dominate the consumer market are Nvidia with its GeForce series and ATI/AMD with its Radeon series [14]. The implementation of this proposal is made in a GPU of the Nvidia GeForce series with the set of CUDA libraries. It is important to consider that the final implementation can be developed indistinctly in CUDA C, or with the set of open-source libraries OpenCL for its operation in GPU of the families that support it like AMD/ATI or Intel. Based on the studies reported con the papers of Fang [15] and Karimi [16] where they compare the performance of CUDA C against OpenCL, CUDA C has been chosen.

A. Equipment description

A personal computer based on a Windows 10 Operating System, with a 1st generation Intel I7 950 processor, 24 GB RAM DDR3 was used for the test. Further, a Nvidia GTX 780-DC2-3GD5 GPU board was included as the parallel processing unit.

B. Control Algorithm

The system execution control is ruled by a sequential routine in the CPU that determines the number of processing units to be called in the GPU, as well as the GPU preprocessing routine itself. The main control routine described in Fig. 1 is intended to prepare the data for execution in the GPU system. In the case that complex data are processed, the x[n] input array could be separated in its real and imaginary part, xr[n] and xi[n] sequences respectively. In addition, the vector containing the required frequency samples, stored in the vector ω[k], must be passed as a parameter. The number

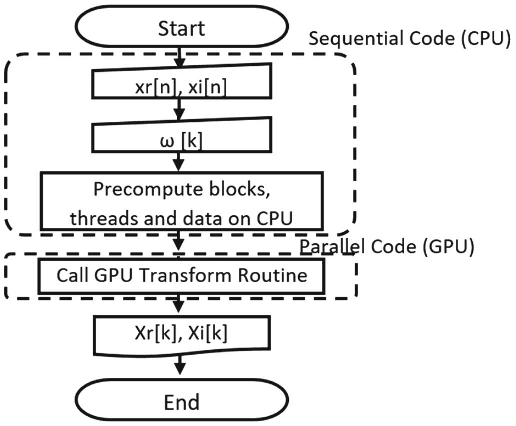

Fig. 1. Flowchart of the system.

of processing elements to be invoked in the GPU is calculated from the number of elements in ω[k].

In the GPU transformation routine, both the required elements of the kernel transform and the DTFT calculation are achieved.

For performance optimization, the closest memory resources in the GPU are used. The input sequence is first passed from the global memory to the shared memory. Data that is kept constant or unique in each thread is sent to the local memory of each thread. Once the data is processed and the results are obtained, they are stored on GPU's global memory and later copied to the CPU memory for their use.

C. Parallel Algorithm

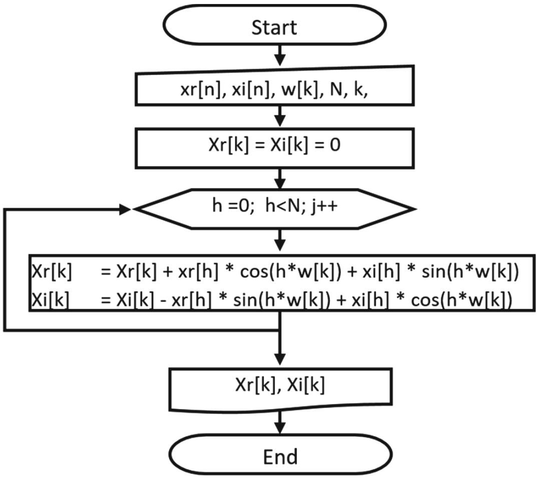
Start
xr[n], xi[n], w[k], N, k,
Xr[k] = Xi[k] = 0
h =0; h<N; j++
Xr[k] = Xr[k] + xr[h] * cos(h*w[k]) + xi[h] * sin(h*w[k])
Xi[k] = Xi[k] - xr[h] * sin(h*w[k]) + xi[h] * cos(h*w[k])
Xr[k], Xi[k]
End

Fig. 2. Flowchart of the GPU thread.

For the calculation of the DTFT in the parallel processing unit based on GPU, K processing threads are invoked where k represents the unique identificator (ID) of each CUDA thread. These thread IDs are directly related to the index of the samples to be calculated. The processing function receives as inputs the real and imaginary part of the input sequence (xr[n], xi[n]), the vector ω[k], the number of elements N in the input sequence and the number of elements K in the discretized frequency vector. These elements are illustrated in Fig. 2. It is important to mention that, for performance reasons, the DTFT kernels are calculated in the GPU. In addition, both the input sequence and the discretized frequency vector are previously stored in the shared memory and in the local memory of the thread for the case of being required more than once.

4 Validation and Performance Analysis

In order to validate the tool, a set of tests was done. The main goal is to validate the proposed characteristic of versatility. To determine the accuracy of the tool, a comparison between the obtained samples and those of the FFT has been done.

A. Validation of versatility

For the validation of the developed proposal's versatility, tests were carryout processing a synthetic signal composed of N = 64 input elements. For this signal were calculated equally spaced samples of the whole period with K = N (Fig. 3), with K = 2 N (Fig. 4) and with K = 2/N (Fig. 5). Equally samples on a spectrum's window from 3/4π to 5/4π with K = 32 were calculated and showed on Fig. 6. Non equally spaced samples from 0 to π were calculated and showed on Fig. 7. And finally, specific arbitrary samples were calculated and showed on Fig. 8. All the scenarios were compared against the continuous DTFT witch was estimated with 12,000 samples of the spectrum.

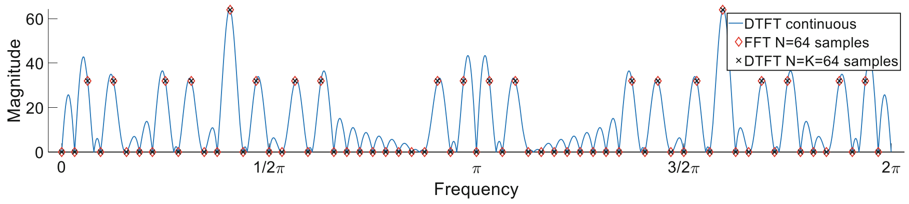

Fig. 3. Equidistant samples of the DTFT with *K* = *N* (equivalent to a DFT).

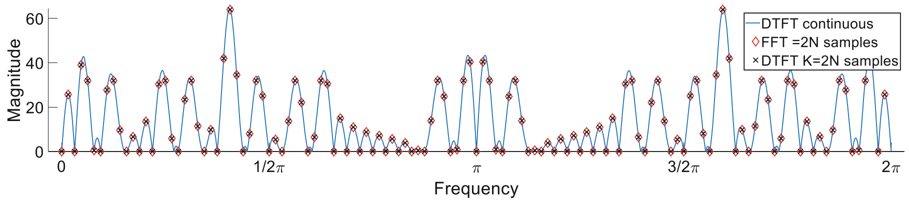

Fig. 4. Equidistant samples of the DTFT with *K* = *2 N* (equivalent to a oversampled DFT).

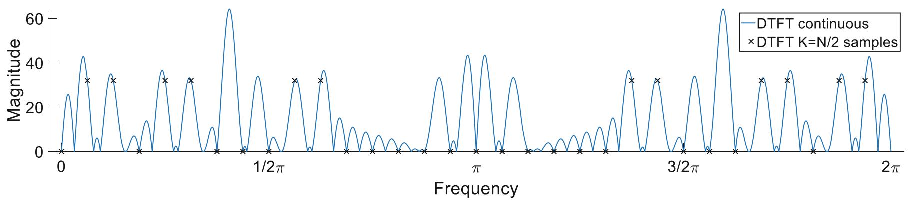

Fig. 5. Equidistant samples of the DTFT with *K* = *N/2* (equivalent to a subsampled DFT).

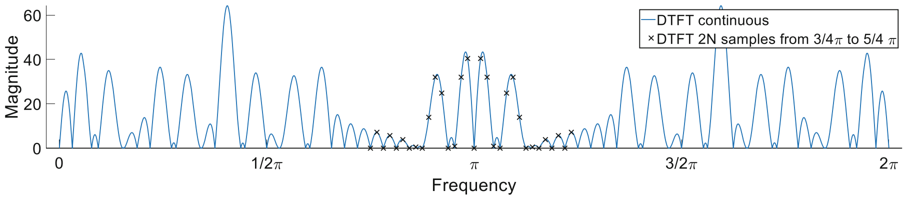

Fig. 6. Samples of the DTFT from *3/4π* to *5/4π* with *K = 32* (equivalent to a pruned oversampled DFT)

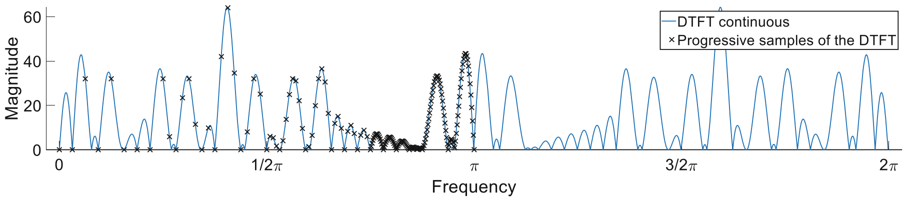

Fig. 7. Non-equally spaced samples of the DTFT from *0* to *π*

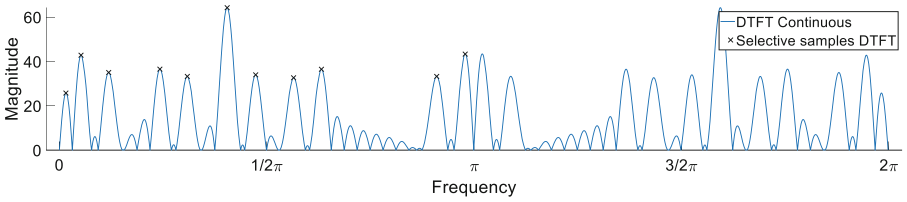

Fig. 8. Specific samples of the DTFT

The results of Fig. 3 could be compared against a DFT where the number of coefficients is equal to the length of the signal. Results in Fig. 4 could be compared against an oversampled DFT who has to be filled with zeros to be calculated. Results in Fig. 5 could be compared against a DFT where half of the coefficients were dropped after the calculations. Results in Fig. 6 could be compared against a pruned over-sampled DFT, which performs unnecessary operations with zeros. Results in Fig. 7 and Fig. 8 couldn't be compared against the DFT, because such tool cannot obtain non-equally spaced spectrum samples. Thus, the disadvantage of the DFT is its lack of versatility and in some cases the additional needed operations.

B. Execution Time

To determine the average time, 1000 executions of each instruction were carried out in an iterative cycle within Matlab, registering the total time. For each execution of different length, the register of the first execution was omitted to exclude the overhead made by the Matlab. The additional overhead for the control of the Matlab iterative cycle is not taken into account.

In order to compare the performance of the proposal method between the sequential version on CPU an parallel version on GPU, run-time tests are carried out by processing vectors of different lengths. The comparison is initially done with the implementation of the same method implemented on CPU. It is important to point out that the host to device transport overhead is omitted in both GPU implementations analysis as the signal is transferred directly in the device's global memory using the tools built into Matlab. The results obtained are shown in the graphs of Fig. 9, the experiments consist of processing sequences of lengths from 1 to 2048 on the DTFT. The results show that for vector longer than 394 elements the parallel version shows a better performance.

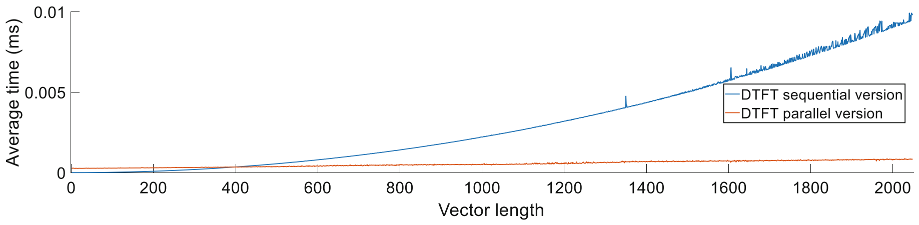

Fig. 9. Run time for sequential and parallel version.

The run times are carried out in order to analyze the viability of the proposed method. Even when the implementation is superior to the direct version on CPU in terms of speed, this could be improved with a more efficient methods matrix multiplication algorithm as cuBLAS.

5 Conclusions

The objective of this paper is to introduce and validate the basis of a versatile method for the Fourier analysis based on the DTFT of discrete, aperiodic and finite signals. This method allows to obtain a tool in which all possible scenarios can be achieved. The implementation takes great advantage from hardware accelerator available for HPC. In this implementation, it can be seen that the tool presents enough performance as to be used in the scenarios where the requirement of versatility is needed. Due to the simplicity of the algorithm, it is possible to implement it in other parallel platforms, including FPGA devices and other GPU devices like AMD and Intel.

Acknowledgment. The authors of this paper are grateful to the Tecnológico Nacional de México, Campus Culiacán, especially to the Division of Postgraduate Studies and Research for the facilities provided. I would also like to express my gratitude to the Consejo Nacional de Ciencia y Tecnología (Conacyt) for the support given during the course of the master's degree studies.

References

1. Oppenheim, A.V., Willsky, A.S.: Señales y Sistemas 2da. ed. Prentice Hall Hispanoamericana, México (1997)
2. Opepnheim, A.V., Shafer, R.W.: Tratamiento de señales en tiempo discreto. Pearson Educacion, Madrid (2011)
3. Proakis, J.G., Manolakis, D.G.: Digital Signal Processing. Principles, Algorithms, and Applications. Prentice Hall, Upper Saddle River (1996)
4. Cooley, J.W., Tukey, J.W.: An algorithm for the machine calculation of complex fourier series. Am. Math. Soc. **301**, 297 (1965)
5. Alves, R.G., Osorio, P.L., Swamy, M.N.S.: General FFT pruning algorithm. In: Proceedings of 43rd IEEE MidWest Symposium on Circuits and Systems, Lansing (2000)
6. Angulo Rios, J., Castro Palazuelos, D., Medina Melendrez, M., Santiesteban Cos, R.: A GPU based implementation of input and output prunning of composite length FFT using DIF DIT transform decomposition. In: Congreso Internacional de Ingenieria Electromecanica y de Sistemas CIEES, Ciudad de Mexico (2016)
7. Melendrez, M.M., Estrada, M.A., Castro, A.: Input and/or output pruning of composite length FFTs using a DIF-DIT transform decomposition. IEEE Trans. Sig. Process. **57**(10), 4124–4128 (2009)
8. Markel, J.D.: FFT prunning. IEEE Trans. Audio Electroacust. **4**, 305–311 (1971)
9. Goertzel, G.: An algorithm for the evaluation of finite trigonometric series. Am. Math. Monthly **65**(1), 34–35 (1958)
10. Stokfiszewski, K., Yatsymirskyy, M., Puchala, D.: Effectiveness of fast fourier transform implementations on GPU and CPU. In: International Conference on Computational Problems of Electric Engineerings (CPEE), pp. 162–164 (2015)
11. Shu, L.: Parallel implementation of arbitrary-sized discrete fourier transform on FPGA. In: 2016 3rd International Conference on Advanced Computing and Communication Systems (ICACCS), Coimbatore, India (2016)
12. Nvidia (2018). https://www.nvidia.com/en-us/high-performance-computing/. Accessed 17 Nov 2018
13. Oak Ridge National Laboratory (2018). https://www.ornl.gov/news/ornl-launches-summit-supercomputer. Accessed 17 Nov 2018
14. Evangelho, J.: AMD Claims Radeon Is #1 Gaming Platform – Here's Their Proof, Forbes, 23 April 2018. https://www.forbes.com/sites/jasonevangelho/2018/04/23/radeon-vs-geforce-which-brand-is-truly-the-1-gaming-platform/#4cfb19276a95. Accessed 17 Dec 2018
15. Fang, J., Varbanescu, A.L., Sips, H.: A comprehensive performance comparison of CUDA and OpenCL. In: 2011 International Conference on Parallel Processing, Taipei (2011)
16. Karimi, K., Dickson, N.G., Hamze, F.: A performance comparison of CUDA and OpenCL, arXiv (2010)