Journal of
Micro/Nanolithography,
MEMS, and MOEMS

SPIEDigitalLibrary.org/jm3

Fast continuous Fourier and Haar transforms of rectilinear polygons from very-large-scale integration layouts

Robin Scheibler
Paul Hurley
Amina Chebira

Fast continuous Fourier and Haar transforms of rectilinear polygons from very-large-scale integration layouts

Robin Scheibler,[a,*] Paul Hurley,[b] and Amina Chebira[a]
[a]Ecole Polytechnique Fédérale de Lausanne, School of Computer and Communication Sciences, Audiovisual Communications Laboratory, BC Building, Station 14, Lausanne, Switzerland
[b]IBM Research Zürich, Systems Department, Rüschlikon, Switzerland

Abstract. We propose two new fast algorithms for the computation of the continuous Fourier series and the continuous Haar transform of rectilinear polygons such as those of mask layouts in optical lithography. These algorithms outperform their discrete counterparts traditionally used. Not only are continuous transforms closer to the underlying continuous physical reality, but they also avoid the inherent inaccuracies introduced by the sampling or rasterization of the polygons in the discrete case. Moreover, massive amounts of data and the intense processing methods used in lithography require efficient algorithms at every step of the process. We derive the complexity of each algorithm and compare it to that of the corresponding discrete transform. For the practical very-large-scale integration (VLSI) layouts, we find significant reduction in the complexity because the number of polygon vertices is substantially smaller than the corresponding discrete image. This analysis is completed by an implementation and a benchmark of the continuous algorithms and their discrete counterparts. We run extensive experiments and show that on tested VLSI layouts the pruned continuous Haar transform is 5 to 25 times faster, while the fast continuous Fourier series is 1.5 to 3 times faster than their discrete counterparts. *© 2013 Society of Photo-Optical Instrumentation Engineers (SPIE)* [DOI: 10.1117/1.JMM.12.4.043008]

Keywords: continuous transform; fast Haar and Fourier algorithms; two-dimensional rectilinear polygons; very-large-scale integration; lithography.

Paper 12075 received Sep. 7, 2012; revised manuscript received Oct. 18, 2013; accepted for publication Nov. 21, 2013; published online Dec. 20, 2013.

1 Introduction

In optical lithography,[1] patterns of the integrated circuits are transferred to silicon by shining a light through a mask and subsequently using a lens to concentrate the light onto a photosensitive layer. This is followed by an etching step, which transfers the pattern to the silicon.

In recent years, the upgrade of manufacturing tools necessary to keep up with the fast-paced reduction in transistor size has not happened. As a consequence, ever more burden is placed onto the computationally intensive techniques to circumvent the optical degradation and thus ensure sufficient manufacturing yield. These techniques, collectively known as computational lithography, include traditional resolution enhancement,[2] source-mask optimization,[3] and inverse lithography.[4] They strive to exploit all the degrees of freedom in the lithography process, including illumination amplitude, direction, and phase.[5] All of these techniques rely on the computationally intensive simulation of the underlying physical processes. In parallel, very-large-scale integration (VLSI) layout file sizes are expanding rapidly, as ever more transistors are packed into a single design.[6] This coincides unfortunately with the increasing complexity of the aforementioned computational lithography algorithms. Taking these factors into account, having highly efficient and accurate algorithms at various steps of the lithography process become crucial.

As the lithography process is a continuous physical process, continuous transforms inherently offer a better representation than a discrete transform, as illustrated by the heavy use of the continuous Fourier transform in Fourier optics,[7] the physical foundation of the optical lithography. In particular, the sampling or rasterization of mask layouts prior to their transformation using the discrete Fourier transform (DFT) may introduce inaccuracies that might lead later to simulation errors.[8]

The Haar transform attempts to represent a target function as a linear combination of square-shaped basis functions. When applied to rectilinear polygons, this representation is remarkably efficient due to the similarity of the target function, the mask layout in our case, and the basis functions.

We believe the application of the Haar transform can lead to innovative techniques in lithography. One such application and our prime motivation for developing a fast Haar transform dedicated to rectilinear polygons is described by Kryszczuk et al.[9] Using the coefficients from orthogonal transforms, they borrow techniques from machine learning and train a classifier capable of predicting the outcome of the printing process without having to go through the costly physical simulation of the process. One such orthogonal transform that can be used in this context is the Haar transform.

To the best of our knowledge, uses of the discrete Haar transform in lithography have been very limited. In one case, a discrete Haar transform is used to compress the Fourier precompensation filters for electron-beam lithography.[10] In another case, the Haar transform is used in inverse lithography to regularize the obtained mask.[11] In a previous paper,[12] we introduced the pruned continuous Haar transform (PCHT), a fast algorithm to compute the continuous Haar transform coefficients. We extend[12] this with the complexity analysis of PCHT in this paper.

*Address all correspondence to: Robin Scheibler, E-mail: robin.scheibler@epfl.ch

0091-3286/2013/$25.00 © 2013 SPIE

The Fourier transform of the mask is a crucial step in the simulation of the lithography process owing to the Fourier transforming properties of lenses.[7] The underlying physical process is continuous, thus using the continuous Fourier series (CFS) is natural and should yield the closest result to the continuous Fourier transform used in the theoretical Fourier optics.

One popular technique for estimating the resultant aerial image from a mask is through photolithography simulation using the Hopkins method.[13] Fundamentally a continuous convolution underpins the analysis, yet this is currently performed using the fast Fourier transforms (FFT) of samples of the naturally continuous description of the mask and the Hopkins eigenfunctions. This situation may then be further exacerbated by the introduced sampling in the photoresist modeling convolution step. Optical proximity correction (OPC) typically involves many of these simulation iterations before possible convergence on an appropriately altered mask function, and accuracy in estimation is thus crucial. Using the CFS outlined here has inherently better accuracy and low complexity.

In addition, uses of the FFT include the computation of a precompensation filter to reduce the proximity effects,[14] and approximating the diffraction orders of the mask.[3] The importance of the two-dimensional (2-D) FFT in the computational lithography is underlined by a road map for its efficient use.[15] The idea of computing the CFS of polygons, not limited to those from VLSI layouts, in itself is not new and diverse derivations of closed form expressions exist.[16–19]

The inherently continuous nature of polygons—due to their physical nature when printed on the mask—makes a continuous transform a natural tool for VLSI layouts. To the best of our knowledge, algorithms making efficient use of this type of representation to quickly compute continuous transform coefficients do not exist. In this paper, we first extend our previous work on the PCHT algorithm.[12] We follow by presenting the first fast Fourier series algorithms applied to rectilinear polygons from VLSI layouts. Both algorithms are based on a closed-form formula that we derive for 2-D continuous separable transforms using a decomposition of rectilinear polygons into rectangles. This formulation leads to the derivation of two fast algorithms to compute the transform coefficients: PCHT and the fast continuous Fourier series (FCFS) algorithms. PCHT has a fast orthogonal wavelet transform (FWT) structure that is pruned using the computational geometry techniques. FCFS results from reducing the CFS computation problem to a few sparse DFTs computed using pruned FFT algorithms. We evaluate the computational complexity of both algorithms.

To validate the practical performance of our implementation, we run extensive and rigorous runtime measurements on real VLSI layouts and compare the outcome to the performance of their discrete counterparts. We use these results to compute the speed-up provided by the continuous transforms and find PCHT and FCFS to be up to 25 and 3 times, respectively, faster than the discrete Haar transform and the FFT, respectively. The runtimes are also found to be consistent with the computational complexity derived for all algorithms. In addition, we show an example of aerial image simulation using both FCFS and conventional FFT methods.

This paper is organized as follows. Section 2 introduces the necessary background in VLSI layouts, the continuous Haar transform (CHT), and the CFS. Section 3 presents a framework for taking continuous transforms of rectilinear polygons as well as the proposed PCHT and FCFS. In Sec. 4, the performance of the both algorithms is evaluated and compared to that of their discrete counterparts. Finally, Sec. 5 concludes by discussing the superiority of PCHT and FCFS over their discrete equivalents, and sketches the possible future directions.

2 Background

In this section, we first briefly describe VLSI layouts and how they are produced. We consider only layouts composed exclusively of rectilinear polygons since they are predominent in technology nodes under 45 nm. These rectilinear polygons are described mathematically, laying the groundwork for the algorithms to come. We then describe the 2-D continuous Haar transform and the fast wavelet transform algorithm used to implement it. Finally, a short refresher on the CFS is given.

2.1 *VLSI Layouts*

2.1.1 *Layouts*

VLSI layouts are composed of several layers, each containing many billions of rectangles, or more generally, rectilinear polygons. Figure 1 shows the fragments of three different types of layers. These are taken from metal 1 (M1), which is mostly random logic, metal 2 (M2), containing some logic and wires, and contact array (CA), providing contacts between the different layers. In addition, there are several other types of layers, omitted here, similar to those of M1, M2, or CA.

2.1.2 *Rectilinear polygons*

We now give a formal definition of the polygons in VLSI layouts. They are rectilinear (only right angles), simple

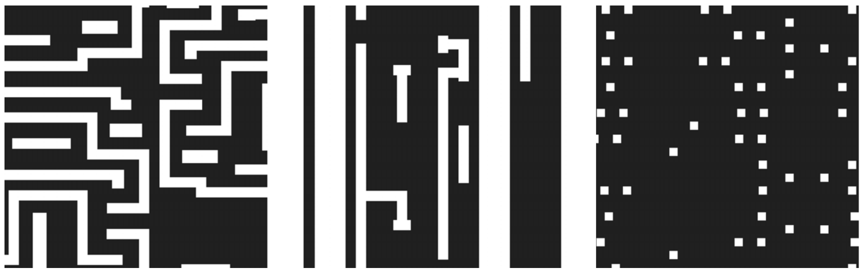

Fig. 1 From left to right: Illustrative examples of 1024 nm × 1024 nm tiles from metal 1 (M1), metal 2 (M2), and contact array (CA) layers, respectively. Note that they exclusively contain rectilinear polygons.

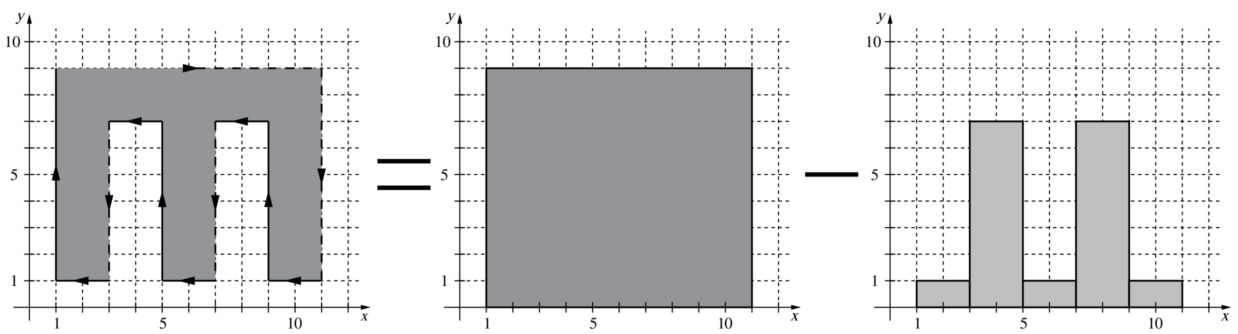

Fig. 2 On the left, a rectilinear simple lattice polygon. The interior of the polygon is shaded. The full-lined edges are included in the polygon, while the dashed edges are not. Arrows indicate vertex ordering. On the right, illustration of the construction of the left polygon from disjoint rectangles. The minus here stands for the set difference operator.

(edges do not intersect, no holes), lattice (vertices are on the integer lattice) polygons. An example of such a polygon is shown in the left panel of Fig. 2. A standard layout description consists of polygons defined by the set of ordered coordinates of their K vertices. This set separates the plane in two: inside and outside of the polygon. A sequence order is necessary, and we arbitrarily choose it to be clockwise. A disjoint partition of the plane is achieved when we exclude edges if the interior of the polygon is on their left or bottom and include them if the interior is on the right or top (see Fig. 2). Rectangles are rectilinear polygons with four vertices, and as they are simpler to handle, we treat them as a special case.

2.2 *Continuous Haar Transform*

Much like the CFS decomposes a function into a sum of the sines and cosines, the continuous Haar transform decomposes a function into a sum of the rectangular functions called the Haar basis. The 2-D Haar basis over $\boldsymbol{T} = [0, N_x) \times [0, N_y)$ is

$$\left\{\varphi_{0,0,0}, \psi^{(hg)}_{j,k_x,k_y}, \psi^{(gh)}_{j,k_x,k_y}, \psi^{(hh)}_{j,k_x,k_y}\right\},$$

where $j \in \mathbb{N}$ and $k_x,\ k_y \in \{0, \ldots, 2^j - 1\}$. In practice, j is limited to some maximum level of decomposition J. The scaling function is $\varphi_{j,k_x,k_y}(x, y) = (2^j/\sqrt{N_x N_y})$ if $(x, y) \in \boldsymbol{T}_{j,k_x,k_y} = [\frac{k_x N_x}{2^j}, \frac{(k_x+1)N_x}{2^j}) \times [\frac{k_y N_y}{2^j}, \frac{(k_y+1)N_y}{2^j})$ and 0 otherwise.

The three other basis functions can be defined using a recursive relationship. For example

$$\psi^{(hg)}_{j,k_x,k_y}(x, y) = \sum_n \sum_m h_n g_m \varphi_{j+1,2k_x+n,2k_y+m}(x, y), \tag{1}$$

where $g_n = [2^{-1/2}, 2^{-1/2}]$ and $h_n = [2^{-1/2}, -2^{-1/2}]$ are the Haar filters. By replacing $h_n g_m$ in the sum by $g_n g_m$, $g_n h_m$, and $h_n h_m$, we obtain φ_{j,k_x,k_y}, $\psi^{(gh)}_{j,k_x,k_y}$, and $\psi^{(hh)}_{j,k_x,k_y}$, respectively. The dyadic CHT of a function f is given by its inner product with the basis functions. The discrete counterpart of the CHT is the discrete Haar transform (DHT). A more thorough introduction to the CHT and DHT is given in Vetterli et al.[20] The CHT and DHT coefficients are identical for the 2-D rectilinear polygonal patterns. Both can be computed using the FWT.[21] This algorithm has a Cooley–Tukey butterfly structure,[22] where only the inner products with the scaling function at the lowest level need to be computed. The full flow diagram for a length-8 one-dimensional (1-D) FWT is shown in light gray in Fig. 3.

2.3 *Continuous Fourier Series*

The 2-D Fourier basis over $\boldsymbol{T} = [0, N_x) \times [0, N_y)$ is

$$\{(N_x N_y)^{-1/2} e^{j(w_x kx + w_y ly)}\}_{(k,l)\in\mathbb{Z}^2}, \tag{2}$$

where $w_x = (2\pi/N_x)$ and $w_y = (2\pi/N_y)$. The Fourier basis assumes that the function f under transformation is periodic with the periods N_x and N_y along the x-axis and y-axis, respectively. The CFS coefficients $\hat{F}_{k,l}$ are then given by the inner product between f and the Fourier basis functions.

This differs from the DFT in that the functions are continuous in the spatial domain, and thus not periodic in the frequency domain. This means that for the perfect reconstruction of the image, an infinite number of coefficients is needed. But in reality, we only care about reconstructing the output of the lithographic system which is limited to a fairly small number of low frequency coefficients.[1] On the other hand, the use of the DFT requires sampling, which introduces aliasing due to the infinite bandwidth of rectilinear polygons, and in turn yields an inaccurate spectral representation of the continuous image. In contrast, the CFS yields the true spectrum of the continuous image.

3 Continuous Transforms of Rectilinear Polygons

In this section, we derive the algorithms to compute the continuous Haar transform and Fourier series coefficients of

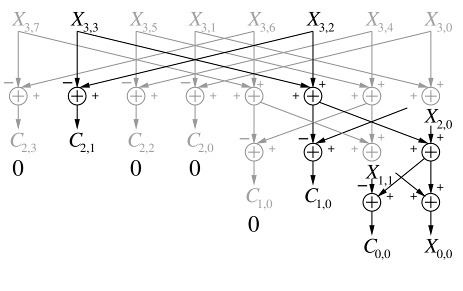

Fig. 3 A pruned signal flow of the one-dimensional (1-D) Haar fast orthogonal wavelet transform. The full flow diagram is shown in light gray. The transformed signal is $f(t) = u(t-3)$, defined on $[0, 8)$, where $u(t)$ is the Heaviside function. $X_{j,k} = \langle f, \varphi^{(8)}_{j,k}\rangle$ and $C_{j,k} = \langle f, \psi^{(8)}_{j,k}\rangle$ where $\varphi^{(8)}_{j,k}$ and $\psi^{(8)}_{j,k}$ are the 1-D Haar basis functions on $[0, 8)$. For simplicity, the scaling of the transform coefficients has been omitted.

rectilinear polygons. Their theoretical computational complexity is evaluated and compared with that of the equivalent discrete transforms.

The main reason continuous transforms can be faster than their discrete counterparts for rectilinear polygons is that the continuous inner product of a basis function is an explicit function of the vertices of the polygon, and the vertex description is very sparse compared to the size of the image. Moreover, no memory is needed to form or store a discrete image. In addition, the sampling of the polygons to create a discrete image can itself be considered a projection on a Dirac basis. Sampling followed by a discrete transform, as illustrated in Fig. 4, effectively is two transforms, whereas a continuous transform can completely omit the sampling operation. Moreover, as discussed in Sec. 1, the CFS circumvents inaccuracies stemming from the sampling required by the FFT.

3.1 *Inner Product over Rectilinear Polygons*

In practice, to cope with very large layouts, a divide-and-conquer strategy is adopted. Layouts are divided into smaller disjoint or overlapping rectangular tiles before applying a transform to each individual tile. Therefore, we consider continuous transforms over a subset $\boldsymbol{T} = [0, N_x) \times [0, N_y) \subset \mathbb{R}^2$ that we call a tile. This poses no restrictions as the size of this subset can be increased sufficiently to cover the entire layout. The size of the tile is chosen with respect to the maximum radius of influence.[1] In practice, N_x and N_y are always chosen to be positive integers.

We use Hilbert space formalism to describe the orthogonal transforms.[20] Given an orthogonal basis of functions $\phi_{k,l}$ over $\boldsymbol{T}$, $\{\phi_{k,l}\colon \boldsymbol{T} \to \mathbb{C}\}_{k=0,l=0}^{\infty,\infty}$, the transform coefficients of a function $f \in L^2(\boldsymbol{T})$ are the inner products with the basis functions

$$\langle f, \varphi_{k,l} \rangle = \iint_{\boldsymbol{T}} f(x,y)\phi_{k,l}^*(x,y)\mathrm{d}x\mathrm{d}y. \tag{3}$$

Applying this definition to the Haar and Fourier basis, as described in Eqs. (1) and (2) respectively, yields the Haar and Fourier transforms, respectively. As all polygons in a layout are disjoint, the image f_T of a tile containing polygons $\mathcal{P}_0, \ldots, \mathcal{P}_{M-1} \subseteq T$ is the sum of the indicator functions of the polygons

$$f_T(x,y) = \sum_{m=0}^{M-1} \mathbb{1}_{\mathcal{P}_m}(x,y). \tag{4}$$

The indicator function is defined as $\mathbb{1}_{\mathcal{P}}(x,y) = 1$ if (x,y) is inside the polygon $\mathcal{P}$ and 0 otherwise. Now, for functions such as Eq. (4), and separable basis functions, the inner product of Eq. (3) can be computed as a sum of the functions over the polygon vertices.

Lemma 3.1 Consider a rectilinear polygon $\mathcal{P}$ with K vertices $\{x_i, y_i\}_{i=0}^{K-1}$, and a separable basis function, namely $\phi_{k,l}(x,y) = \phi_k(x)\phi_l(y)$. Let the function $\Omega = \int\phi$ be a primitive of ϕ. Then

$$\langle f_{\mathcal{P}}, \phi_{k,l} \rangle = \sum_{i=0}^{K/2-1} \Omega_l^*(y_{2i})[\Omega_k^*(x_{2i+1}) - \Omega_k^*(x_{2i})], \tag{5}$$

where indices are taken modulo K.

The proof of this result is given in Appendix A. This result can be extended to multiple disjoint polygons by the linearity of the inner product.

3.2 *Pruned Continuous Haar Transform*

We briefly describe the previously introduced PCHT algorithm[12] for rectilinear polygons. Then, we advance this previous work with the derivation of the complexity of the PCHT.

3.2.1 *Algorithm derivation*

Consider the FWT as described in Sec. 2.2 and whose full 1-D flow diagram is shown in light gray in Fig. 3. First, note that the linear complexity of the FWT and Eq. (13) mean that we can use the FWT on individual polygons and sum up the transform coefficients to obtain the transform of a tile, as illustrated in Fig. 4. Thus, from now on, we consider only the transform of a single polygon. Second, we use the computational geometry techniques to compute the inner product. The continuous inner product between the indicator of a polygon and the support of the scaling function as defined in Sec. 2.2, as given by Lemma 3.1, is the area of the geometrical intersection of the polygon and the scaling function multiplied by $2^j/\sqrt{N_x N_y}$. A method to compute the intersection area for the rectilinear polygons is described in Algorithm 1.

The Haar transform acts as a discontinuity detector, and all transform coefficients will be zero except when basis functions intersect the boundary of the polygon. Therefore, the basis functions completely inside or outside the polygon can be ignored. In addition, all coefficients positioned below such basis functions in the transform tree are also zero.

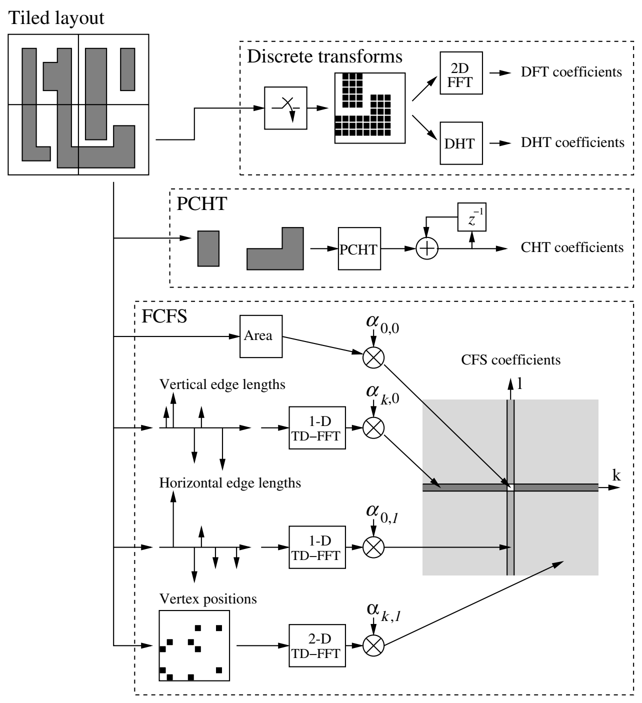

Fig. 4 Flow diagram of all the transforms. From top to bottom: fast Fourier transform (FFT), discrete Haar transform (DHT), pruned continuous Haar transform (PCHT), and fast continuous Fourier series (FCFS).

Algorithm 1 Intersection area ($\mathscr{P}$, $\mathbf{T}_{j,k_x,k_y}$).

```
Require: A rectilinear polygon $\mathscr{P}$. The support $T_{j,k_x,k_y}$ of the basis function $\varphi_{j,k_x,k_y}$.
Ensure: $I$ is the intersection area of $\mathscr{P}$ and $T_{j,k_x,k_y}$.
1: $I \leftarrow 0$
2: $a_1 \leftarrow k_x N_x / 2^j$, $b_1 \leftarrow k_y N_y / 2^j$
3: $a_2 \leftarrow (k_x + 1) N_x / 2^j$, $b_2 \leftarrow (k_y + 1) N_y / 2^j$
4: for every horizontal edge $(x_i, y_i) \to (x_{i+1}, y_i)$ of $\mathscr{P}$ do
5:     $u \leftarrow \min(x_i, x_{i+1})$, $v \leftarrow \max(x_i, x_{i+1})$
6:     if $a_2 \le u$ or $v \le a_1$ or $y_i \le b_1$ then
7:         continue
8:     end if
9:     $s \leftarrow \mathrm{sign}(x_{i+1} - x_i)$
10:    $I \leftarrow I + s[\min(v, a_2) - \max(u, a_1)][\min(y_i, b_2) - b_1]$
11: end for
(where $\leftarrow$ denotes assignment)
```

Figure 5 shows in gray the support of the basis functions at a given scale that yield nonzero inner products. As with the original FWT, PCHT can be written as a divide-and-conquer algorithm: divide the tile into four rectangular parts recursively until the part under consideration is completely inside or outside the polygon. This happen ultimately when we reach the granularity of the grid and the basis functions are 1×1. Pseudocode for the PCHT can be found in the previous work.[12] An example of the pruned transform flow diagram for the 1-D case is shown in Fig. 3 in black.

3.2.2 *Complexity*

We will now estimate the complexity of the PCHT. As it is highly dependent on the geometry of the polygon, we make the worst case assumption and describe the complexity as a function of a few general properties of the polygon, namely its number of vertices K and its perimeters P. The complexity is also a function of the computational complexity $\Lambda_{ia}(K)$ of the intersection area algorithm. The exact value depends on the assumptions made regarding the type of polygons, but is $O(K)$.[23] In particular, for the rectilinear polygons, we have

$$\Lambda_{ia}(K) = \begin{cases} 11 & \text{if } K = 4 \\ 6K - 1 & \text{if } K > 4 \end{cases}, \tag{6}$$

additions, multiplications, and comparisons.

We begin by estimating the number of nonterminating recursive calls. A call is nonterminating if the support of the waveform intersects the boundary of the polygon. Assume here that $N_x = N_y = N$. Then, at a given scale j, there are roughly $\lceil 2^j P / N \rceil$ such basis functions. Each of these calls makes four calls to itself and thus to the intersection area routine and also uses three comparisons. The intersection area routine uses in turn 11 additions and three multiplications per nonterminating call. Finally, a total of $M - 1$ additions are needed to sum up the scaling coefficients of the polygons, and a single multiplication is needed for the final scaling factor. We thus formulate our estimate of the complexity of the PCHT as

$$\Lambda_{\mathrm{PCHT}} \approx \left\{ \sum_{m=0}^{M-1} \sum_{j=0}^{J-1} \left\lceil \frac{2^j P_m}{N} \right\rceil [\Lambda_{ia}(K_m) + 26] \right\} + M \tag{7}$$

operations (additions, multiplications, and comparisons), where P_m and K_m are the perimeter and number of vertices of the m'th polygon, respectively, M is the number of polygons in a given tile, and J is the maximum level of decomposition. For a DHT with $N_x = N_y = N = 2^J$, the total number of operations is $\Lambda_{\mathrm{DHT}} = (8/3)(N^2 - 1)$. The relationship between the complexity and the actual runtime is discussed in Sec. 4.

3.3 *Fast Continuous Fourier Series*

We now develop the FCFS algorithm for computing the CFS coefficients of rectilinear polygons. The computational complexity of the FCFS will be evaluated and compared with that of the FFT.

3.3.1 *Algorithm derivation*

Recall the definitions of w_x, w_y, N_x, and N_y in Sec. 2.3. We begin by a closed-form formula for the CFS coefficients of rectilinear polygons. Applying Lemma 3.1 to the Fourier basis equation (2) results in the following proposition:

Proposition 1. Given a polygon $\mathcal{P}$ with K vertices, its CFS $\hat{F}_{k,l}$, k, $l \in \mathbb{Z}^2$, is given by

$$\hat{F}_{0,0} = \alpha_{0,0} \sum_{i=0}^{K/2-1} y_{2i}(x_{2i+1} - x_{2i}), \tag{8}$$

$$\hat{F}_{k,0} = \alpha_{k,0} \sum_{i=0}^{K/2-1} (y_{2i-1} - y_{2i}) e^{-jw_x k x_{2i}}, \tag{9}$$

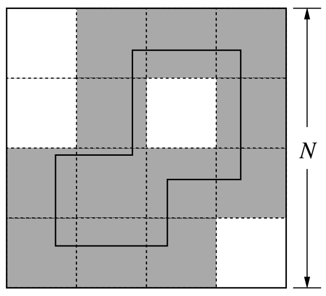

Fig. 5 The Haar basis functions yield zero inner product except when they intersect the edge of a polygon. The big thick square is the tile with $N_x = N_y = N$. The dashed squares are the supports of the basis functions at a given scale. The thick polygonal border marks the contour of the polygon. Basis functions yielding nonzero inner product are shown in gray.

$$\hat{F}_{0,l} = \alpha_{0,l} \sum_{i=0}^{K/2-1} (x_{2i+1} - x_{2i}) e^{-jw_y l y_{2i}}, \tag{10}$$

$$\hat{F}_{k,l} = \alpha_{k,l} \sum_{i=0}^{K/2-1} e^{-jw_y l y_{2i}} (e^{-jw_x k x_{2i+1}} - e^{-jw_x k x_{2i}}), \tag{11}$$

where the scaling factor $\alpha_{k,l}$ is defined as follows:

$$\alpha_{k,l} = \begin{cases} \frac{1}{\sqrt{N_x N_y}} & \text{if } k = l = 0 \\ \frac{j}{2\pi k}\sqrt{\frac{N_x}{N_y}} & \text{if } k \neq 0 \quad \text{and} \quad l = 0 \\ \frac{j}{2\pi l}\sqrt{\frac{N_y}{N_x}} & \text{if } k = 0 \quad \text{and} \quad l \neq 0 \\ -\frac{\sqrt{N_x N_y}}{4\pi^2 kl} & \text{if } k \neq 0 \quad \text{and} \quad l \neq 0 \end{cases}$$

It can be observed that except for $\hat{F}_{0,0}$, all the coefficients are DFT coefficients of very sparse signals as we restrict the vertices to lie on the integer lattice. We can decompose the fast algorithm into four main steps, with each step computing one of Eqs. (8)–(11), steps 2 and 3 using the DFT formalism. For simplicity, we consider only the transform of a single polygon here:

Step 1: Directly compute Eq. (8). It is the scaled area of $\mathcal{P}$. $\hat{F}_{0,0} = \alpha_{0,0}\text{Area}(\mathcal{P})$. This corresponds to lines 11 and 18 in Algorithm 2.

Step 2: Compute Eq. (9) as a 1-D DFT: $\hat{F}_{k,0} = \alpha_{k,0}\text{DFT}_{k'}\{\tilde{f}_x\}$, where $k' \equiv k \bmod N_x$ and

$$\tilde{f}_x[n] = \sum_{i \in \mathcal{X}_n} (y_{i-1} - y_i), \qquad n = 0, \ldots, N_x - 1$$

where $\mathcal{X}_n = \{i | x_i \equiv n \bmod N_x\}$. The values $(y_{i-1} - y_i)$ are the lengths of the vertical edges. This corresponds to lines 5 and 16 in Algorithm 2.

Step 3: Compute Eq. (10) as a 1-D DFT: $\hat{F}_{0,l} = \alpha_{0,l}\text{DFT}_{l'}\{\tilde{f}_y\}$, where $l' \equiv l \bmod N_y$ and

$$\tilde{f}_y[n] = \sum_{i \in \mathcal{Y}_n} (x_{i+1} - x_i), \qquad n = 0, \ldots, N_y - 1$$

where $\mathcal{Y}_n = \{i | y_i \equiv n \bmod N_y\}$. The values $(x_{i+1} - x_i)$ are the lengths of the horizontal edges. This corresponds to lines 8 and 17 in Algorithm 2.

Step 4: Compute Eq. (11) as a 2-D DFT: $\hat{F}_{k,l} = \alpha_{k,l}\text{DFT}_{k',l'}\{\tilde{f}_{xy}\}$, where $k' \equiv k \bmod N_x$, $l' \equiv l \bmod N_y$ and

$$\tilde{f}_{x,y}[m,n] = \sum_{i=0}^{K/2-1} [\mathbb{1}_{\mathcal{Z}_{m,n}}(x_{2i+1}, y_{2i}) - \mathbb{1}_{\mathcal{Z}_{m,n}}(x_{2i}, y_{2i})],$$

where $\mathcal{Z}_{m,n} = \{(x,y) | (x,y) \in \mathbb{Z}^2, x \equiv m \bmod N_x, y \equiv n \bmod N_y\}$. This is a sparse $N_x \times N_y$ image with 1's and −1's placed at the vertices. This corresponds to lines 9, 10, and 15 in Algorithm 2.

For multiple polygons, we sum the areas of all polygons for step 1, and the linearity of the DFT is used to include all the polygons in $\tilde{f}_x$, $\tilde{f}_y$, and $\tilde{f}_{xy}$ for the three other steps. The four steps are illustrated in Fig. 4. Algorithm 2 gives the pseudocode of the FCFS.

Algorithm 2 FCFS $(\mathcal{P}, N_x, N_y, F)$.

Require: $\mathcal{P}$ contains the lists of the vertices of the M rectilinear polygons where $(x_{m,i}, y_{m,i})$ is the i'th, out of K_m, vertex of the m'th polygon $\mathcal{P}_m$. For all $m: x_{m,0} \neq x_{m,1}$. The tile size is $N_x \times N_y$. TD-FFT 1-D and TD-FFT 2-D take as input arrays containing the nonzero values (V), their locations (S), and number, along with the DFT size.

Ensure: F is an $N_x \times N_y$ matrix containing the $N_x \times N_y$ first Fourier series coefficients of the polygon $\mathcal{P}$.

```
1: $n \leftarrow 0$; $A \leftarrow 0$
2: for $m = 0$ to $M - 1$ do
3:   for $i = 0$ to $K_m - 1$ do
4:     if $x_{m,i} = x_{m,i+1}$ then
5:       $V^K[n] \leftarrow y_{m,i} - y_{m,i+1}$; $S^K[n] \leftarrow x_{m,i}$
6:       $n \leftarrow n + 1$
7:     else if $y_{m,i} = y_{m,i+1}$ then
8:       $V^L[n] \leftarrow x_{m,i+1} - x_{m,i}$; $S^L[n] \leftarrow y_{m,i}$
9:       $V^{KL}[2n] \leftarrow 1$; $S^{KL}[2n] \leftarrow (x_{m,i+1}, y_{m,i})$
10:      $V^{KL}[2n+1] \leftarrow -1$; $S^{KL}[2n+1] \leftarrow (x_{m,i}, y_{m,i})$
11:      $A \leftarrow A + y_{m,i}(x_{m,i+1} - x_{m,i})$
12:    end if
13:  end for
14: end for
15: $F[k,l] \leftarrow \alpha_{k,l}\mathrm{TD-FFT-2-D}(V^{KL}, S^{KL}, K, N_x, N_y)$
16: $F[k,0] \leftarrow \alpha_{k,0}\mathrm{TD-FFT-1-D}(V^K, S^K, K/2, N_x)$
17: $F[0,l] \leftarrow \alpha_{0,l}\mathrm{TD-FFT-1-D}(V^L, S^L, K/2, N_y)$
18: $F[0,0] \leftarrow \alpha_{0,0}A$
```

Having reduced the problem of computing the CFS to a few real DFTs of sparse signals, we can exploit the extensive collection of available DFT algorithms. The FFT algorithm[24] is not the most efficient in our case, as we could also use a single FFT on a sampled version of f_T, albeit with some loss of precision. If we are only interested in a few CFS coefficients, we can apply a Goertzel-like algorithm[25] to the direct computation of the CFS coefficients using Proposition 1. The input pruned FFT, whereas the FFT structure is pruned for a sparse input, is the most appealing approach for the computation of a large number of CFS coefficients, and in particular, the transform decomposition FFT (TD-FFT) as it is the fastest of all existing pruned FFT algorithms. The implementation of the 1-D TD-FFT can be found in Refs. 26 and 27. The implementation of the 2-D TD-FFT is a straightforward extension of the 1-D TD-FFT to the 2-D case. However, to the best of our knowledge, it has not been reported in the literature and we thus give a short derivation as well as pseudocode in Appendix B. It is worth noting that neither the 1-D nor 2-D TD-FFT algorithms assume the nonzero inputs to be consecutive. Our complexity analysis considers both TD-FFT and Goertzel, while our implementation focuses solely on TD-FFT.

3.3.2 *Complexity*

As in the Haar case, the computational complexity depends on the polygon geometry. In this case, only the sum of the number of vertices K of the M polygons present in the tile is important.

We compare, in terms of complexity, FCFS with the split-radix FFT (Ref. 28) performed on the discrete image of a polygon. We choose the split-radix FFT algorithm since it is one of the fastest and most widely used FFT algorithms. For FCFS, we consider both Goertzel and TD-FFT algorithms to compute the sparse DFTs.

For the complexity analysis, consider the transform of an $N_x \times N_y$ tile, where N_x and N_y are the composite numbers (not prime). Table 1 details the complexity of each step of the FCFS algorithm described in Sec. 3.3.1. We need to choose the TD-FFT subFFTs lengths $P_x^{(\mathrm{I})}$, $P_y^{(\mathrm{I})}$, $P_x^{(\mathrm{II})}$, and $P_y^{(\mathrm{II})}$ such that the overall complexity is minimized with the constraint that they are dividers of N_x and N_y, respectively.[27] There is, however, no closed-form solution to this problem. In addition, this optimization requires knowledge of the specific FFT algorithm used for the subFFTs in TD-FFT, which is impossible with modern libraries such as FFTW. Therefore, the lengths of the subFFTs need to be chosen such that the runtime on a given architecture is minimized. This can be done offline and the optimal lengths stored in a look-up table.

For the sake of analysis, we assume that the split-radix FFT algorithm is used for all FFTs. The complexity of the 2-D $N_x \times N_y$ complex split-radix FFT is

$$C_{\mathrm{FFT2-D}} = 4N_xN_y \log_2 N_xN_y - 12N_xN_y + 8N_x + 8N_y$$

real operations. Real-valued data implies the need for roughly half this number of operations. The Goertzel algorithm requires approximately $O(KN_xN_y)$ real operations for an $N_x \times N_y$-point real DFT with K nonzero inputs. Our algorithm requires the computation of the area of the polygons (step 1), two length-N DFTs with $K/2$ inputs each (steps 2 and 3) and one 2-D length-$N_x \times N_y$ DFT with K inputs (step 4). The exact complexity of the FCFS is given in Table 1. Table 2 provides a summary of the computational complexity of all the transforms.

4 Performance Evaluation of PCHT and FCFS

In this section, we first describe how PCHT and FCFS are implemented, and then present results benchmarked on the real VLSI layouts, comparing the performance to that of the traditional discrete transform algorithms. This performance evaluation has two goals. The first is to validate the theoretical computational complexity and analyze the behavior of the runtime as a function of the number of vertices K in a tile. The second goal is to measure the improvement in runtime provided by the PCHT and FCFS over DHT and FFT, respectively. Finally, we perform a simple aerial image simulation using both FCFS and FFT. We compare images produced using a comparable number of operations for both the algorithms.

4.1 *Implementation and Benchmark Setup*

All the algorithms were implemented in the computational lithography tool and run on a 3 GHz Intel© Xeon 5450 running Linux© in 64-bit mode. All the code is C++, single threaded and was compiled using GCC 4.1.2 with option "-O3." The tool takes a layout file as input, parses it, breaks down polygons, and places them in their corresponding tiles. Each tile is then transformed individually. Figure 4 shows the flow diagrams of the different steps involved in the process of transforming a layout using the PCHT, DHT, FCFS, and FFT. The transform is initially performed twice to get the machine into a steady state, and then repeated 10 more times to average out the timing noise. Both PCHT and DHT, as described in Sec. 3.2.1, were fully custom-implemented. Contrary to our previous implementation,[12] here, we include the storage of the transform coefficients. We use the FFTW3 to perform the FFT.[29] We custom-implemented FCFS using the TD-FFT algorithm for pruned FFTs, which in turn use

Table 1 Complexity of the FCFS algorithm. $K = \sum_{m=0}^{M-1} K_m$ is the sum of the vertices of the M polygons in the $N_x \times N_y$ tile. The parameters, $P_x^{(I)}$, $P_x^{(II)}$, dividers of N_x and $P_y^{(I)}$, $P_y^{(II)}$, dividers of N_y, are chosen to minimize the overall complexity.

Step	Number of operations	Operation performed
1	$(3/2)K - M$	Sum of the polygon areas
2	$[(5/2) + (3/2)Q_x^{(I)}] \times K + [(1/2)Q_x^{(I)} + 1] \times C_{\mathrm{FFT1-D}}(P_x^{(I)})^{\mathrm{a}}$	One-dimensional (1-D) $K/2$-input pruned FFT
3	$[(5/2) + (3/2)Q_y^{(I)}] \times K + [(1/2)Q_y^{(I)} + 1] \times C_{\mathrm{FFT1-D}}(P_y^{(I)})^{\mathrm{a}}$	1-D $K/2$-input pruned FFT
4	$[-(17/2) + (19/2)Q_x^{(II)} + (5/2)Q_x^{(II)}Q_y^{(II)}] \times K + Q_x^{(II)}[(1/2)Q_y^{(II)} + 1] \times C_{\mathrm{FFT2-D}}(P_x^{(II)}, P_y^{(II)})^{\mathrm{b}}$	Two-dimensional (2-D) K-input pruned FFT
Total	$[-2 + (3/2)Q_x^{(I)} + (3/2)Q_y^{(I)} + (19/2)Q_x^{(II)} + (5/2)Q_x^{(II)}Q_y^{(II)}]K - M + [(1/2)Q_x^{(I)} + 1] \times C_{\mathrm{FFT1-D}}(P_x^{(I)})$ $+[(1/2)Q_y^{(I)} + 1] \times C_{\mathrm{FFT1-D}}(P_y^{(I)}) + Q_x^{(II)}[(1/2)Q_y^{(II)} + 1] \times C_{\mathrm{FFT2-D}}(P_x^{(II)}, P_y^{(II)})$ operations	

[a]1-D Split-Radix: $C_{\mathrm{FFT1-D}}(N) = 4N \log_2 N - 6N + 8$.
[b]2-D Split-Radix: $C_{\mathrm{FFT2-D}}(N_x, N_y) = 4N_xN_y \log_2 N_xN_y - 12N_xN_y + 8N_x + 8N_y$.

Table 2 Summary of the computational complexity of the continuous and discrete transforms of a single rectilinear polygon in terms of the total number of additions and multiplications.

	Complexity of the transforms			
PCHT	$\approx \{\sum_{i=0}^{M-1} \sum_{j=0}^{J-1} \lceil 2^j P_i/N \rceil [\Lambda_{ia}(K_i) + 26]\} + M$[a]		DHT	$8[(N^2 - 1)/3]$
	Goertzel	TD-FFT		
FCFS	$\{[N_x + (1/4)][N_y + (5/4)] + (19/16)\}K - M$	See Table 1	FFT	$(1/2)(4N_xN_y\log_2 N_xN_y - 12N_xN_y + 8N_x + 8N_y)$[b]

[a]given in Eq. (6).
[b]Row-column real split-radix FFT complexity (Ref. 28).

FFTW3 as FFT kernel. For the discrete transforms, we first created an image of the tile and then fed it to the transform algorithm. The time needed to create the discrete image is added to the runtime of the discrete transform.

For the evaluation, the algorithms were run on three layers from a 22 nm layout of modest size (0.43 mm × 0.33 mm) containing rectangles and more complex rectilinear polygons. These layers are M1 and M2, which contain both rectangles and other polygons, and CA, which contains only rectangles, as shown in Fig. 1. We ran the experiment on squared tiles, where side lengths were powers of two from 128 nm × 128 nm to 4096 nm × 4096 nm. The tests for FCFS and FFT were performed using the sampling at 1 nm on a grid. As part of a tool chain together with a low-pass filter (from a lens), FCFS and FFT could both be sped up from sampling at lower rates. To enable exact and fair comparison of the algorithms, we choose not to add this layer of approximation.

This experiment has two distinct goals. The first is to validate the theoretical complexity derived in Sec. 3 as a predictor of the behavior of the runtime of the transforms. To that effect, we use the runtime from the M1 tiled in 1024 nm × 1024 nm. We compute the average of the 10 runs for each tile and for all transforms. We then take the median of this average over all tiles containing a given number of vertices K. It may happen that the number of tiles with K vertices is quite small (say four tiles), and that the runtime for one of those tiles is disproportionate. Taking the median will mitigate the effect of these specific tiles, considered as outliers in this case. This is shown in Figs. 6 and 7. We plot K on the x-axis, the median runtime on the left y-axis, and the complexity on the right y-axis. The plot also shows in light gray the empirical distribution of K to indicate which range of K is the most important one.

The second is to measure the relative difference of runtime between PCHT and FCFS, and their respective discrete counterparts. To that effect, we aggregate the results to get the average of 10 runtimes over a full layer, for all layers and all transforms. Our metric is the speed-up, computed by dividing the average runtime of the discrete transform by the average runtime of the corresponding continuous transform for a given layer and tile size. This is shown in Figs. 8 and 9. We plot the speed-up of the average runtime for all layers and tile sizes considered.

4.2 *Benchmark Results*

Figure 6 shows the runtime and the complexity of the PCHT and the DHT as a function of K for 1024 nm × 1024 nm tiles from the M1. The left and right y-axis show the runtime and complexity, respectively. We observe that the complexity given in Eq. (7) describes the qualitative behavior of the runtime of the PCHT very well, even for the outliers around $K = 190$. When $K > 200$ and the tiles are composed exclusively of rectangles, Eq. (7) underestimates the complexity. In this case, the runtime is dominated by memory transfers.

The gap between the runtime of the DHT and its complexity is also explained by the domination of memory transfers, which are not accounted for in the computational complexity in Eq. (7). This is in contrast with our previously published results,[12] where storage of the coefficients was not taken into account. The dependence of the DHT on K stems from discrete image creation and from the fact that our implementation uses an if statement to avoid storing zero transform coefficients whose number decreases with

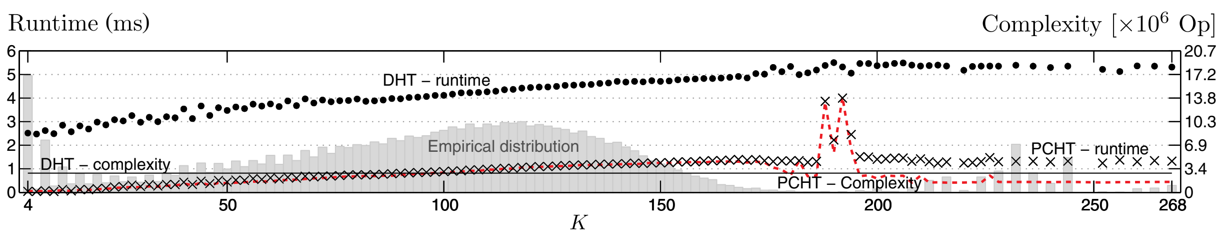

Fig. 6 Median runtime (left y-axis, data points) and complexity (right y-axis, lines) of the PCHT (×, dashed line) and DHT (•, solid line) of 1024 nm × 1024 nm tiles from the M1 layer containing K vertices. Note the accuracy of our complexity estimate in predicting the qualitative behavior of the PCHT (superposition of crosses and dashed line). Even the outliers around $K = 190$ are predicted. Tiles with $K > 200$ have a lower complexity as they contain only rectangles, which are less complex. However, the complexity is underestimated as it does not take memory transfers into account. The empirical distribution of the number of vertices is shown in gray.

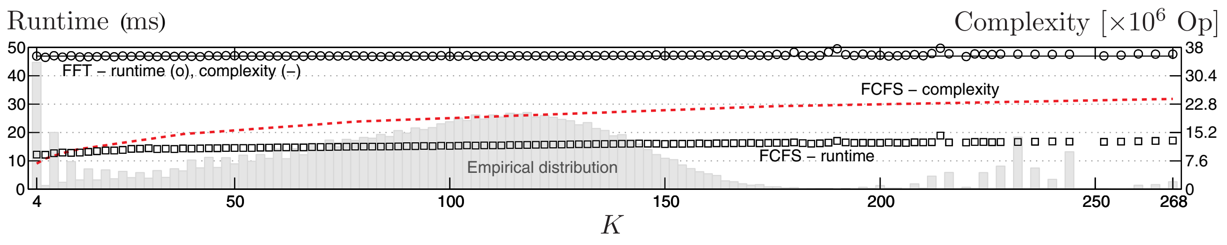

Fig. 7 Median runtime (left y-axis, data points) and complexity (right y-axis, lines) of the FCFS (□, dashed line) and FFT (°, solid line) of 1024 nm × 1024 nm tiles from the M1 layer containing K vertices. We observe that the gap between the runtimes of the FCFS and the FFT is larger than that of their respective complexities. The reason being that the pruned FFT can be implemented more efficiently than the FFT as shown in Ref. 30.

increasing K. Given the very high number of zero coefficients, the cost of the if statement is justified by the large memory transfer savings. Overall, Fig. 6 shows that for small values of K encountered in VLSI layouts (the number of vertices in a given tile of VLSI layout is small relative to the total coordinate space), the PCHT is significantly faster than the DHT.

Figure 8 shows that the average runtime of the PCHT for a full M1 layer is about five times shorter than that of the DHT, for all considered tile sizes. For the CA layer, which contains only rectangles that have a lower complexity, the speed-up is 25-fold for large tiles. The M2 layer shows higher speed-up because it contains less and larger polygons than M1.

Figure 7 shows the runtime and the complexity of the FCFS and the FFT as a function of K for 1024 nm× 1024 nm tiles from the M1. The left and right y-axis show the runtime and complexity, respectively. The runtime of FCFS compared with that of the FFT is lower than the expected given the theoretical complexity. Indeed, as shown by Franchetti and Püschel,[30] the pruned FFT can be implemented more efficiently than the FFT. Here again, the FCFS is significantly faster than the FFT for small values of K found in VLSI layouts.

The speed-up achieved by FCFS over the FFT, shown in Fig. 9, is similar for all considered layers. The M2 layer shows a slightly higher speed-up because its vertex density is lower than that of other layers. For all layers, the highest speed-up found is for 1024 nm × 1024 nm tiles, for which the FCFS is about three times faster than the FFT. In contrast, the results for 128 nm × 128 nm and 256 nm × 256 nm show only a modest speed-up of about 1.5. The peak speed-up at 1024 nm × 1024 nm is most likely explained by a more efficient cache usage at that size. However, further investigation would be required to confirm that.

For all speed-up results, confidence intervals were computed at the 95% level but found to be negligible and have thus been omitted in this figures. However, they can be found in Table 3, which contains average runtime values of all considered transforms, for the M1 layer divided into 1024 nm × 1024 nm tiles, both per tile and for the whole layer. For the 127,544 tiles of M1, the runtime is found to be on average about five times lower for the PCHT and three times lower for the FCFS, respectively, than for their discrete counterparts. Although these numbers might at first glance seem modest, the runtime is reduced from about 9 min for the DHT to only 2 using the PCHT, whereas the FCFS runs in 30 min instead of 1 h 40 min for the FFT. These are significant time savings.

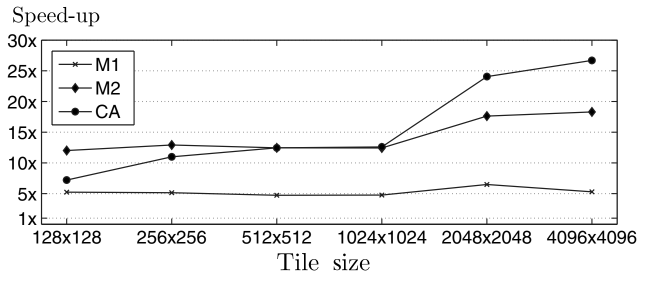

Fig. 8 Speed-up of the average runtime provided by the PCHT on layers M1 (×), M2 (♦), and CA (•). The higher the better. One times speed-up means same performance as the DHT. We see that PCHT is at least five times faster than the DHT. Confidence intervals have been omitted in this figure as in the worst case they were found to be within ±0.06 of the value given, with 95% confidence.

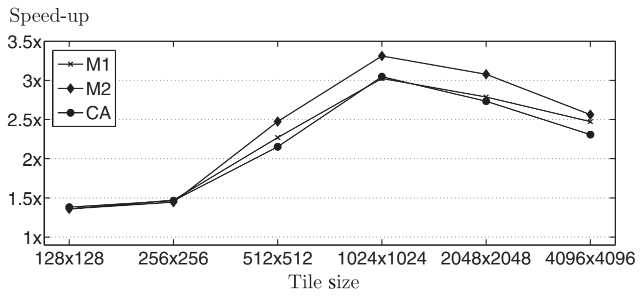

Fig. 9 Speed-up of the average runtime provided by the FCFS on layers M1 (×), M2 (♦), and CA (•). The FFT is sampled on the 1-nm grid. The higher the better. A speed-up of 1× means the same performance as the FFT. We see that FCFS is at least 1.4 times faster than the FFT. Confidence intervals have been omitted in this figure as in the worst case they were found to be within ±0.012 of the value given with 95% confidence.

Table 3 Average runtime of the transforms of an M1 layer divided into 1024 nm × 1024 nm tiles. There are 127,544 nonempty tiles for a total of 995,497 rectangles and 428,817 other polygons. Runtime per tile is in μ-seconds. Total runtime is in seconds.

	Average runtime (with 95% confidence intervals)			
	PCHT	DHT	FCFS	FFT
Per Tile (μs)	919 ± 0.6	4375 ± 12.8	$15{,}648 \pm 10$	$47{,}341 \pm 8$
Total (s)	117.2 ± 0.2	558 ± 4.5	1996 ± 4	6038 ± 3

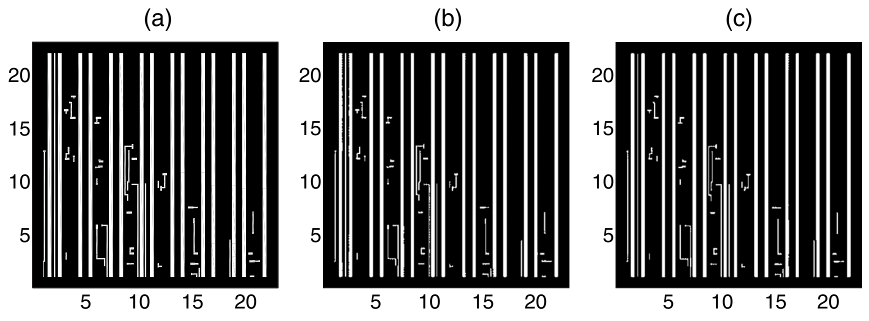

Fig. 10 Aerial image simulation of the 23×23 μm tile shown in (a) using (b) FCFS and (c) FFT methods (light wavelength $\lambda = 193$ nm, numerical aperture NA $= 0.85$, pixel size 57×57 nm, thresholding at 0.7 used to simulate etching). The simulation of (b) uses approximately 21×10^6 operations and does not use any approximation. The simulation of (c) relies on a rasterized image of the polygons and needs roughly 28×10^6 operations.

Simulation runs on different layouts, whose results we omit for brevity, returned similar performance improvements.

4.3 *Aerial Image Simulations*

We simulate aerial image by applying the ideal filter $H(f,g)$ in the frequency domain. The filter $H(f,g) = 1$ if $f^2 + g^2 \leq (\mathrm{NA}/\lambda)$ and 0 otherwise. The parameters NA and λ are the optical aperture and the light source wavelength, respectively. In this simulation, we use $\mathrm{NA} = 0.85$ and $\lambda = 193$ nm. The light field on the wafer is then reconstructed and thresholded at 0.7 to simulate the effect of etching. We consider a 23×23 μm tile containing 50 polygons and an ambient of 1 nm around them. The vertices of the polygons lie on the 1-nm grid. The output images are sampled at 57 nm, that is, in that case, twice the Nyquist rate of the filter. The resulting aerial images can be seen in Fig. 10.

For the FCFS, we use Proposition 1 to directly compute only the frequencies within the pass-band area of the filter $H(f,g)$. The output aerial image is reconstructed by an FFT-based method on a 57-nm grid. This output is thresholded to simulate the effect of etching. Note that there is no approximation used in this computation. The number of operations used to compute the image is roughly 21×10^6.

For the FFT technique, we first raster the polygons into 57×57 nm pixels. The value of one pixel is the ratio of the area of the intersection of the polygon and the pixel to the area of the pixel itself. Note that this is a lossy operation that decreases accuracy. We apply the FFT to the raster image, zero out all spectral coefficients outside the pass-band area of $H(f,g)$, and finally an iFFT is used to compute the output image. Again the output image is thresholded to simulate the effect of etching. Ignoring the rasterization operation, approximately 28×10^6 operations are necessary. This contrasts with the approximately 76×10^9 operations that would be needed if the mask was sampled on the 1-nm grid to avoid losing accuracy through rasterization.

Algorithm 3 TD-FFT 2-D (V, S, L, N_x, N_y).

Require: Vectors V and S containing the nonzero inputs and their locations, respectively, such that $f[S[2i], S[2i+1]] = V[i]$. L, the length of V. N_x and N_y, the FFT lengths. FFT 2-D is an FFT routine.

Ensure: F is an $N_x \times N_y$ matrix containing the $N_x \times N_y$ FFT of the sequence described by V and S.

```
1: for $k_1 = 0$ to $Q_x - 1$ and $l_1 = 0$ to $Q_y - 1$ do
2:     for $i = 0$ to $L - 1$ do
3:         $m \leftarrow S[2i]$, $n \leftarrow S[2i+1]$
4:         $m_2 \leftarrow m \bmod P_x$, $n_2 \leftarrow n \bmod P_y$
5:         $f_{k_1,l_1}[m_2, n_2] \leftarrow f_{k_1,l_1}[m_2, n_2] + V[i] W_{N_x}^{mk_1} W_{N_y}^{nl_1}$
6:     end for
7:     $F_{k_1,l_1} \leftarrow \mathrm{FFT} - 2\mathrm{D}(f_{k_1,l_1}, P_x, P_y)$
8:     for $k_2 = 0$ to $P_x - 1$ and $l_2 = 0$ to $P_y - 1$ do
9:         $F[k_1 + Q_x k_2, l_1 + Q_y l_2] = F_{k_1,l_1}[k_2, l_2]$
10:    end for
11: end for
```

5 Conclusions and Future Work

We developed a framework for computing continuous transforms of rectilinear polygons. We showed how this framework is applied to PCHT and we developed FCFS, two new fast algorithms for the computation of the continuous Haar transform and CFS, respectively, of rectilinear polygons. We showed that the polygon description can be exploited by continuous transforms, resulting in significant speed-up of the transform coefficients computation. The complexity of each of the algorithms was obtained as a function of the number of polygon vertices. We implemented the algorithms and performed extensive runtime measurements on VLSI layouts. The speed-up of PCHT relative to DHT was found to be at least five times for all considered layers and tile sizes. A maximum speed-up of 30 times was achieved in the case of the CA layer divided into 4096 nm $\times$ 4096 nm tiles. FCFS showed least improvement for small tile sizes where it is only 1.5 times faster than the FFT sampled on the 1-nm grid. However, it showed a peak performance for 1024 nm $\times$ 1024 nm tiles where it is over three times faster than the FFT sampled on the 1-nm grid. These speed-ups result in significant time savings due to the magnitude of the problem at hand. We therefore conclude that the PCHT and FCFS are superior to the DHT and FFT, respectively, for rectilinear polygons in VLSI layouts.

We tested thus far the two algorithms on preOPC layouts. Although the accuracy of our algorithms would clearly benefit model-based OPC methods, postOPC layouts typically have a higher vertex density, which slows down PCHT and FCFS. Provisional tests have shown that the speed-up from the continuous methods is significant, albeit slightly reduced. OPC effects are an interesting topic in their own right, and an extensive study will be performed as part of the future work. Also, when using a given optical system which has a low-pass filter with an effective Nyquist rate, carefully down-sampling to reduce complexity is possible. Future work would be to implement and test this combined with FFT and FCFS.

While the Fourier transform is firmly embedded in the computational lithography, the Haar transform has yet to have any major applications. The next step will be to integrate both algorithms in a practical VLSI tool chain and measure the impact, both in terms of speed-up and accuracy. Another important work would be to generalize the algorithm to work with more arbitrary polygons, such as those with 45 deg edges. This can likely be done following a similar approach to that of Sec. 3.1.

In a final toolset implementation, the methods described require careful tuning as one would expect, and, given the trend toward multicore architectures, a parallel implementation would be natural. Fortunately, both PCHT and FCFS are designed with highly suitable structures for parallelization. In addition to the mere implementation, the integration into existing tools and tuning to other processes (e.g., resist, etching) is likely to be a major challenge.

Appendix A: Inner Product with Rectilinear Polygons

In this appendix, we prove Lemma 3.1. First, we define a rectangle $\mathcal{R}$ by its lower left and upper right vertices $\mathcal{R}_{(x_1,y_1)}^{(x_2,y_2)} = [(x,y)|x_1 \le x < x_2, y_1 \le y < y_2]$. As illustrated in Fig. 2, it is possible to construct any rectilinear polygon as union and difference of rectangles.

Proposition 2. Any rectilinear polygon $\mathcal{P}$ with K vertices can be expressed by $K/2$ disjoint rectangles, each with a side lying on the x-axis

$$\mathcal{P} = \bigcup_{\{i|x_{i+1}>x_i\}} \mathcal{R}_{(x_i,0)}^{(x_{i+1},y_i)} \setminus \bigcap_{\{i|x_{i+1}\}} \mathcal{R}_{(x_{i+1},0)}^{(x_i,y_i)}, \tag{12}$$

where (x_i, y_i) is the i'th vertex, the indices are taken modulo K and \ is the set difference operator. An equivalent result exists for the case where a side is lying on the y-axis. The next step is to compute the inner product of Eq. (3) on a tile containing disjoint rectilinear polygons such as Eq. (4).

Proposition 3. If $f_T(x,y)$ is given by Eq. (4) and $\phi_{k,l}(x,y)$ is a basis function, their inner product according to Eq. (3) is

$$\langle f_T, \phi_{k,l}\rangle = \sum_{m=0}^{M-1} \sum_{i=0}^{K_m/2-1} \int_0^{y_{m,2i}} \int_{x_{m,2i}}^{x_{m,2i+1}} \phi_{k,l}^*(x,y)\mathrm{d}x\mathrm{d}y,$$

where $(x_{m,i}, y_{m,i})$ is the i'th vertex, out of the K_m, of the m'th polygon and i is taken modulo K_m. Here, it is assumed without loss of generality that for all $mx_{m,0} \neq x_{m,1}$.

Proof. By the linearity of the inner product and since the polygons are disjoint, we have

$$\langle f_T, \phi_{k,l}\rangle = \sum_{m=0}^{M-1} \langle f_{\mathcal{P}_m}, \phi_{k,l}\rangle = \sum_{m=0}^{M-1} \iint_{\mathcal{P}_m} \phi_{k,l}^*(x,y)\mathrm{d}x\mathrm{d}y. \tag{13}$$

Then, using Proposition 2, we split the integral over $\mathcal{P}_m$ into $K_m/2$ integrals over rectangles. If $x_{m,i} < x_{m,i+1}$, the integral is positive and is added to the final result. If $x_{m,i} > x_{m,i+1}$, the integral is negative and is subtracted from the final result. If $x_{m,i} = x_{m,i+1}$, the i'th term of the sum is zero.

Finally, the proof of Lemma 3.1 follows from Proposition 3, the separability of $\phi_{k,l}$ and the fact that $\sum_{i=0}^{K/2-1} [\Omega_k^*(x_{2i+1}) - \Omega_k^*(x_{2i})] = 0$ for rectilinear polygons since the index i is cyclic and vertical edges cancel. □

Appendix B: 2-D Transform Decomposition Fast Fourier Transform

We present briefly here the 2-D TD-FFT algorithm for a subset of input points. This algorithm is a direct extension of the 1-D TD-FFT found in the literature[26] to the 2-D FFT.

Let $f[m,n] \in \mathbb{C}$ be a complex 2-D sequence with $m = 0,\dots,N_x - 1$ and $n = 0,\dots,N_y - 1$. The 2-D FFT of this sequence is

$$F[k,l] = \sum_{m=0}^{N_x-1} \sum_{n=0}^{N_y-1} f[m,n] W_{N_x}^{km} W_{N_y}^{\ln}, \tag{14}$$

where $W_N = e^{-j\frac{2\pi}{N}}$, $k = 0,\dots,N_x - 1$ and $l = 0,\dots,N_y - 1$. Assume that N_x and N_y are the composite and there exist P_x, Q_x, P_y, and Q_y such that $N_x = P_x Q_x$ and $N_y = P_y Q_y$. Define the following variable substitutions: $m = P_x m_1 + m_2$, $k = k_1 + Q_x k_2$ with m_1, $k_1 = 0,\dots,Q_x - 1$ and m_2, $k_2 = 0,\dots,P_x - 1$, and similarly $n = P_y n_1 + n_2$, $l = l_1 + Q_y l_2$ with n_1, $l_1 = 0,\dots,Q_y - 1$ and n_2, $l_2 = 0\dots,P_y - 1$. Substituting m, k, n, and l in Eq. (14) and rearranging the sums we obtain

$$F_{k_1,l_1}[k_2,l_2] = \sum_{m_2=0}^{P_x-1} \sum_{n_2=0}^{P_y-1} f_{k_1,l_1}[m_2,n_2] W_{P_x}^{k_2 m_2} W_{P_y}^{l_2 n_2}, \tag{15}$$

which is a $P_x \times P_y$ 2-D FFT, where

$$f_{k_1,l_1}[m_2,n_2] = \sum_{m_1=0}^{Q_x-1} \sum_{n_1=0}^{Q_y-1} f[P_x m_1 + m_2, P_y n_1 + n_2] W_{N_x}^{k_1(P_x m_1 + m_2)} W_{N_y}^{l_1(P_y n_1 + n_2)} \tag{16}$$

and

$$F_{k_1,l_1}[k_2,l_2] = F[k_1 + Q_x k_2, l_1 + Q_y l_2] = F[k,l].$$

When only a small subset of the input is nonzero, a significant reduction in complexity can be achieved by computing Eq. (16) directly, and then using a 2-D FFT for Eq. (15). The reduction is due to very few terms being nonzero in the sum in Eq. (16). By carefully choosing P_x and P_y, the execution time can be minimized. In the original paper,[26] the number of operations is minimized by optimizing analytically the computational complexity. This approach does not work well for minimizing execution time on modern computer architectures due to the cost of memory transfers not being accounted for in the computational complexity.[29] Instead, we experimentally found the optimal P_x and P_y by measuring the execution time for a large range of input and FFT sizes using the different decomposition factors. The algorithm is described in Algorithm 3. The inputs to the algorithm are a sparse array containing the nonzero coefficients, not necessarily in order, along with the FFT lengths N_x and N_y.

If the input is real-valued, the computational complexity of the algorithm can be roughly halved by performing only the computations for $l_1 = 0, \dots, \lfloor Q_y/2 \rfloor + 1$ (or k_1 conversely) and using the conjugate symmetry to compute the missing outputs.

Acknowledgments

The authors thank Markus Püschel for providing useful advice on the FFT and pruned FFT and Paolo Ienne for kindly reviewing our paper and providing invaluable feedback.

References

1. C. Mack, *Fundamental Principles of Optical Lithography: The Science of Microfabrication*, Wiley, Chichester (2008).
2. A. K. K. Wong, *Resolution Enhancement Techniques in Optical Lithography*, SPIE Press, Bellingham, Washington (2001).
3. A. E. Rosenbluth et al., "Optimum mask and source patterns to print a given shape," *Proc. SPIE* **4346**, 486–502 (2001).
4. A. Poonawala, P. Milanfar, and D. G. Flagello, "OPC and PSM design using inverse lithography: a nonlinear optimization approach," *Proc. SPIE* **6154**, 61543H (2006).
5. F. M. Schellenberg and B. W. Smith, "Resolution enhancement technology: the past, the present, and extensions for the future," *Proc. SPIE* **5377**, 1–20 (2004).
6. International Technology Roadmap for Semiconductors, "2007 edition," http://www.itrs.net/Links/2007ITRS/2007_Chapters/2007_Lithography.pdf (2007).
7. J. W. Goodman, *Introduction to Fourier Optics*, 3rd ed., Roberts & Company, Greenwood Village (2004).
8. R. Nasser and P. Hurley, "On the accuracy of different Fourier transforms of VLSI designs," *Proc. SPIE* **8683**, 868319 (2013).
9. K. Kryszczuk, P. Hurley, and R. Sayah, "Direct printability prediction in VLSI using features from orthogonal transforms," in *Proc. of the IAPR Int. Conf. on Pattern Recognition*, pp. 2764–2767, IEEE, Istanbul, Turkey (2010).
10. M. E. Haslam et al., "Two-dimensional Haar thinning for data base compaction in Fourier proximity correction for electron beam lithography," *J. Vac. Sci. Technol. B: Microelectron. Nanometer Struct.–Process. Meas. Phenom.* **3**(1), 165–173 (1985).
11. X. Ma and G. R. Arce, "Generalized inverse lithography methods for phase-shifting mask design," *Opt. Express* **15**(23), 15066–15079 (2007).
12. R. Scheibler, P. Hurley, and A. Chebira, "Pruned continuous Haar transform of 2D polygonal patterns with application to VLSI layouts," in *Proc. of the 2010 IRAST Int. Cong. on Comp. App. and Computational Science (CACS 2010)* pp. 984–987, IRAST, Singapore (2010).
13. N. B. Cobb, "Fast optical and process proximity correction algorithms for integrated circuit manufacturing," PhD Thesis, University of California, AAI9902038 (1998).
14. D. G. L. Chow et al., "An image processing approach to fast, efficient proximity correction for electron beam lithography," *J. Vac. Sci. Technol. B: Microelectron. Nanometer Struct.–Process. Meas. Phenom.* **1**(4), 1383–1390 (1983).
15. J. F. Chen et al., "Development of a computational lithography roadmap," *Proc. SPIE* **6924**, 69241C (2008).
16. S. W. Lee and R. Mittra, "Fourier transform of a polygonal shape function and its application in electromagnetics," *IEEE Trans. Antennas Propag.* **31**(1), 99–103 (1983).
17. F. L. Chu and C. F. Huang, "On the calculation of the Fourier transform of a polygonal shape function," *J. Phys. A* **22**(14), L671–L672 (1989).
18. L. Brandolini, L. Colzani, and G. Travaglini, "Average decay of Fourier transforms and integer points in polyhedra," *Ark. Mat.* **35**(2), 253–275 (1997).
19. Y. M. Lu, M. N. Do, and R. S. Laugesen, "A computable Fourier condition generating alias-free sampling lattices," *IEEE Trans. Signal Process.* **57**(5), 1768–1782 (2009).
20. M. Vetterli, J. Kovačević, and V. K. Goyal, "The world of Fourier and wavelets: theory, algorithms and applications," http://www.fourierandwavelets.org/ (2009).
21. S. Mallat, *A Wavelet Tour of Signal Processing: The Sparse Way*, 3rd ed., Academic Press, Burlington (2008).
22. N. U. Ahmed and K. R. Rao, *Orthogonal Transforms for Digital Signal Processing*, Springer-Verlag, New York (1975).
23. F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, New York (1985).
24. J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.* **19**(90), 297–301 (1965).
25. G. Goertzel, "An algorithm for the evaluation of finite trigonometric series," *Am. Math. Mon.* **65**(1), 34–35 (1958).
26. H. Sorensen and C. Burrus, "Efficient computation of the DFT with only a subset of input or output points," *IEEE Trans. Signal Process.* **41**(3), 1184–1200 (1993).
27. M. Medina-Melendrez, M. Arias-Estrada, and A. Castro, "Input and/or output pruning of composite length FFTs using a DIF-DIT transform decomposition," *IEEE Trans. Signal Process.* **57**(10), 4124–4128 (2009).
28. P. Duhamel and H. Hollmann, "Split radix FFT algorithm," *Electron. Lett.* **20**(1), 14–16 (1984).
29. M. Frigo and S. Johnson, "The design and implementation of FFTW3," *Proc. IEEE* **93**(2), 216–231 (2005).
30. F. Franchetti and M. Püschel, "Generating high performance pruned FFT implementations," in *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, pp. 549–552, IEEE, Taipei (2009).

Robin Scheibler is a PhD student at the Laboratory for Audiovisual Communications (LCAV), École Polytechnique Fédérale de Lausanne (EPFL). He holds the BS and MS in communication systems from EPFL. After graduating in 2009, he worked as a research engineer at IBM Research Zürich until 2010, and then at NEC Research Laboratories, Kawasaki, until 2012.

Paul Hurley is a research staff member at IBM Research in Zurich, Switzerland. He holds a PhD in communication systems from École Polytechnique Fédérale de Lausanne (EPFL), and BSc in mathematical and computer science from NUI Galway, Ireland. He has worked as a design engineer at Videologic and DEC. His research interests include signal processing in lithographic systems, information and communications theory, and computer networking.

Amina Chebira is a senior research and development engineer at the Swiss Center for Electronics and Microtechnology (CSEM), Switzerland. She holds a BS in mathematics (Paris 7) and the BS and MS in communication systems from École Polytechnique Fédérale de Lausanne. In 2008, she obtained the PhD degree and the research award from the Biomedical Engineering Department, Carnegie Mellon University. She then held a postdoctoral position at LCAV, EPFL until 2012. Her research interests include frame theory and design, biomedical signal and image processing, pattern recognition, and filterbanks.