Sparse 2D Fast Fourier Transform

André Rauh and Gonzalo R. Arce
Department of Electrical and Computer Engineering
University of Delaware
Newark, DE 19716
Email: rauh@udel.edu, arce@udel.edu

***Abstract*—This paper extends the concepts of the Sparse Fast Fourier Transform (sFFT) Algorithm introduced in [1] to work with two dimensional (2D) data. The 2D algorithm requires several generalizations to multiple key concepts of the 1D sparse Fourier transform algorithm. Furthermore, several parameters needed in the algorithm are optimized for the reconstruction of sparse image spectra. This paper addresses the case of the exact k-sparse Fourier transform but the underlying concepts can be applied to the general case of finding a k-sparse approximation of the Fourier transform of an arbitrary signal. The proposed algorithm can further be extended to even higher dimensions. Simulations illustrate the efficiency and accuracy of the proposed algorithm when applied to real images.**

I. Introduction

The Fast Fourier Transform (FFT) has become ubiquitous in signal processing applications. While the FFT does not make any assumptions about the structure of the signal, very often the signal of interest is obtained from a structured source resulting in a nearly sparse Fourier spectrum. Assuming that a signal of length N is k-sparse ($k < N$) in the Fourier domain, we can describe the signal with only these k coefficients. This fact is the basis for signal compression and is used among others in the popular MP3 codec. Due to the fact that the signal is accurately described with just k coefficients it seems natural that there should be a better performing algorithm that exploits this property of the signal. Several algorithms have been proposed with this goal [2], [3], [4], [5], [6]. One particular approach is the so called sparse FFT (sFFT) which lowered the computational complexity significantly was introduced recently in [1]. The authors focused on the one dimensional case and the extension to two or multi-dimensions is not straight forward. Since the Fourier transform is separable, it is tempting to sequentially apply the 1D sFFT algorithm separately on all rows and columns. This approach, however, would not be efficient as the algorithm would be of complexity at least $O(N)$ for a signal of NxN samples which is not nearly as good as the proposed sub-linear algorithm. In addition, this strategy does not exploit the intrinsic two dimensionality of the signal and leads to a sub-optimal algorithm. This paper aims to fill this gap and introduces the extensions that are necessary to the algorithm.

The paper is organised as follows: Section II introduces the main ideas and workings of the sparse Fourier transform. Section III describes the new concepts and necessary modifications of the algorithm to extend it to 2D. Due to the sensitivity of the parameter of the algorithm, Section IV lays out guidance as to how the parameters should be selected under the assumption of a natural image as the input. Simulations and the conclusion are provided in the last two sections.

II. Sparse Fourier Transform Algorithm

It would be infeasible for this paper to describe in detail the sFFT algorithm in its entirety. Instead we refer the reader to [1] (up to page 9) and only describe the principal components of the algorithm which are necessary to understand the proposed extension. First, the notation is introduced. Note however that the notation will be re-used for the 2D case in the next Section. Given a signal x of length N we denote its discrete Fourier transform as $\hat{x}$. A signal is considered to be k-sparse if there are only k non-zero components in $\hat{x}$. Furthermore we define $\omega = e^{-2\pi\imath/N}$.

The key idea of the sFFT algorithm is to hash the k coefficients into few buckets in sublinear time. This is achieved by using a carefully designed filter that is concentrated in time as well as in the frequency domain. Due to the sparsity of the signal and the careful selection of the number of bins, each bin is likely to only contain one coefficient. After the coefficients of each bin are obtained the actual positions in the frequency domain are recovered by *locating* and *estimating*. The algorithm does this hashing twice and "encodes" the frequency of the coefficient into the phase difference between the two hashed coefficients. This technique achieves the *locating* part of the algorithm by decoding the phase and obtaining the frequency. Before the coefficients are hashed into buckets, the procedure (HashToBins) permutes the signal x in the time domain by applying the permutation operator P which is defined as

$$(P_{\sigma,a,b}x)_i = x_{\sigma(i-a)}\omega^{\sigma bi}, \tag{1}$$

where the parameter b is uniformly random between 1 and N, σ is uniformly random odd between 1 and N, and a is 0 for one hashing operation and 1 for the other. With the use of some basic properties of the Fourier transform the following can be proved (page 5 of [1]):

$$\widehat{P_{\sigma,a,b}x}_{\sigma(i-b)} = \hat{x}_i\omega^{a\sigma i}. \tag{2}$$

Informally, this equation states the following: A permutation, defined by equidistant subsampling in the time domain in addition to a linear phase, results in a permutation in the frequency domain with a linear phase. By carefully choosing the parameters of (2) it is possible to design the permutation such

that the phase difference between the two hashed coefficients is linear in frequency which can then be recovered.

The previous paragraph describes the key ideas of one iteration of the algorithm. A high level overview which was taken from [1] is the following:

- HASHTOBINS permutes the spectrum of $\widehat{x-z}$, then hashes to B bins. Where z is the already recovered signal which is initially all zero.
- NOISELESSSPARSEFFTINNER runs HASHTOBINS twice and *estimates* and *locates* "most" of $\widehat{x-z}$'s coefficients.
- NOISELESSSPARSEFFT iterates NOISELESSSPARSEFFTINNER until it finds $\hat{x}$ exactly.

NOISELESSSPARSEFFTINNER generates the random parameters for the permutation (among others) and passes it to HASHTOBINS. The permutations are $P_{\sigma,0,b}$ for the first call of HASHTOBINS and $P_{\sigma,1,b}$ for the second call respectively. The number of bins is denoted by B and gradually reduced with each call of NOISELESSSPARSEFFTINNER. HASHTOBINS performs an FFT on B samples and thus has a complexity of $O(B \log B)$. By carefully reducing B per iteration the 1D sFFT algorithm runs in time $O(k \log N)$. Again, see [1] for a detailed descriptions of the 1D sFFT algorithm.

III. EXTENSION TO 2D

For simplicity we will reuse the symbols and redefine the notation for the two dimensional case. Let x be an NxN signal with sparsity k, and the number of bins be BxB. It is intuitive to extend the filtering and permutation to two dimensions. However, the fact that the phase difference between the two hashes is always a one dimensional entity even in a 2D sample poses a problem. To be able to recover the frequencies in both dimensions it is necessary to hash a total of three times and encode one dimension in the second and the other dimension in the third call of HASHTOBINS. This allows to *locate* the coefficient in two dimensions. Additionally it is necessary to extend the permutation to 2D which is done with the following definition:

$$\begin{aligned}&(P_{\sigma_x,\sigma_y,\tau_x,\tau_y,a_x,a_y,b_x,b_y}x)_{i_x,i_y}\\ &= x_{\sigma_x i_x+a_x+\tau_x i_y,\sigma_y i_y+a_y+\tau_y i_x}\omega^{-(b_x\sigma_x i_x+b_y\sigma_y i_y)}.\end{aligned} \quad (3)$$

Note that, in addition to extending the permutation to two dimensions a new parameter τ was introduced to allow more powerful permutations. A similar equation as (2), which provides a relationship between the time and frequency domain, can be obtained for the 2D case:

$$\begin{aligned}&\widehat{(P_{\sigma,\tau,\mathbf{a},\mathbf{b}}x)}_{\sigma_x(i_x-b_x)+\tau_x i_y,\sigma_y(i_y-b_y)+\tau_y i_x}\\ &\quad= \hat{x}_{i_x,i_y}\omega^{a_x\sigma_x i_x}\omega^{a_y\sigma_y i_y}.\end{aligned} \quad (4)$$

For the proposed algorithm the high level overview is similar to that previously introduced in Section II. The main difference is within the function NOISELESSSPARSEFFTINNER which needs to properly select the parameters $\sigma_x, \sigma_y, \tau_x, \tau_y$, b_x, b_y and a_x, a_y. For the three calls of HASHTOBINS, a_x, a_y are selected as follows:

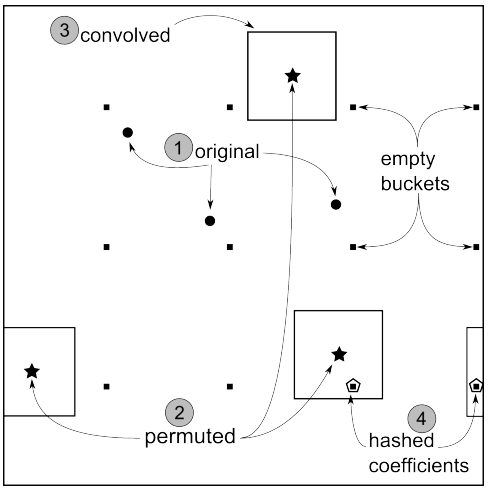

Fig. 1. Graphical depiction of the steps performed in HASHTOBINS. The original spectrum (1) has only three non-zero coefficients ($k = 3$) which are then permuted (2) and convolved with the low pass filter (3). Note that only two coefficients are hashed (4) and the third (a) is missed. There is no collisions in this particular example which could occur if the spectrum overlaps with neighboring coefficients and the area is hashed.

1) $a_x = 0$, $a_y = 0$
2) $a_x = 1$, $a_y = 0$
3) $a_x = 0$, $a_y = 1$.

This approach encodes the frequency of the first dimension in the phase difference of the first and second hashed coefficients and the frequency of the second dimension in the phase difference between the first and third hashed coefficient, respectively. In order to allow the reconstruction of the frequency by inverting the applied permutation, the parameter σ needs to be carefully chosen. In the one dimensional case the constraint for the parameter σ was for it to be odd. For the two dimensional case the following conditions are to be met:

$$\sigma_x \text{ odd}, \sigma_y \text{ odd}, \tau_x \text{ even}, \tau_y \text{ even}$$

or

$$\sigma_x \text{ even}, \sigma_y \text{ even}, \tau_x \text{ odd}, \tau_y \text{ odd}.$$

These constraints ensure that the permutation applied in HASHTOBINS is reversible which is necessary to decode the frequencies in NOISELESSSPARSEFFTINNER.

The newly proposed algorithm has similar parameters as the 1D algorithm. Though, at certain locations the parameters need to be changed two accommodate for two dimensions or extended to two dimensions. An example is the number of bins B which changes to B^2. Since the total number of non-zero efficient is still k, the parameter k occurring in NOISELESSSPARSEFFT is changed to $\sqrt{k}$.

The new procedure NOISELESSSPARSEFFTINNER generates the random parameters according to the constraints laid out above and calls HASHTOBINS three times after which the frequency locations can be recovered and $w_{i_x,i_y} = v$ is performed for "most" of $\widehat{x-z}$ where i_x and i_y are extracted from a combination of the three hashed coefficient and v is taken from the first hash as in the one dimensional case.

Fig. 1 depicts the concept of hashing the coefficients in two dimension.

The proposed algorithm uses the same filter as that introduced in [1] and extends it to two dimensions which is straight forward and is not discussed here.

IV. Optimal parameter selection

In [1], the authors only considered signals with random spectra. That is, spectra where the k non-zero coefficients have no structure. Often, however, signals encountered in real world applications are structured. For instance, audio signals often carry their majority of energy in harmonic frequencies. Additionally, an image often contains most of its energy in low frequency coefficients around the origin. This structure of Fourier coefficients is the foundation of signal compression where only the major coefficients are kept and low energy coefficients are discarded [7]. In many signal processing applications a randomized algorithm works extremely well [8]. Often, however, it is beneficial to exploit the inherent structure to obtain a better performing algorithm. In the proposed algorithm, if the parameters are chosen randomly, the performance can possibly be very poor which can be seen in Fig. 3.

In particular, in the 1D algorithm of [1] the parameters σ and b are chosen randomly as described after (1). In our proposed algorithm the parameters that need special attention are σ_x, σ_y and τ_x, τ_y. For the remaining of this paper we will assume that we deal with two dimensional data whose spectrum is concentrated around the origin.

The 2D permutation defined in (3) essentially performs a linear mapping of the following form:

$$\begin{pmatrix} i'_x \\ i'_y \end{pmatrix} \mapsto \begin{pmatrix} \sigma_x i_x + \tau_x i_y \\ \sigma_y i_y + \tau_y i_x \end{pmatrix} \tag{5}$$

In this form it is easy to see that σ and τ can be interpreted as scaling and shearing parameters. In particular the scaling is linear in σ_x and σ_y and the shear is linear in $S_x = \tau_x/\sigma_x$ and $S_y = \tau_y/\sigma_y$.

In order to optimize the parameters it is important to know that the low pass filter that is used in the algorithm has an approximately rectangular shape. It is also necessary to understand the inner workings of HashToBins:

First, the spectrum is permuted using the permutation in (3). Note that, if the permutation maps coefficients outside of the valid range of 1 to N, the number is automatically taken modulo N as the discrete Fourier transform is periodic with N. Next the permuted spectrum is convolved with the nearly rectangular two dimensional filter. Eventually the hashes are obtained by evenly subsampling the spectrum. It is important to note, that if the permuted samples are too close together the hashed coefficients can be errornous due to colliding filter windows after the convolution was applied.

Taking the above into consideration it is straight forward to see how the parameters σ and τ can be optimized such that the number of collisions are minimized:

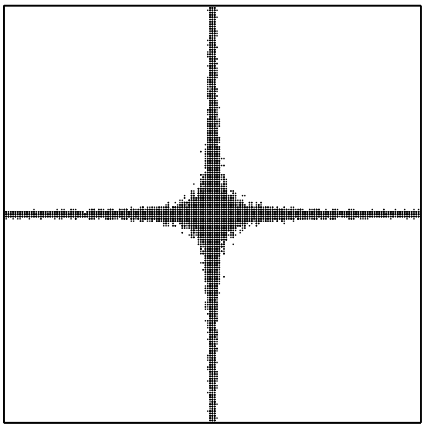

Fig. 2. A 3% sparse spectrum of a 2D Wafer. Note the coefficients are concentrated around the center and principal axes

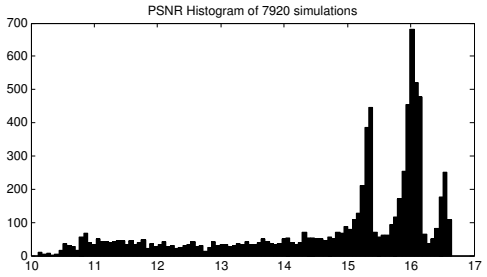

Fig. 3. Histogram of 7920 simulation of a 16384x16384 pixel Wafer with scale parameter S range from 3 to 4201

The first step is to minimize the number of parameters and to set the scale $C = \sigma_x = \sigma_y$ due to the fact that most natural occurring images have similar spectral characteristics along each dimension. An example spectrum is depicted in Fig. 2. Note that the coefficients are concentrated around the center and the principal axes. Secondly, the shear S_x and S_y are set such that $S_x \approx 1$ and $S_y \approx -1$ which can be achieved by setting $\tau_x = \sigma_x - 1$ and $\tau_y = -\tau_x$. This results in an approximately 45° rotation around the origin. This shear is crucial in achieving a low collision rate, as the coefficients along the principal axes would collide with each other without the shear. Furthermore, it is important to choose the scale parameter C carefully. Figure 3 depicts a histogram of a series of simulations where the scale S was swept from 3 to $N/2$. Note that, the PSNR can possibly be very poor if the scale parameter were to be chosen randomly. Instead, our proposed algorithm chooses the scale as $S^* = N/B - 1$ which more consistently results in a good performance in regards to PSNR. The scale S^* is chosen because each bin contains N/B samples and so that it is likely that only one coefficient falls into one bucket since the original coefficients are concentrated prior to the application of the permutation which in turn minimized collisions after the permutation.

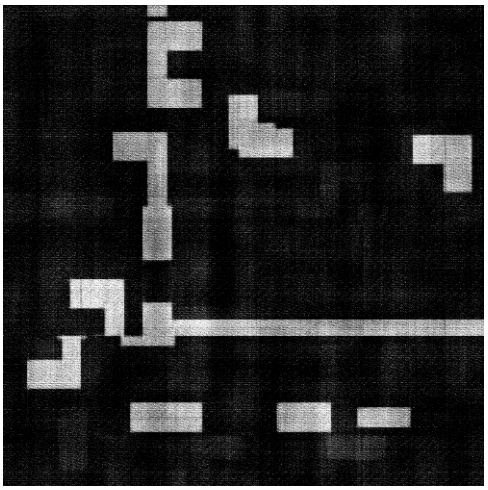

Fig. 4. A cropped 800x800px segment of a "reconstructed" 16384x16384px image of a Wafer. Input was 2% sparse.

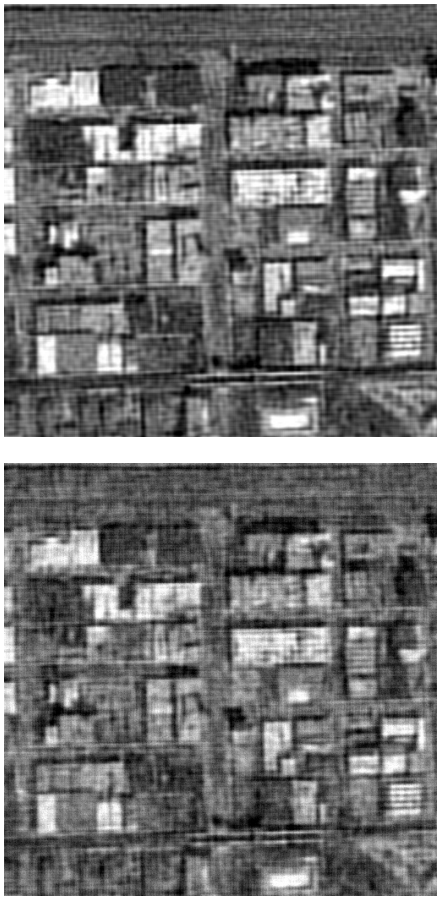

Fig. 5. Top: A 512x512 crop of a 2048x2048 image with a sparsity of 2%. Bottom: Image after running the proposed sFFT algorithm. The PSNR is 23.3dB when compared to the original sparse image.

V. Simulation results

We implemented the proposed algorithm in MATLAB and therefore only simulated the algorithm itself rather than implementing it in C/C++ and measuring real world speedup. Hence, no actual performance comparisons to a C/C++-implementation (such as FFTW) were carried out and the input signal size was limited to 32768^2 due to memory constraints. In order to compare the performance of different parameters, simulations terminated after one outer iteration of the algorithm. An example "reconstructed" image of a Wafer is depicted in Fig. 4. In this case, the Wafer image was sparsified to 3% of the coefficients and then the proposed algorithm was run on the sparse signal.

Figure 5 depicts a 512x512 crop of a 2048x2048 black and white image. The resulting bottom image shows that the proposed 2D sFFT algorithm successfully computed the sparse FFT. First, the original image was loaded, sparsified and then transformed to the spatial domain. This is the top image of Fig. 5. Then the 2D sFFT algorithm was applied to that image followed by an inverse FFT. This is the bottom image which has a PSNR of 23.3dB.

VI. Conclusion

In this paper a new sparse 2D Fourier transform algorithm was introduced. The proposed algorithm is based on the very efficient sFFT algorithm of [1]. The extension to 2D was done by hashing the coefficients into two dimensional buckets and decoding both frequencies from only three hashes. We showed that it is crucial to pay special attention to the parameters σ and τ of the newly introduced permutation, especially when dealing with natural images which usually have the main coefficients around the origin. The result is an algorithm with a time complexity of $O(k \log(N/k) \log^2 N)$ which is similar to the one dimensional algorithm of [1]. Even though, we only considered the optimization of the parameters of the 2D algorithm, the findings can be also be applied to the 1D algorithm when dealing with structured signals such as natural speech.

References

[1] H. Hassanieh, P. Indyk, D. Katabi, and E. Price, "Nearly optimal sparse fourier transform," *CoRR*, vol. abs/1201.2501, 2012.

[2] A. Akavia, S. Goldwasser, and S. Safra, "Proving hard-core predicates using list decoding," in *Annual Symposium on Foundations of Computer Science*, vol. 44. IEEE COMPUTER SOCIETY PRESS, 2003, pp. 146–159.

[3] A. Akavia, "Deterministic sparse fourier approximation via fooling arithmetic progressions," in *Proceedings of the 2010 Conference on Learning Theory, AT Kalai and M. Mohri, eds., Omnipress*, 2010, pp. 381–393.

[4] A. Gilbert, S. Guha, P. Indyk, S. Muthukrishnan, and M. Strauss, "Near-optimal sparse fourier representations via sampling," in *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*. ACM, 2002, pp. 152–161.

[5] M. Iwen, "Combinatorial sublinear-time fourier algorithms," *Foundations of Computational Mathematics*, vol. 10, no. 3, pp. 303–338, 2010.

[6] Y. Mansour, "Randomized interpolation and approximation of sparse polynomials," in *Automata, languages, and programming: 19th international colloquium, Wien, Austria, July 13-17, 1992: proceedings*, vol. 623. Springer, 1992, p. 261.

[7] A. Gersho and R. Gray, *Vector quantization and signal compression*. Springer, 1992, vol. 159.

[8] J. Tropp and A. Gilbert, "Signal recovery from random measurements via orthogonal matching pursuit," *Information Theory, IEEE Transactions on*, vol. 53, no. 12, pp. 4655–4666, 2007.