CAPLET: A Highly Parallelized Field Solver for Capacitance Extraction Using Instantiable Basis Functions

Yu-Chung Hsiao and Luca Daniel

***Abstract*—Parallelization of traditional accelerated techniques for integral equation solvers has been shown to be inefficient and to scale poorly with the number of parallel computing nodes. This is because traditional methods typically represent the solution using piecewise constant basis functions, resulting in gigantic systems of linear equations to solve. In this work, we propose instantiable basis functions, which generate smaller systems than piecewise constant basis functions for the same accuracy. Furthermore, they redistribute computation from the system solving step to the embarrassingly parallelizable system setup step, hence enabling highly scalable and efficient parallelization. In the examples we tested, our new solver is six to ten times faster than FASTCAP in serial execution and achieves 90% parallel efficiency in a 10-core distributed-memory system. We developed a toolkit that automates a complete extraction flow from GDSII layout files to capacitance matrices. Our code has been released in the public domain.**

***Index Terms*—boundary element method, basis functions, capacitance extraction, VLSI interconnects, parallel computing, field solver.**

I. Introduction

Accurate and efficient capacitance extraction is critical for high-performance integrated circuit (IC) design and cell optimization. State-of-the-art 2D scanning methods for IC interconnects, or pattern-matching approaches, usually involve two steps: i) decomposing 3D layouts into 2D cross-sections and ii) matching the 2D cross-sectional geometries to pre-computed table entries or approximation formulae [1]. These procedures are illustrated in Fig. 1. Although such 2D scanning methods are fast and accurate enough (i.e., within 5% errors) for the majority of layout geometries, there are certain 3D geometries which suffer from significant errors (i.e., $> 20\%$ errors), such as comb capacitors and crossing wires in adjacent layers. Those 3D geometries, referred to as full 3D structures in this paper, consist of tens of conductors, and are scattered all over an IC layout.

Fast capacitance extraction for full 3D structures becomes even more critical in standard cell optimization. During optimization, the geometry of a cell is sequentially modified at each search step based on the previous extraction solutions. Therefore, the parallelization has to be done within each individual extraction problem. To this aim, we propose an efficiently parallelized capacitance extraction algorithm that targets small- to medium-sized problems, provides field-solver accurate results, performs ultra-fast extraction, and scales almost linearly with the number of parallel computing nodes. In addition, to facilitate benchmarking algorithms on realistic interconnect structures, we developed a toolkit that automates a complete extraction flow from GDSII layout files to capacitance matrices. Our code is released in the public domain [2].

The traditional strategy for extracting more accurate capacitance is to use 3D electrostatic field solvers for full 3D structures. Two mainstream approaches have been proposed: deterministic and stochastic methods. In the category of deterministic methods, boundary element methods (BEM) [3]–[8] are of particular interest because the charge distributions can be efficiently solved without discretizing the interior of dielectric materials. Such BEMs usually involve a two-step process (Fig. 2): (i) set up a linear system (the system setup step), and (ii) solve the linear system (the system solving step). Each element of the system matrix is the coupling coefficient between a pair of basis functions. The overall time complexity for filling up the system matrix is $O(N^2)$. The resulting dense system can be solved using Gaussian elimination with $O(N^3)$ complexity or standard iterative methods with $O(N^2)$ complexity.

Efficient algorithms based on floating random walks have also been proposed [9], [10]. In contrast to the BEMs, such stochastic approaches do not rely on formulating systems of linear equations, and hence, they require less memory for the same size of structures. Besides, floating random walk methods are more scalable for large problems and can be parallelized more easily. Recent advances can be found in [11]–[14]. However, these floating random walk approaches are efficient only when the structure is sufficiently large, as demonstrated in [6]. In this paper, since we are interested in small- to medium-sized problems, the following discussions are exclusively focused on the development of boundary element methods.

Several acceleration methods were proposed in the past decades for improving the computational complexity of BEMs [3]–[7]. Those works typically focused on reducing the asymptotic time complexity of the system solving step from $O(N^2)$ to $O(N \log N)$, or even to $O(N)$, and are most efficient when the charge distributions are represented by piecewise constant (PWC) basis functions. However, the improvement on asymptotic complexity may not directly reflect the actual improvement in runtime. This is because the

The authors are with the Research Laboratory of Electronics, Massachusetts Institute of Technology, MA 02139, USA (e-mail: yuchsiao@mit.edu, luca@mit.edu).

This works is funded by Semiconductor Research Corporation, Mentor Graphics, and Advanced Micro Devices.

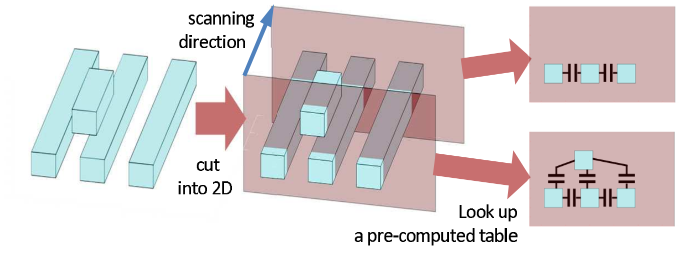


Fig. 1: 2D scanning and table lookup extraction method. The quasi-3D capacitance is built based on the 2D capacitance on each cross-section.

above acceleration methods commonly suffer from initialization overheads. Such overheads are mostly negligible for structures that include hundreds of wires but appear significant for smaller full 3D structures.

In addition to the limited speedup for smaller structures, the aforementioned Krylov-subspace acceleration methods are also not embarrassingly parallelizable. The computation of an efficient parallelizable algorithm has to be "divisible" with (i) low data dependency between subproblems, (ii) minimal memory accesses, and (iii) small amount of data transfer between parallel computing nodes. However, Krylov-subspace acceleration methods, such as [3] and [4], need to exchange large residual vectors between parallel computing nodes or they need access the shared memory pool, which incurs extra overheads for maintaining data concurrency. In addition, the major parallelization bottleneck for such Krylov-based methods occurs in their specialized matrix-vector product approximations, such as fast multipole expansion [3], and fast Fourier transform in the pre-corrected FFT algorithm [4]. These approximations reduce the time complexity of the matrix-vector products from $O(N^2)$ to $O(N \log N)$, or close to $O(N)$, but at the same time, they largely increase data dependency. Some previous works, such as [15]–[17], have shown that parallelizing such matrix-vector product approximations is inefficient. The parallel speedup saturates very quickly with the number of parallel computing nodes, and its parallel efficiency drops to 40%–60% at eight nodes for their best available results in [15] and [16]. Therefore, one can conclude that the above acceleration methods [3]–[5] are best suited for single-core serial execution.

These two major drawbacks of the existing acceleration methods are rooted in their choices of basis functions. The PWC basis functions are one of the most prevalent choices for the electrostatic boundary element method (BEM) thanks to the availability of their closed-form integrals for both collocation and Galerkin testing, and also for their versatility in representing various charge distributions. However, such versatility does come at the cost of a large number of basis functions, which in turn results in difficulties in parallelization and acceleration.

To reduce the system size while maintaining the same level of field-solver accuracy, we introduce a more compact representation for charge distribution: instantiable basis functions (INS), which was first presented in [18] and was inspired by the conduction mode basis functions in [19]–[23]. Instantiable basis functions exploit the fact that the regularity of Manhattan geometries in IC layouts only produces charge distribution in limited shapes. Such charge shapes are highly correlated with the neighboring conductor geometries and can be predicted simply by inspecting the geometries. Our experiments show that, by combining *only two* basic shapes, arch shapes and flat shapes, we can achieve the target 5% capacitance error. Another important feature of our approach is that such basic shapes are *reusable*: they only need to be extracted *once* per technology node and stored for later use in other extraction problems using the same IC technology. Because both the charge shape and the geometry regularity are considered, the resulting instantiable basis functions can generate much more compact systems than those generated by the general-purpose PWC basis functions.

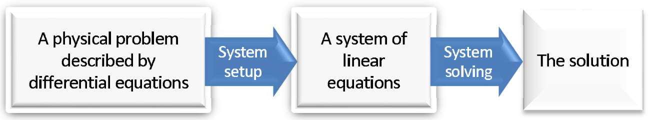

A physical problem described by differential equations
System setup
A system of linear equations
System solving
The solution

Fig. 2: A general two-step process for solving differential equtions.

The compactness of instantiable basis functions in representing charge solutions not only improves the extraction runtime in serial execution, but also facilitates efficient parallelization [24]. This is because using fewer basis functions significantly reduces the system solving time due to its cubic complexity $O(N^3)$ at the cost of involving more expensive integration schemes. Such two effects are compounded and effectively redistribute the computation from the system solving step to the system setup step. While the system solving part is difficult to parallelize with good efficiency, the system setup part is embarrassingly parallelizable and takes the majority of the overall computation after adopting instantiable basis functions. As a result, the parallelization is highly efficient and scales almost linearly in our examples.

We organize this paper as follows. Section II describes the background of the electrostatic boundary element method and basis functions. Section III explains the fundamental ideas of instantiable basis functions. In Section IV, we demonstrate the instantiation procedures of our basis functions, followed by the parallelization strategy in Section V. The implementation of our open-source toolkit is discussed in Section VI. We compare the performance of this work, CAPLET, with FASTCAP in Section VII. Section VIII includes a few concluding remarks to summarize this work.

II. Background

This section briefly reviews the formulation of the Boundary Element Method (BEM) for the electrostatic problem. Given an n-conductor interconnect geometry embedded in a uniform material with a dielectric constant ε, the capacitance matrix can be obtained by solving the integral form of the electric static problem

$$\int_{S'} \frac{\rho(\mathbf{r}')}{4\pi\varepsilon\|\mathbf{r}-\mathbf{r}'\|} \mathrm{d}s' = \phi(\mathbf{r}) \tag{1}$$

for charge distribution $\rho(\mathbf{r}')$. In equation (1), $\mathbf{r}$ and $\mathbf{r}' \in \mathbb{R}^3$ are position vectors in 3D space, $\phi(\mathbf{r}) : \mathbb{R}^3 \mapsto \mathbb{R}$ is the electric

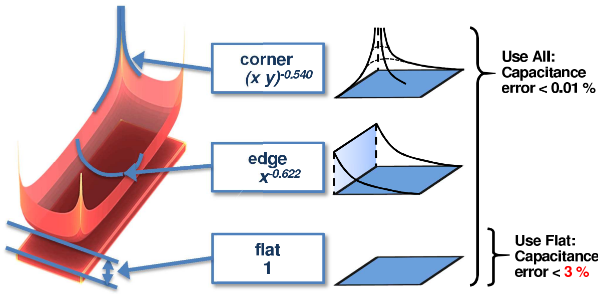

corner
$(x\ y)^{-0.540}$
edge
$x^{-0.622}$
flat
1
Use All: Capacitance error < 0.01 %
Use Flat: Capacitance error < 3 %

Fig. 3: Left: Charge singularities of a 3D straight wire, colored and overlaid with a charge density profile for the top surface. Basis functions and their corresponding expressions are labeled in the figure. Right: Capacitance errors when i) using all three types of basis functions (top) and ii) using only the flat basis functions (bottom).

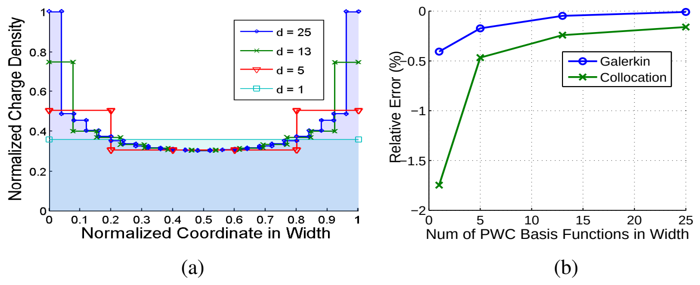


Fig. 4: (a) Charge distribution at the center of a straight wire's top surface (a cross-sectional cut of the profile in Fig. 3) for different uniform discretizations in width. Wire dimension of (length, width, height) is (2, 1, 0.2) in um. The shaded areas are the total charges of the case of 25 discretization panels (in purple) and the single-panel case (in cyan). (b) Capacitance errors for different discretizations *only* in the horizontal direction of the wire in (a) using Galerkin and collocation testing. Galerkin testing is more accurate than collocation testing, especiallly when the discretization number is small. Negative errors indicate that the capacitances are underestimated.

potential, $\rho(\mathbf{r}') : \mathbb{R}^3 \mapsto \mathbb{R}$ is the unknown charge distribution, S' is the conductor surfaces, and the operator $\|\cdot\|$ computes the Euclidean distance of its argument. By approximating $\rho(\mathbf{r}')$ in (1) as $\sum_{j=1}^{N} \rho_j \psi_j(\mathbf{r}')$, where ρ_j are the coefficients to be determined and $\psi_j(\mathbf{r}')$ are basis functions, and projecting the equation residue of (1) onto the null space of a set of testing functions $\{\tilde{\psi}_i \mid i = 1, 2, \ldots, N\}$, one can generate N equations for N unknowns ρ_j

$$\sum_{j=1}^{N} \left[\int_{s_i} \int_{s'_j} \frac{\tilde{\psi}_i(\mathbf{r})\psi_j(\mathbf{r}')}{4\pi\varepsilon\|\mathbf{r}-\mathbf{r}'\|} \mathrm{d}s' \mathrm{d}s \right] \rho_j = \int_{s_i} \tilde{\psi}_i(\mathbf{r})\phi(\mathbf{r})\mathrm{d}s, \quad (2)$$

where s_i and s'_j are the supports of $\tilde{\psi}_i$ and ψ_j, respectively. When $\tilde{\psi}_i = \psi_i$ for $i = 1, 2, \ldots, N$, it is referred to as Galerkin testing. When the testing basis functions are Dirac delta functions $\tilde{\psi}_i(\mathbf{r}) = \delta(\mathbf{r} - \mathbf{r}_i)$ for $i = 1, 2, \ldots, N$ and $\mathbf{r}_i$ is the centroid of the support of $\psi_j(\mathbf{r}')$, it is called centroid collocation testing. Equation (2) can be further represented as a system of linear equations $P\bar{\rho} = \bar{\Phi}$, where $\bar{\rho} = [\rho_1, \rho_2, \ldots, \rho_N]^T$, P is an $N \times N$ system matrix of the bracketed factors in (2), later referred to as the "coupling coefficients" between testing and basis functions, and the right-hand side $\bar{\Phi}$ is an excitation vector of presumed electric potentials for each basis function.

To compute the pairwise capacitance between n conductors, it is sufficient to set up n linearly independent excitation vectors $\bar{\Phi}_k$ and solve for the corresponding solution $\bar{\rho}_k$, $k = 1, \ldots, n$. Combining such n columns forms an expanded system

$$P\rho = \Phi, \quad (3)$$

with $\rho = [\bar{\rho}_1, \bar{\rho}_2, \ldots, \bar{\rho}_n]$ and $\Phi = [\bar{\Phi}_1, \bar{\Phi}_2, \ldots, \bar{\Phi}_n]$. It is convenient to set up the k-th column of Φ in (3) with the excitation

$$\phi(\mathbf{r}) = \begin{cases} 1 & \text{, if } s_j \text{ belongs to the } k\text{-th conductor} \\ 0 & \text{, otherwise.} \end{cases} \quad (4)$$

The capacitance matrix C can then be computed by $C = \Phi^T \rho$, where ρ is the solution to (3).

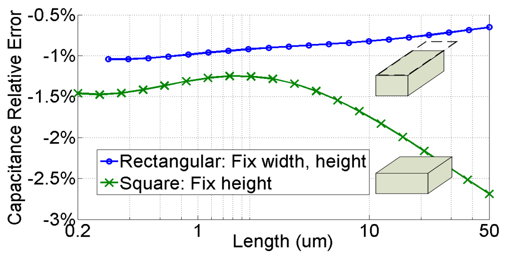


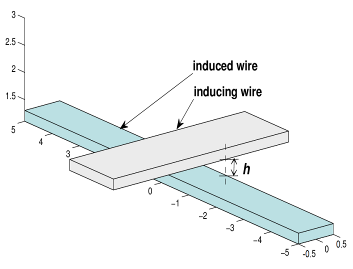
Fig. 5: Capacitance errors for straight wires (top) and square pads (bottom) compared to reference values when using flat basis functions only solved by Galerkin testing. The horizontal axis indicates the length of straight wires (0.3×0.2 um^2 in the cross section) and the size of square pads (0.2 um in thickness). Negative errors indicate that the capacitances are underestimated.

III. Observations

Contemporary lithography technologies typically introduce more stringent design rules every generation. The Manhattan geometry, which only allows right-angle shapes, has been a common layout rule for the technologies at tens of nanometers and beyond. We exploit such geometry regularity and make two major observations that serve as the foundation for the basis function extraction in Section IV.

A. *Effect of charge singularities on capacitance*

Charge singularities correspond to the significant charge accumulation around an edge or a corner of a charged conductor. An example is illustrated in Fig. 3, in which a simple 3D straight wire is overlaid with a charge distribution profile on the top surface. Both the color and the height of the overlaid profile indicate the relative charge density: lighter



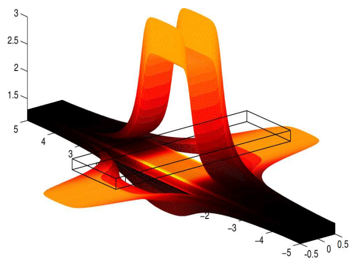


Fig. 6: (a) A pair of crossing wires at different metal layers separated by h. (b) Induced charge distribution solved by the standard BEM using PWC basis functions with self-capacitance removed. Units are um.

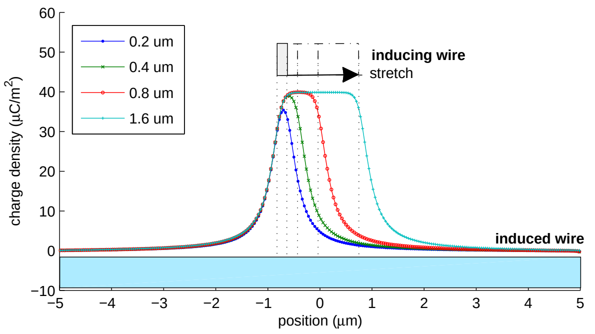


Fig. 7: Induced charge distribution versus the width w of the inducing wire. The series of the curves demonstrates the stretchability of induced charge. The center flat part is stretched out whereas the shapes of the decaying arches on both sides are preserved and only shifted outwards.

color and larger height for higher density. Charge singularities are closely related to the electrostatic discharge (ESD) problem, which is of great importance in various fields, such as interconnection wires [25], MEMS [26], semiconductor devices [27], and electronic packages [28].

Although the local profile of three-dimensional charge singularities around edges and corners has been well studied in the literature, for instance, [29]–[31], the studies on their "aggregate" effect, or capacitance, are relatively sparse. It is commonly believed that charge singularities are a major source of total capacitance and cannot be neglected arbitrarily [25].

However, removing the basis functions dedicated to singularities is *not* the same as neglecting singularities. The former involves the boundary element method that seeks optimal solutions via projection *after* simplifying basis functions, whereas the latter directly removes singularities from the solution, which may result in significant errors. Our experiment in Fig. 4(a) shows a set of cross-sectional charge distributions at the middle of a straight wire in Fig. 3 with the number of discretization elements d, solved by Galerkin testing. The capacitance errors compared against a reference value over different discretizations are shown in Fig. 4(b). Note that both Galerkin and collocation testing exhibit a compensation effect, with an error no worse than 2% in this example. For the same structure, on the other hand, the error due to a direct singularity removal is about 30%. Such a singularity removal is performed by leveling out any accumulation toward edges or corners, for instance, removing any charge more than 0.3 with $d = 25$ in Fig. 4(a). This large error is consistent with the observation in [25]. In addition, it is also worth noting that Galerkin testing is relatively less sensitive to the discretization. The largest error difference of Galerkin and collocation testing occurs at the coarsest case $d = 1$.

This observation indicates that singularity-specific basis functions may only be needed when the accuracy requirement is stringent. As shown in Fig. 3, the fundamental shapes of charge distribution for a straight wire can be recognized as corner, edge, and flat shapes for each flat surface. The flat shape is simply a constant-valued function over its domain. The edge and the corner basis functions can be numerically extracted using finely discretized PWC basis functions. If such three types of basis functions (flat, edge, and corner) are used all together, an extremely accurate result, less than a 0.01% error, can be achieved. Such accuracy is practically not necessary because a typical standard error of capacitance due to process variation may easily exceed 5%; see, for example, Table 1 in [32].

On the other hand, using the flat shape alone with Galerkin testing for the straight wire case can still achieve errors less than 3%. This result is generally true regardless of the actual wire dimensions. Fig. 5 sweeps over practical sizes from 200 nm to 50 um of wires and square pads at a fixed thickness 100 nm. The sweep stops at 50×50 um^2 because it is the size of an external DC pad, which is typically the largest possible metal dimensions in a layout. It is also possible to sweep further into tens of nanometer toward the left, which exhibits no particularly interesting trend: errors saturate around -1% and -1.5% for wires and pads, respectively. Note that these results can be scaled to other thicknesses because of the linear nature of the electrostatic problem.

Based on the aforementioned observations, we make the following empirical statement:

Observation 1. Singularity-specific basis functions are negligible for the straight wire cases with capacitance errors up to 3% using Galerkin testing described in Section II.

B. *Stretchability of induced charge distribution*

Induced charge distribution of a conductor, in our definition, is a separable part of the charge distribution due to the presence of nearby conductors with different electric potentials. We are interested in knowing how induced charge distribution behaves in response to the geometric transformation of nearby conductors. Fig. 6(a) visualizes a pair of wires that cross each other in the middle at different metal layers. The induced charge distribution of the bottom wire is specifically displayed in Fig. 6(b) with self-capacitance removed. Such removal is performed in two steps. First, extract both the charge distributions of (i) the pair of crossing wires in Fig. 6(a) with the bottom wire at 1 V and the top wire at 0 V as ρ_{total} and (ii) the bottom wire alone at 1 V as ρ_{self} (without the presence of the top wire). Second, scale ρ_{self} by γ such that the corner

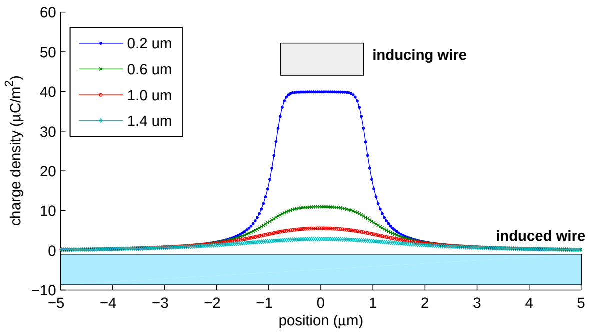


Fig. 8: Induced charge distribution versus separation h.

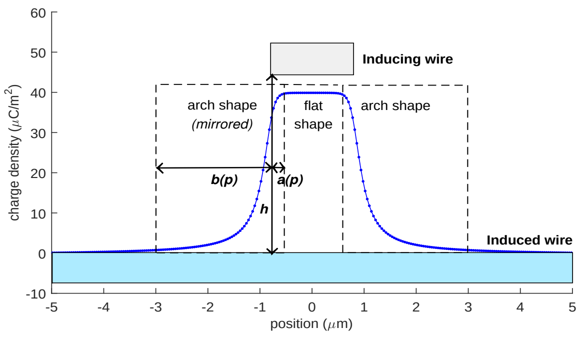


Fig. 9: Induced charge decomposition suggested by Observation 2. The parameter h denotes the separation distance between two wires, $a(\mathbf{p})$ denotes the ingrowth length, and $b(\mathbf{p})$ denotes the extension length. The vector of geometric parameters $\mathbf{p}$ consists of h and other geometric information.

singularities of ρ_{total} and ρ_{self} are aligned. Then the induced charge with self-capacitance removed can be computed by $\rho_{\text{ind}} = \rho_{\text{total}} - \gamma\rho_{\text{self}}$. Fig. 7 shows a set of the induced charge distributions along the central axis of the top surface of the induced wire in response to different widths of the inducing wire. It can be seen that for different widths of the inducing wire, only the center plateaus stretch or shrink, while both the shapes of the decaying side slopes are kept nearly unchanged. Another strong dependency can be identified from Fig. 8, which shows that the induced charge distributions strongly depend on the vertical wire separation h. Accordingly, we make the following observation and will term the decaying side slopes as “arch” shapes in the rest of this work.

Observation 2. Given a pair of crossing wires vertically separated by h as shown in Fig. 6(a), the arch shapes, illustrated in Fig. 9, are insensitive to the width of the inducing wire. In addition, the arch shapes strongly depends on the wire separation h as shown in Fig. 8, and are much less sensitive to the rest of other geometric parameters.

Observation 2 suggests that an induced charge distribution is decomposable, as illustrated in Fig. 9. In the following section, we will further extend and apply Observation 1 to a more general setting, considering multiple conductors in the Manhattan geometry, and construct instantiable basis functions by combining flat and arch shapes on the fly during extraction, according to Observation 2.

IV. Instantiable Basis Functions

In this section, we would like to establish the synthesis procedure for basis functions from the two fundamental flat and arch shapes that we observed in Fig. 9. The synthesis, or instantiation procedure, involves two levels: from 1D shapes as in Fig. 9 to 2D rectangular blocks, and from 2D rectangular blocks to instantiable basis functions. To facilitate our discussion, we introduce several definitions, followed by the extraction and the instantiation procedure.

A. Terminology

Definition 1 (*Face* of a conductor)**.** Given an interconnect structure which consists of n conductors, M_i denotes the i-th conductor, and S^i is the overall surface of M_i. In Manhattan geometry, S^i can be decomposed into a set of non-overlapping rectangles, called *faces*, such that any union of two or more rectangles is not a rectangle. The faces of the i-th conductor are indexed by j in an arbitrary order, denoted by S^i_j.

This definition ensures that faces are the locally largest possible rectangles that cannot be further combined to form a larger rectangle. An example is illustrated in Fig. 10. The union of the rectangles S^i_1 and S^i_4 in Fig. 10(a) forms a larger rectangle S^i_1 in Fig. 10(b). Hence, the surface decomposition in Fig. 10(a) is not a set of faces. Note that such a decomposition is not unique. The rectangle S^i_4 in Fig. 10(a) can be either combined with S^i_2 or with S^i_1 to form a valid face.

Definition 2 (*Face* and *Induced basis functions*)**.** A *face basis function* $\psi_F(u,v)$ of a face S^i_j is defined as

$$\psi_F(u,v) = \begin{cases} 1, & (u,v) \in S^i_j \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

An instantiable basis function that is not a face basis function is called an *induced basis function*.

Definition 3 (*Face-induced and Side-induced basis functions*)**.** Consider the pair of faces involved in the instantiation of an induced basis function. The induced basis function is a *face-induced* basis function if the two faces are in parallel, and a *side-induced* basis function if two faces are perpendicular to each other.

We take Fig. 6(b) as an example. The induced basis function on the top face of the induced wire at the bottom is a *face-induced* basis function, whereas the induced basis functions on the side faces of the induced wire are *side-induced* basis functions.

Both the face-induced and the side-induced basis functions are built by combining 2D flat and arch blocks to account for the accumulated charge. In particular, the arch block is the 2D extension of the 1D arch shape as in Fig. 9. We formalize the arch shape as follows:

Definition 4 (*Arch shape*)**.** An *arch shape* $A_{\mathbf{p}}(r) : \mathbb{R} \mapsto [0,1]$ is a function parameterized by a vector of geometric parameters $\mathbf{p}$ with support in $[-a(\mathbf{p}), b(\mathbf{p})]$, where $a(\mathbf{p})$ is the *ingrowth length* and $b(\mathbf{p})$ is the *extension length* as functions of $\mathbf{p}$.

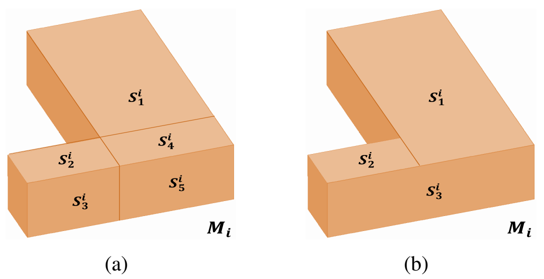


Fig. 10: Examples of (a) invalid and (b) valid faces of the conductor M_i. Some rectangles in (a) are not valid faces because $\{S_1^i, S_4^i\}$ and $\{S_3^i, S_5^i\}$ can be further combined into rectangles S_1^i and S_3^i in (b).

The parameters $a(\mathbf{p})$, $b(\mathbf{p})$, and h are depicted in Fig. 9. Among various geometric parameters, the wire separation h is the most essential parameter and must be included in $\mathbf{p}$. Other geometric parameters, such as the width of the induced wire, can be included in $\mathbf{p}$ to further improve capacitance accuracy if necessary.

Definition 5 (*Arch block*)**.** An *arch block* $T_{A,\mathbf{p}}(u,v) : \mathbb{R}^2 \mapsto [0,1]$ is a function supported in a subset S of a face and $T_{A,\mathbf{p}}(u,v) = A_{\mathbf{p}}(u)$ for all $(u,v) \in S$.

Definition 6 (*Flat block*)**.** A *flat block* $T_F(u,v) : \mathbb{R}^2 \mapsto \{0,1\}$ is a function supported in a subset S of a face and $T_F(u,v) = 1$ for all $(u,v) \in S$ and 0 otherwise.

An arch block is constructed simply by extending the value of a 1D arch shape into the second dimension.

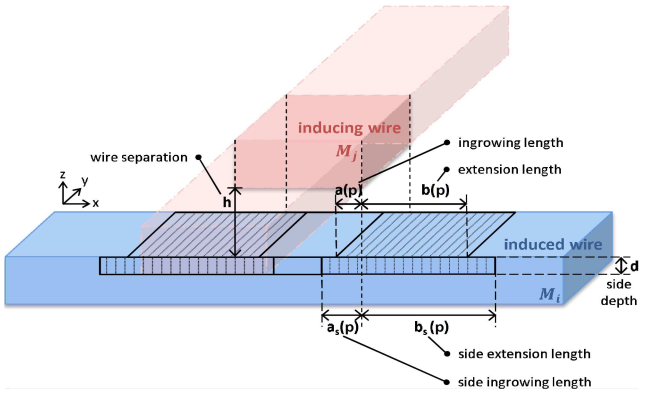
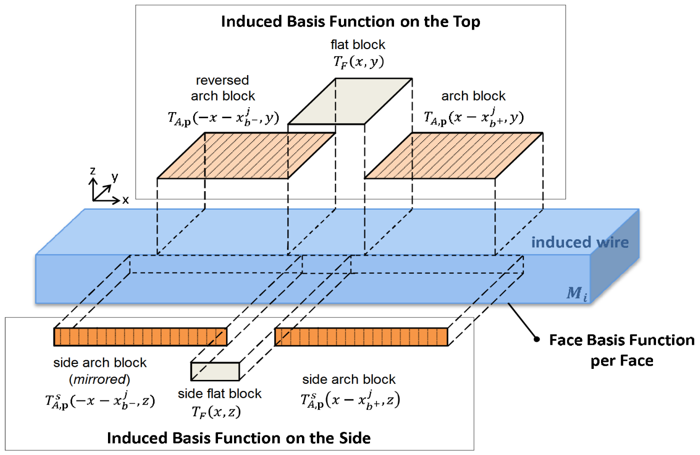


Fig. 11: Arch extraction setup.



Induced Basis Function on the Top
flat block
$T_F(x,y)$
reversed arch block
$T_{A,\mathbf{p}}(-x-x_{b^-}^j, y)$
arch block
$T_{A,\mathbf{p}}(x-x_{b^+}^j, y)$
z
y
x
induced wire
M_i
Face Basis Function per Face
side arch block (mirrored)
$T_{A,\mathbf{p}}^s(-x-x_{b^-}^j, z)$
side flat block
$T_F(x,z)$
side arch block
$T_{A,\mathbf{p}}^s(x-x_{b^+}^j, z)$
Induced Basis Function on the Side

Fig. 12: Arch extraction setup displayed in separate layers.

B. Arch shape extraction

The arch shapes from the decomposition in Fig. 9 are not best suited for constructing arch blocks because the long decaying tails significantly complicate the integration in (2). To extract arches with shorter tails, we truncate the support of arches based on the discretization in Fig. 11 and then use Galerkin testing to determine the corresponding charge distribution. It is worth noting that the tail truncation should be performed *before* Galerking testing. Directly truncating tail solutions in Fig. 9 *after* Galerkin testing is not recommended because it may invalidate the stretchability in Fig. 7 and results in significant loss of accuracy.

Fig. 12 shows that the function supports for the induced basis functions are separated from the face basis functions. Note that induced basis functions are laid directly on top of face basis functions and share a fraction of the support of the underlying face basis function. During the shape extraction, both the face basis functions and the flat blocks are represented as a single PWC basis function without further discretization. Arch blocks, on the other hand, are discretized along the u-direction into a set of PWC basis functions to account for the u-dependence of $T_{A,\mathbf{p}}(u,v) = A_{\mathbf{p}}(u)$ in Definition 5. Side arch shapes are also included and discretized similarly.

The choice of the arch shape parameters in Fig. 11 constitutes a tradeoff between computational efficiency, capacitance accuracy, and the generalization capability: whether the same set of parameter choices performs equally well across various wire configurations. Our experiments are summarized in Fig. 13. The corresponding capacitance values are compared against a reference value calculated using the standard BEM with a very fine discretization.

Another experiment is performed in Fig. 13(d), which shows arch shapes and side arch shapes, extracted by Galerkin testing based on the discretization in Fig. 12 for a range of extension lengths $b = b_s$. It can be seen from the figure that, for large values of b and b_s, the arch tails may become negative-valued. Such negative arch tails do not preserve the physical meaning and also do not generalize well for other wire configurations. Therefore, the maximally allowed extension and side extension lengths should be the length either until the wire boundaries or until the shape value decays to zero, whichever is smaller.

In practice, it is preferable to choose shorter lengths, or equivalently, smaller supports of basis functions, when considering the integration complexity. For the same integration quality, larger supports of basis functions typically requires more quadrature points, which slow down the overall computation. On the other hand, for the same computation time, larger supports may incur more errors in numerical integration. Accordingly, for the experiments in Fig. 13, we choose $d = 50\%$ for possible inductions from both upper and lower layers, choose $b = b_s = 0.3\,\mathrm{um}$ as it is the "elbow" point of the error curves in Fig. 13(b), and choose $a = a_s = 0$ because of their very minor effects on capacitance errors.

The "elbow" in Fig. 13(b) is of particular interest because it indicates a good tradeoff point that decreasing the lengths from this value greatly impacts the accuracy and increasing beyond this point only mildly improves the accuracy. This mild improvement is generally overshadowed by the complication of the integration scheme. Therefore, we choose such an elbow point as the extension lengths.

It is worth noting that parameter choices and arch shape extractions are technology node specific. Each technology node has its own layer spacing, wire thickness, and the effective dielectric constant, all of which affect the shapes of the reusable arches extracted from the setup in Fig. 12. Therefore, for the best capacitance accuracy, it is important to extract shapes and then reuse those shapes to construct basis functions in the same technology node.

C. Arch shape storage and retrieval

The arch shapes extracted in Section IV-B are reusable for basis function construction and must be stored for easy retrieval. Since arch shapes are smooth and low dimensional functions, tabulation or numerical fitting are viable solutions. In the following, we formulate an optimization problem for fitting a rational function to the arch shapes extracted in Section IV-B. Once the curve fitting is performed, only the coefficients of the resulting rational function need to be stored, and the shape retrieval is simply a function evaluation. Rational functions are a more suitable representation than polynomials because arch shapes exhibit a decaying slope. Combining the geometric parameters $\mathbf{p}$ and the position variable r of $A_{\mathbf{p}}(r)$ in Definition 4 with into a vector $\mathbf{r}$, we approximate

$$A_{\mathbf{p}}(r) = f(\mathbf{r}) = \frac{f_N(\mathbf{r})}{f_D(\mathbf{r})}, \tag{6}$$

where $f_N(\mathbf{r})$ and $f_D(\mathbf{r})$ are polynomials of degree n and m with coefficients β^N and β^D, respectively. Specifically,

$$f_N(\mathbf{r}) = \sum_{|\alpha|\le n} \beta^N_\alpha \mathbf{r}^\alpha, \tag{7}$$

and

$$f_D(\mathbf{r}) = \sum_{|\alpha|\le m} \beta^D_\alpha \mathbf{r}^\alpha. \tag{8}$$

The multi-index α, which is a (k+1)-tuple,

$$\alpha = (\alpha_r, \alpha_1, \ldots, \alpha_k) \tag{9}$$

represents the degree of each variable in $\mathbf{r}$ with $\mathbf{p} \in \mathbb{R}^k$. The notation $|\alpha|$ is the degree of $\mathbf{r}^\alpha$. With M data points, we can fit the rational function (6) to the extracted arch shapes using the following optimization program:

$$\begin{aligned} \underset{\beta^N,\beta^D}{\text{minimize}} \quad & \sum_{i=1}^{M} |A_{\mathbf{p}_i}(r_i) f_D(\mathbf{r}_i) - f_N(\mathbf{r}_i)|^2 \\ \text{subject to} \quad & \sum_{|\alpha|\le m} \beta^D_\alpha = 1. \end{aligned} \tag{10}$$

The program in (10) is a linearly constrained least-squares problem. The constraint is used to avoid common factors in $f_N(\mathbf{r})$ and $f_D(\mathbf{r})$. Otherwise, the trivial case $\beta^N_\alpha = \beta^D_\alpha = 0$ for all α is also a feasible solution. Because of the simple form of the constraint, the program in (10) can be further simplified by substituting the constraint into the objective function, resulting in a standard linear least-squares problem. For the IC technology we tested with and $\mathbf{p} = (h, w)$, i.e. wire separation and the width of the induced wire, a fitted rational function of degree $(n, m) = (2, 3)$ is sufficient for a maximum 0.3% arch shape error.

D. Instantiation procedures

Fig. 14(a) shows a pair of partially overlapping wires in adjacent metal layers, separated by h. We would like to demonstrate the procedure of constructing the instantiable basis function on the bottom wire induced by the top wire. Induced wire (the bottom wire) is denoted by M_i and the inducing wire (the top wire) is by M_j. The coordinates for the x, y, and z boundaries of M_i are denoted by $x^i_\pm$, $y^i_\pm$, and $z^i_\pm$, respectively. The faces of M_i directed outward in the $(+x)$- and $(-x)$-directions are referred to as S^i_{+x} and S^i_{-x}. The same naming convention is applied to the $\pm y$ and $\pm z$ directions and M_j. The instantiation procedure is as below:

1) Face basis functions: Create six face basis functions, each of which covers the entire domain of S^i_k, where $k = \pm x, \pm y, \pm z$.

2) Face-induced basis functions:

i) Check whether M_i and M_j overlap in their x-y plane projections. Stop if not.

ii) Check whether each edge of the bottom face S^j_{-z} of M_j is partially or completely within the projection of S^i_{+z}, i.e., $[x^i_-, x^i_+] \times [y^i_-, y^i_+]$. If yes, take the part of the edge that is within the projection of S^i_{+z} and instantiate on S^i_{+z} an arch block. For example, the right-front edge of S^j_{-z}, oriented in the y-direction, is first examined and an overlapping area is detected within $[y^j_-, y^i_+]$ by a width w. Accordingly, an arch block $T_{A,(h,w)}(x - x^j_+, y)$ is instantiated with the support $\left([x^j_+ - a, x^j_+ + b] \times [y^j_-, y^i_+]\right) \cap S^i_{+z}$ at $z = z^i_+$. The result is illustrated in Fig. 14(b).

iii) Repeat Step ii until all the edges of S^j_{-z} are examined. Additional two arch blocks are instantiated in Fig. 14(c). No arch block is for the right-back edge because the edge is outside the projection of S^i_{+z}.

iv) A flat block with the support $[x^j_- + a, x^j_+ - a] \times [y^j_- + a, y^i_+]$ at $z = z^i_+$ is placed between the arch blocks as in Fig. 14(c) if the support is not empty.

It is worth emphasizing that the face-induced basis function we instantiate here consists of one flat and three arch blocks but it is still a single basis function that contributes only one unknown to the linear system in (3).

3) Side-induced basis functions: A side-induced basis function is instantiated on a side face whenever the inducing wire protrudes out of the boundary of the induced wire. The side face S^i_{+y} in Fig. 14(a) is such an example. We demonstrate the procedures for S^i_{+y} as below:

i) Instantiate a pair of side arch blocks $T^s_{A,h}(x - x^j_+, z)$ and

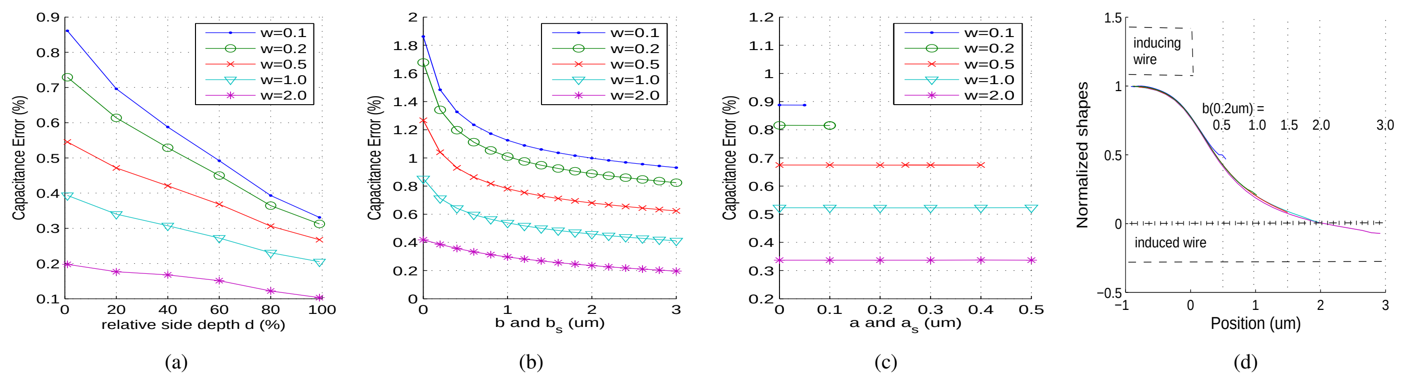


Fig. 13: Capacitance errors versus arch parameters and arch shapes versus b. These results are computed using Galerkin testing baed on the discretization in Fig. 12. The following dimensions (length, width, height) are in um^3: inducing wire $(15, 2, 0.2)$, induced wire $(15, w, 0.2)$. Vertical separation: 0.2 um. The dielectric constant is assumed to be $\varepsilon = \varepsilon_0$. (a) Errors versus d when a, a_s, b, and b_s are set to be maximum. (b) Errors versus $b = b_s$ when d is 50%, and a and a_s are set to be maximum. (c) Errors versus $a = a_s$ when d is 50%, and $b = b_s = 1.5$. (d) Arch shapes versus $b = b_s$ when $d = 50\%$, $a = a_s = 1$, and $h = 0.2$. The negative values of arch shapes can be considered as a subtraction from the face basis function.

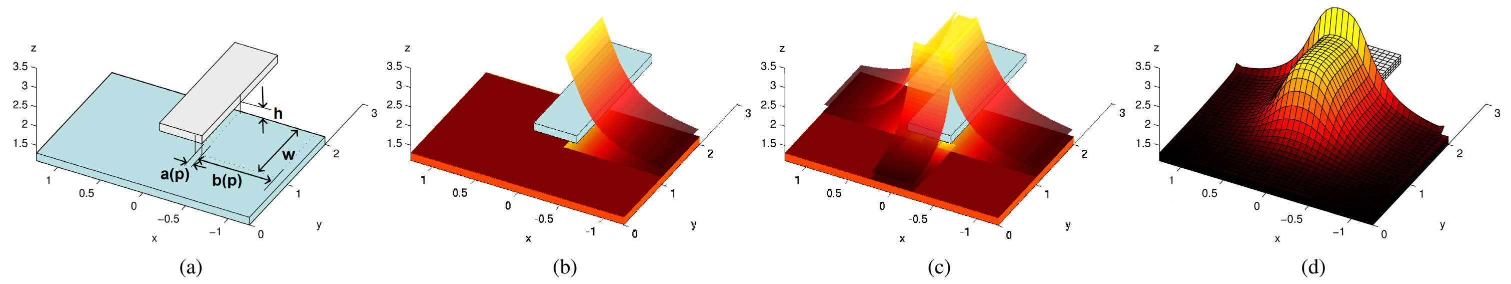


Fig. 14: Partially overlapping wires and instantiation of face-induced basis function. Units are um. (a) Wire geometry. (b) An arch block instantiated. (c) A complete face-induced basis function. (d) Comparison: Charge distribution using PWC basis functions with self-capacitance removed.

$T^s_{A,h}(-x + x^j_-, z)$ at $y = y^i_+$ with the supports

$$\left([x^j_+ - a_s, x^j_+ + b_s] \times [z^i_m, z^i_+]\right) \cap S_{i,+y} \text{ and}$$
$$\left([x^j_- - b_s, x^j_- + a_s] \times [z^i_m, z^i_+]\right) \cap S_{i,+y},$$

respectively, where $z^i_m = z^i_- d + z^i_+(1-d)$ and T^s_A denotes a side arch blocks.

ii) A flat block with the support $[x^j_- + a_s, x^j_+ - a_s] \times [z^i_m, z^i_+]$ at $y = y^i_+$ is placed between the two side arch blocks.

For the comparison purposes, the charge distribution of PWC basis functions (with self-capacitance removed) is shown in Fig. 14(d). In this example, only 17 instantiable basis functions are needed for 1.6% errors, whereas it requires 572 PWC basis functions for the same accuracy.

4) General setup: For the general case of multiple wires in the Manhattan geometry, the same procedure can be applied and the indices i and j are still referred to the induced wire and the inducing wire. The complete algorithm goes through every pair of wires M_i and M_j and apply the procedure to each pair of faces of M_i and M_j that directly face each other with opposite outward normal directions. Note that *no* induced basis function is needed for faces that come from the *same* wire. Only faces from different wires require induced basis functions to represent the induced charge due to the nearby presence of a potential difference.

V. Parallelization

Fig. 14(c) and 14(d) suggest that instantiable basis functions are a compact representation of charge distribution. Such compactness generally implies a smaller system in (3), or a faster system solve, and at the same time, complicates the integrals in the system setup (2) due to the employment of complex shapes. Accordingly, the percentage of the total computation for the system setup and for the system solving step is redistributed. The resulting problem consists of the majority of computation in the system setup, which is embarrassingly parallelizable, and a minor portion in the system solving part. Therefore, highly efficient parallelization can be achieved if the system setup workload is evenly distributed among parallel computing nodes. A standard parallelized direct solution method is then invoked to solve the resulting linear system (3).

We introduce a balanced workload partition scheme for constructing the system matrix P with minimal data communications as follows. Consider a single instantiable basis function ψ_i consisting of one or multiple blocks $T_{i,\bar{\imath}}$ as in Fig. 14(c),

$$\psi_i(\mathbf{r}) = \sum_{\bar{\imath}} T_{i,\bar{\imath}}(\mathbf{r}). \tag{11}$$

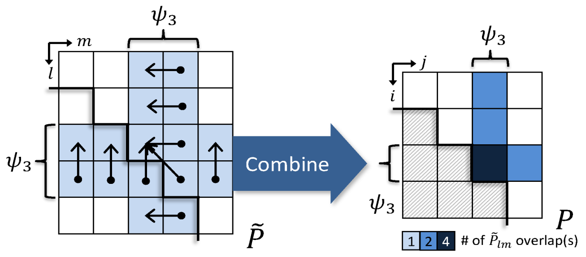


Fig. 15: Matrix $\tilde{P}$ transformed into system matrix P.

For total N instantiable basis functions constructed for a given set of wires, a matrix element in $P \in \mathbb{R}^{N\times N}$ can be computed as

$$P_{ij} = \sum_{\bar{\imath}}\sum_{\bar{\jmath}} \int_{s_{i,\bar{\imath}}}\int_{s'_{j,\bar{\jmath}}} \frac{T_{i,\bar{\imath}}(\mathbf{r})T_{j,\bar{\jmath}}(\mathbf{r}')}{4\pi\varepsilon\|\mathbf{r}-\mathbf{r}'\|}\mathrm{d}s'\mathrm{d}s = \sum_{\bar{\imath}}\sum_{\bar{\jmath}} P_{i\bar{\imath},j\bar{\jmath}}. \tag{12}$$

Each $T_{k,\bar{k}}$ is either an arch block or a flat block with its support $s_{k,\bar{k}}$ for $k = i, j$ if it belongs to an induced basis function, or simply a constant value 1 if it is a face basis function. From (12), it can be seen that workloads of parallel computing nodes are more balanced if the workload distribution is based on individual blocks than basis functions. Accordingly, we collect all blocks from each basis function and relabel them from 1 to M. A matrix $\tilde{P} \in \mathbb{R}^{M\times M}$ is defined such that

$$\tilde{P}_{lm} = \int_{s_l}\int_{s_m} \frac{T_l(\mathbf{r})T_m(\mathbf{r}')}{4\pi\varepsilon\|\mathbf{r}-\mathbf{r}'\|}\mathrm{d}s'\mathrm{d}s. \tag{13}$$

The matrix $\tilde{P}$ can then be transformed into P through summing the rows and the columns if the corresponding indexes are associated with the same basis function. Fig. 15 shows an example that a matrix P is constructed by four basis functions ($N = 4$) in which only ψ_3 consists of two blocks ($M = 5$). Then the index mapping between $T_{i,\bar{\imath}}$ and T_l is the following ordered set:

$$\{T_{1,1}, T_{2,1}, T_{3,1}, T_{3,2}, T_{4,1}\} = \{T_1, T_2, T_3, T_4, T_5\}. \tag{14}$$

In practice, M is typically 1.2 to 3 times greater than N, or equivalently, $\tilde{P}$ can be nine times larger than P in terms of memory allocation. To reduce memory access, row or column summations as in Fig. 15 can be performed "in-place" to avoid allocating $\tilde{P}$. To achieve high parallel efficiency, the construction of the matrix P is divided into equal number of $\tilde{P}_{lm}$ elements and distributed to each parallel computing node.

Memory allocation for P depends on the underlying architecture. For share-memory systems, a single memory pool is allocated for P and accessible to every parallel computing node. For distributed-memory systems, the size of a fraction of P is allocated for each parallel computing node except for the main node. The size of the entire P is allocated for the main node because the main node, at the end of system setup, aggregates and assembles the partial P from other nodes into the complete P matrix. The overall parallelization strategy is listed in **Algorithm**.

VI. Implementation

Our implementation CAPLET aims to cover the complete capacitance extraction flow from GDSII layout files to capacitance matrices. The main procedures include GDSII binary parsing, surface geometry computation, basis function construction, and the BEM using instantiable basis functions with parallelization (the core solver). The first three steps together can generate either instantiable or PWC basis functions *automatically* from GDSII files. This feature enables direct performance comparisons between our solver and FASTCAP using realistic IC layouts. The core solver part consists of the Galerkin integral computation (13) and the solver parallelization. In this section, we will focus our discussion mainly on the surface geometry computation, the computation of the Galerkin integrals, and the parallelization implementation. Other details can be consulted from the code release website: http://www.rle.mit.edu/cpg/codes/caplet/

Algorithm Parallelization for system setup

```
input: $D$ parallel nodes $d = 1,\dots,D$, $D \geq 2$. Basis functions $\psi_i, i = 1,\dots,N$ and the corresponding blocks $T_l, l = 1,\dots,M$.
output: The system matrix $P$.

Allocate and initialize $P = \mathbf{0}$.
Construct an index mapping $I : l \mapsto i$ s.t.
    $i = I(l) = I(l')$ where $T_l, T_{l'} \in \psi_i$ for all $l, l'$.
$K \longleftarrow M(M+1)/2$.
Partition $\{k \mid k = 1,\dots,K\}$ into $K_1 = \{1,\dots,|K_1|\},\dots,$
    $K_D = \{\sum_{j=1}^{D-1}|K_j|+1,\dots,\sum_{j=1}^{D}|K_j| = K\}$ s.t.
    $|K_j| = \lfloor K/D \rfloor, j = 2,\dots,D$, and $|K_1| = K-(D-1)|K_D|$.
for each parallel node $d \in \{1,\dots,D\}$ do
    for all $k \in K_d$ do
        $m \longleftarrow \lfloor (-1+\sqrt{1+8k})/2 \rfloor + 1$
        $l \longleftarrow k - m(m-1)/2$
        $\tilde{p} \longleftarrow$ Galerkin integral (13) of $T_l$ and $T_m$
        if $l \neq m$ and $I(l) = I(m)$ then
            $P_{I(l),I(m)} \longleftarrow P_{I(l),I(m)} + 2\tilde{p}$
        else
            $P_{I(l),I(m)} \longleftarrow P_{I(l),I(m)} + \tilde{p}$
        end if
    end for
end for
```

A. Surface geometry computation

GDSII layout files record layered paths and polygons, which are easier for layout design but not suited for setting up 3D geometric surfaces. For the end goal of generating basis functions automatically, it is necessary to transform the given layered paths and polygons into 3D non-overlapping, disjoint rectangular surfaces and be able to identify each conductor from unordered geometries.

The procedure of generating surface geometries is depicted in Fig. 16. The key step is to decompose polygons into rectangles because the number of surface rectangles is closely related to the resulting number of basis functions. However, direct minimization of the number of rectangles as in [33] may not be a good strategy. This is because an additional layer of induced basis functions are instantiated on top of surface rectangles. Therefore, preserving large rectangles should be prioritized as opposed to premature optimization on the number of rectangles. Our algorithm starts from searching for the longest edge of a polygon, sweeping the longest edge over an area until touching another edge, storing and taking away the rectangular area defined by the sweep. The process is repeated

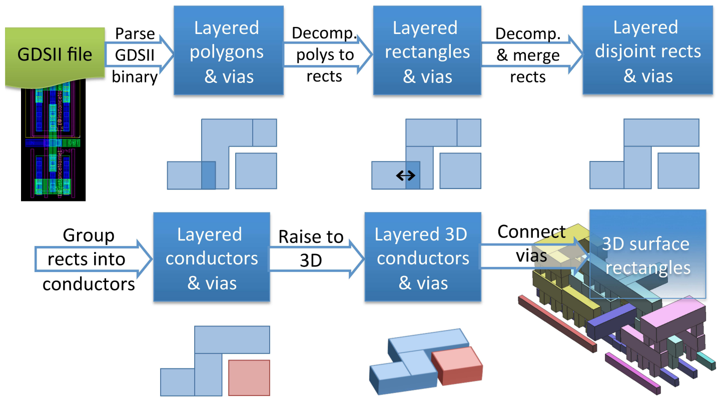


Fig. 16: Flowchart for generating surface rectangles from GDSII files.

until all remaining polygons are by themselves rectangles. This strategy empirically provides larger supports for flat and arch blocks without being fragmented.

After the decomposition, further computation is performed to ensure that the rectangles are disjoint, followed by combining neighboring rectangles to reduce the unnecessary fragmentation of face basis functions. The connectivity between rectangles is then computed to form conductors. Finally, the construction of surface rectangles are completed after layered conductors are raised to 3D and connected through vias.

B. *Computation of Galerkin integrals*

The integral in (13) can be simplified significantly when only the Manhattan geometry is considered:

$$\tilde{P}_{l,m} = \int_{y_{l1}}^{y_{l2}} \int_{x_{l1}}^{x_{l2}} \int_{y'_{m1}}^{y'_{m2}} \int_{x'_{m1}}^{x'_{m2}} \frac{T_l(x,y)T_m(x',y')}{4\pi\varepsilon\sqrt{\delta x^2+\delta y^2+\delta z^2}} \mathrm{d}s'\mathrm{d}s, \tag{15}$$

where $\delta u = u - u'$, $u = x, y, z$, and the differentials are defined as $\mathrm{d}s = \mathrm{d}x\mathrm{d}y$, $\mathrm{d}s' = \mathrm{d}x'\mathrm{d}y'$. The support of $T_l(x, y)$ is $[x_{l1}, x_{l2}] \times [y_{l1}, y_{l2}]$ at z, and likewise for $T_m(x', y')$.

The functions $T_k(x, y)$, $k = l$ or m in (15) can be either an arch or a flat block. Note that a face basis function is mathematically equivalent to a flat block, so we will refer to "flat block" for both cases. Therefore, there are total three combinations of block types as follows:

1) Arch-Arch: When arch blocks are functions of different directions, for instance, $T_l(x)$ and $T_m(y')$, the expression in (15) can be rearranged into

$$\int_{x_{l1}}^{x_{l2}} T_l(x) \int_{y'_{m1}}^{y'_{m2}} T_m(y') \left[\int_{y_{l1}}^{y_{l2}} \int_{x'_{m1}}^{x'_{m2}} \frac{\mathrm{d}x'\mathrm{d}y/(4\pi\varepsilon)}{\sqrt{\delta x^2+\delta y^2+\delta z^2}} \right] \mathrm{d}y'\mathrm{d}x. \tag{16}$$

The 2D integral in the inner bracket of (16) is identical to the collocation integral using PWC basis functions and can be computed analytically. The remaining outer 2D integral is then numerically integrated through Gaussian quadrature.

When both arch blocks are functions of the same direction, say, $T_l(x)$ and $T_m(x')$, the inner integral is no longer the collocation integral for PWC basis functions. For such a case, we approximate the shorter arch on $[x'_{m1}, x'_{m2}]$ as a linear combination of K PWC basis functions

$$T_m(x') = \sum_{k=1}^{K} T_m(x'_k)\chi_k(x'), \tag{17}$$

where $\chi_k(x')$ is a PWC basis function with support in discretization segments $[x'_{d,k-1}, x'_{d,k}]$ and $x'_k = \frac{1}{2}(x'_{d,k-1}+x'_{d,k})$, in which each $x'_{d,k}$ is a discretization point along $[x'_{m1}, x'_{m2}]$ such that

$$x'_{m1} = x'_{d,0} < x'_{d,1} < \cdots < x'_{d,K-1} < x'_{d,K} = x'_{m2}. \tag{18}$$

The expression in (15) is then rearranged as

$$\sum_{k=1}^{K} T_m(x'_k) \int_{x_{l1}}^{x_{l2}} T_l(x) \left[\int_{y_{l1}}^{y_{l2}} \int_{x'_{d,k-1}}^{x'_{d,k}} \int_{y'_{m1}}^{y'_{m2}} \frac{\mathrm{d}y'\mathrm{d}x'\mathrm{d}y/(4\pi\varepsilon)}{\sqrt{\delta x^2+\delta y^2+\delta z^2}} \right] \mathrm{d}x, \tag{19}$$

which is a weighted sum of the Arch-Flat cases.

2) Arch-Flat: The Arch-Flat case can be computed using the same expression as (19) with $K = 1$ and $T_m(x'_1) = 1$. The bracketed 3D integral can be analytically calculated and the outer integral is handled via Gaussian quadrature.

3) Flat-Flat: When both blocks are flat, (15) becomes a 4D integral of the $1/r$ Green's function and can also be integrated analytically.

One performance consideration is that the computation cost of analytical integration grows rapidly with dimension. For instance, the expression for the 2D integral is a summation of eight terms whereas more than 100 terms are involved in the 4D integral. In terms of runtime, the former is roughly 10 to 20 times faster than the latter. To lower the chance of invoking higher dimensional expressions, we exploit the decaying property of the Green's function: The higher dimensional expression is invoked only when two blocks are overlapping or close by. The lower dimensional expression is substituted when two blocks are far away. We partition the distance between blocks into the regions for 4D, 3D, 2D, and 0D integrals, the last of which is referred to the $1/r$ approximation. As a concrete example, when two blocks are far away from each other, a 2D integral can be approximated as

$$\int_{y'_1}^{y'_2} \int_{x'_1}^{x'_2} \frac{\mathrm{d}x'\mathrm{d}y'}{\sqrt{(x-x')^2+(y-y')^2+(z-z')^2}} \approx \frac{\Delta}{\|\mathbf{r}-\mathbf{r}'\|}, \tag{20}$$

where $\Delta = (x'_2 - x'_1)(y'_2 - y'_1)$ is the area of the integration ranges and $\mathbf{r}, \mathbf{r}'$ are position vectors defined as $\mathbf{r} = (x, y, z)$ and $\mathbf{r}' = \frac{1}{2}(x'_2+x'_1, y'_2+y'_1, 2z')$. We choose the approximation boundaries such that using lower dimensional expressions incur less than the 0.8% errors, which are empirically less than 0.5% capacitance errors in the examples we tested. Note that such approximations are, in fact, a necessity, especially when involving extreme geometric distances or lengths. In such a situation, evaluating the analytical expressions without using the lower dimensional approximants may result in several orders of magnitude of errors due to the floating point underflow problem. The approximation in (20) is also implemented in FASTCAP to accelerate computation and also to avoid the aforementioned finite-precision problem. We

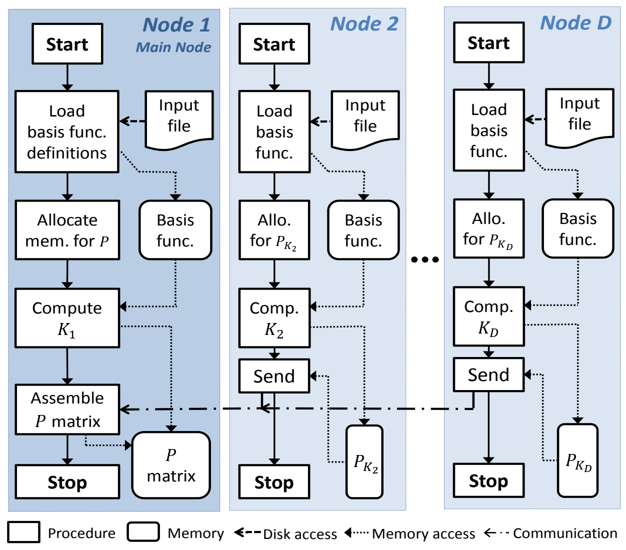


Fig. 17: Flowchart for parallelizing the system setup on a distributed-memory system.

extend such a notion to multi-level approximations in this work.

C. Parallelization of system setup

Our implementation supports both shared-memory and distributed-memory systems using OpenMP and MPI, respectively.

1) On a shared-memory system: Once the input file is loaded, the system matrix P is allocated in the shared memory pool. Each parallel computing node computes its own partition of $\tilde{P}$ elements and store the results directly in the memory for P.

2) On a distributed-memory system: The parallel computing node i loads its own copy of the input file, allocates the memory for the corresponding partial matrix of P, compute the integrals, and stores the results in its allocated memory as a partial system matrix P_{K_i}. The only exception is the main node, which allocates the memory for the whole system matrix P. When the work for the node i is done, the resulting partial matrix P_{K_i} is sent to the main node for being integrated into the system matrix P. It is worth noting that the communication between parallel computing nodes only happens once through the entire system setup step. This procedure is flowcharted in Fig. 17.

D. Parallelization of system solution

The state-of-the-art LAPACK libraries are highly optimized for specific computer architectures. We use OpenBLAS [34], an optimized LAPACK library based on GotoBLAS2 [35], to solve the symmetric system in (3) in our implementation.

VII. Examples

We test the general performance of our algorithm using realistic logic gates in Fig. 18 and test the scalability of parallel efficiency using the structure in Fig. 19.

Fig. 18 shows an inverter and a NAND gate, each of which consists of eight and ten conductors, respectively. Individual conductors of both structures are indicated by different colors. The surface of the inverter is discretized into PWC basis functions and the induced basis functions for the NAND gate are placed on top of its surface for the demonstration purpose. For these two structures, we test both the serial and the parallel performance on a workstation with a 3.2-GHz 4-core CPU. We first compute the reference capacitance values by iteratively refining discretization and using FASTCAP with the 4th-order multipole expansion to extract solutions. The discretization panel size is refined by 10% for each iteration, and the solution

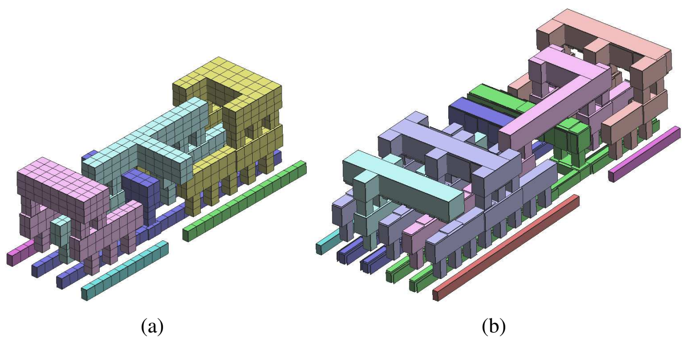


Fig. 18: (a) An inverter gate with PWC basis functions. (b) An NAND gate with instantiable basis functions elevated from the surface to indicate their locations. Individual conductors are indicated by different colors.

TABLE I: Performance summary for logic gates.

Inverter (err=4.1%)	# Basis functions	Computation Time (ms) Setup	Solving	Total
Reference	12342	1090	5660	6750
FASTCAP	2340	270	730	1010
CAPLET	497	90	4.3	94
Improvement	4.7 : 1	3.0×	170×	11×
CAPLET: 2 cores		51 (88%)	4.1	55
CAPLET: 3 cores		34 (88%)	4.0	38
CAPLET: 4 cores		28 (84%)	3.9	32
NAND (err=4.5%)				
Reference	26062	2310	16070	18390
FASTCAP	4884	580	1870	2450
CAPLET	1075	290	35	325
Improvement	4.5 : 1	2.0×	53×	7.5×
CAPLET: 2 cores		170 (83%)	28	198
CAPLET: 3 cores		120 (82%)	26	146
CAPLET: 4 cores		90 (79%)	24	114

TABLE II: Performance for 24×24 buses in Fig. 19.

# of used	Shared-memory system with 4 nodes			Dist.-memory system with 10 nodes		
nodes	Time(s)	Spdup.	Eff.	Time(s)	Spdup.	Eff.
1	40.5	1.00×	100%	44.1	1.00×	100%
2	21.7	1.86×	93%	22.7	1.94×	97%
4	11.1	3.65×	91%	12.3	3.56×	93%
8				6.04	7.30×	91%
10				4.95	8.91×	89%

is considered converged when the capacitance values differ less than 0.1% for the last two iterations. Next, CAPLET is used to extract the capacitance matrix C and the result is measured against the reference value C^0 through

$$\text{error} = \max_{i,j} \left[\frac{C_{ij} - C^0_{ij}}{C^0_{ii}} \right]. \tag{21}$$

The resulting errors for the inverter and the NAND gate are 4.1% and 4.5%, respectively. Several factors contribute to such errors, including shape simplification and reuse (Observation 1, 2, and Definition 5), the neglect of basis functions for corner induction as in Fig. 14(c), and integration errors. To make a fair comparison with FASTCAP, we adjust the discretization such that the error of FASTCAP with the default 2nd-order multipole expansion is at the same level as the error of the previous CAPLET result, both of which are compared against the reference value C^0. The performance for both structures is summarized in Table I.

In serial execution, our algorithm achieves 11 and 7.5 times speedup over FASTCAP in total times. The solving times are significantly improved, 170 and 53 times faster for each case, while there is still improvement for the system setup part, 3.0 and 2.0 times, respectively. The basis function construction times are not included as they are below 1ms for all cases. This result reflects the redistribution property discussed in Section V. The 2, 3, and 4-core parallelization performances for either structure are listed under each improvement calculation. The memory footprints for FASTCAP and CAPLET for the inverter are 28MB and 1.9MB, for the NAND gate 65MB and 8.9MB, respectively.

The parallel scalability is tested using a 24×24 bus structure depicted in Fig. 19. In serial execution, the computation times using our solver and using FASTCAP are 40.5 s and 92.0 s, respectively, for the same 2.3% error. Our solver achieves 91% parallel efficiency using our OpenMP implementation on a 3.2 GHz 4-core workstation and 89% using our MPI implementation on a 2.6 GHz machine that contains two processors with total 10 cores, whereas the parallel efficiencies of the parallel pre-corrected FFT [15] and of the parallel fast multipole expansion method [16] drop significantly to 42% and 65% at eight cores, respectively. These efficiency values are the best available data for a much smaller 2×2 bus example from their original papers. We further our experiment on a 80-node IBM BladeCenter H cluster with the LAPACK solver restricted to only four cores. It can be seen that the system setup part roughly keeps the linear trend with the number of parallel computing nodes, whereas the efficiency for the full extraction saturates around 44% when the LAPACK solving routine dominates the computation. The detailed parallel efficiencies for different number of nodes (up to 10 cores) are summarized in Table II.

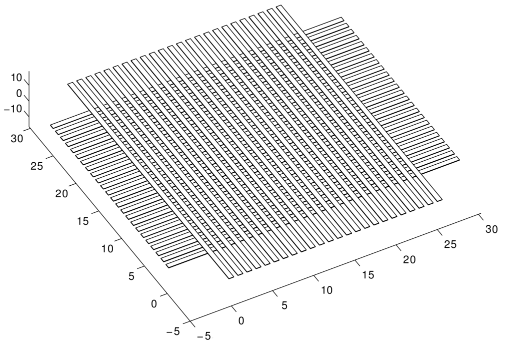


Fig. 19: Structure of 24×24 buses. Units are um.

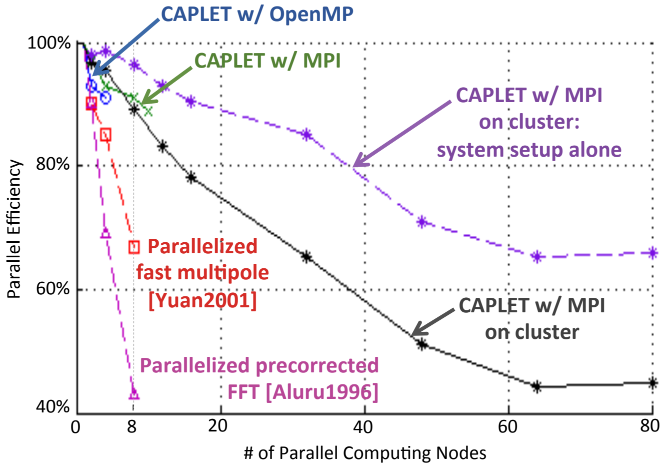


Fig. 20: Parallel scalability of the proposed method on the 24×24 bus example compared with the best available parallel scalability in [15], [16].

VIII. Conclusion

In this work, we developed a set of instantiable basis functions for Manhattan geometries and employed it as a compact charge representation in the boundary element method. Such compactness redistributes the computation from the system solving step to the system setup step, and effectively transforms the problem into a more efficiently parallelizable one. We achieve about 90% parallel efficiency with ten parallel computing nodes through MPI implementation. In addition, for realistic logic gate cells, our algorithm in serial execution outperforms FASTCAP by a factor of ten for an inverter, and by a factor of six for a NAND gate. When computed with four cores, our method achieves sub-second field-solver-accurate performance: 0.038s for an inverter and 0.19s for a NAND gate. The extension of instantiable basis functions to non-Manhattan geometries is possible by using the same instantiation rules and considering more general integration schemes.

We developed an open-source toolkit that provides an automatic extraction flow from GDSII layout files to capacitance matrices with GUI and OpenGL visualization. Our code is available at [2].

IX. Acknowledgement

The authors would like to thank Roberto Suaya, Professor at UTN FRBA and Chief Scientist Emeritus of Mentor Graphics, for his long-term support for this research direction, Prof. Stefano Grivet-Talocia of Politecnico di Torino, Italy, for providing the access to the cluster, Tarek El-Moselhy of the D. E. Shaw Group for many helpful discussions at the early stage of this project, Zhen Li of MIT for providing logic cell layouts, and Ertugrul Demircan of Freescale Semiconductor for sharing his industry perspective.

References

[1] W.-Y. Jung, G.-U. Cha, Y.-B. Kim, J.-H. Baek, and C.-K. Kim, "Integrated interconnect circuit modeling for vlsi design," in *Design Automation Conference, 1995. Proceedings of the ASP-DAC '95/CHDL '95/VLSI '95., IFIP International Conference on Hardware Description Languages; IFIP International Conference on Very Large Scale Integration., Asian and South Pacific*, aug. 1995, pp. 165 –169.

[2] Y.-C. Hsiao. (2014) Project CAPLET: Parallelized capacitance extraction toolkit. [Online]. Available: http://www.rle.mit.edu/cpg/codes/caplet/

[3] K. Nabors and J. White, "Fastcap: a multipole accelerated 3-D capacitance extraction program," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 10, no. 11, pp. 1447–1459, Nov 1991.

[4] J. Phillips and J. White, "A precorrected-FFT method for electrostatic analysis of complicated 3-D structures," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 16, no. 10, pp. 1059–1072, Oct 1997.

[5] S. Kapur and D. Long, "IES3: efficient electrostatic and electromagnetic simulation," *Computational Science & Engineering, IEEE*, vol. 5, no. 4, pp. 60–67, Oct-Dec 1998.

[6] W. Shi, J. Liu, N. Kakani, and T. Yu, "A fast hierarchical algorithm for three-dimensional capacitance extraction," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 21, no. 3, pp. 330–336, Mar 2002.

[7] W. Chai, D. Jiao, and C.-K. Koh, "A direct integral-equation solver of linear complexity for large-scale 3D capacitance and impedance extraction," in *Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE*, July 2009, pp. 752–757.

[8] W. Yu and Z. Wang, "Enhanced qmm-bem solver for three-dimensional multiple-dielectric capacitance extraction within the finite domain," *Microwave Theory and Techniques, IEEE Transactions on*, vol. 52, no. 2, pp. 560–566, Feb 2004.

[9] Y. L. Coz and R. Iverson, "A stochastic algorithm for high speed capacitance extraction in integrated circuits," *Solid-State Electronics*, vol. 35, no. 7, pp. 1005 – 1012, 1992.

[10] Y. L. Coz, H. Greub, and R. Iverson, "Performance of random-walk capacitance extractors for IC interconnects: A numerical study," *Solid-State Electronics*, vol. 42, no. 4, pp. 581 – 588, 1998.

[11] T. El-Moselhy, I. Elfadel, and L. Daniel, "A hierarchical floating random walk algorithm for fabric-aware 3D capacitance extraction," in *Computer-Aided Design - Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM International Conference on*, Nov 2009, pp. 752–758.

[12] W. Yu, H. Zhuang, C. Zhang, G. Hu, and Z. Liu, "RWCap: A floating random walk solver for 3-D capacitance extraction of very-large-scale integration interconnects," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 32, no. 3, pp. 353–366, March 2013.

[13] K. Zhai, W. Yu, and H. Zhuang, "GPU-friendly floating random walk algorithm for capacitance extraction of VLSI interconnects," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, March 2013, pp. 1661–1666.

[14] B. Zhang, W. Yu, and C. Zhang, "Improved pre-characterization method for the random walk based capacitance extraction of multi-dielectric VLSI interconnects," *International Journal of Numerical Modelling: Electronic Networks, Devices and Fields*, 2015.

[15] N. R. Aluru, V. B. Nadkarni, and J. White, "A parallel precorrected FFT based capacitance extraction program for signal integrity analysis," in *Proceedings of the 33rd annual Design Automation Conference*, ser. DAC '96. New York, NY, USA: ACM, 1996, pp. 363–366.

[16] Y. Yuan and P. Banerjee, "A parallel implementation of a fast multipole-based 3-D capacitance extraction program on distributed memory multicomputers," *Journal of Parallel and Distributed Computing*, vol. 61, no. 12, pp. 1751–1774, 2001.

[17] F. Gong, H. Yu, L. Wang, and L. He, "A parallel and incremental extraction of variational capacitance with stochastic geometric moments," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 20, no. 9, pp. 1729–1737, Sept 2012.

[18] Y.-C. Hsiao, T. El-Moselhy, and L. Daniel, "Efficient capacitance solver for 3D interconnect based on template-instantiated basis functions," in *Electrical Performance of Electronic Packaging and Systems, 2009. EPEPS '09. IEEE 18th Conference on*, Oct. 2009, pp. 179 –182.

[19] P. Silvester, "Modal network theory of skin effect in flat conductors," *Proceedings of the IEEE*, vol. 54, no. 9, pp. 1147–1151, Sept. 1966.

[20] L. Daniel, A. Sangiovanni-Vincentelli, and J. White, "Interconnect electromagnetic modeling using conduction modes as global basis functions," in *Electrical Performance of Electronic Packaging, 2000, IEEE Conference on.*, 2000, pp. 203–206.

[21] ——, "Proximity templates for modeling of skin and proximity effects on packages and high frequency interconnect," in *Computer Aided Design, 2002. ICCAD 2002. IEEE/ACM International Conference on*, Nov. 2002, pp. 326–333.

[22] S. Ortiz and R. Suaya, "Fullwave volumetric maxwell solver using conduction modes," in *Computer-Aided Design, 2006. ICCAD '06. IEEE/ACM International Conference on*, Nov. 2006, pp. 13–18.

[23] K. J. Han and M. Swaminathan, "Inductance and resistance calculations in three-dimensional packaging using cylindrical conduction-mode basis functions," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 28, no. 6, pp. 846–859, June 2009.

[24] Y.-C. Hsiao and L. Daniel, "A highly scalable parallel boundary element method for capacitance extraction," in *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, 2011, pp. 552–557.

[25] Y. Zhang and A. Zemanian, "Contributions of corner singularities of the capacitances of interconnections wires," in *Circuits and Systems, 1995. ISCAS '95., 1995 IEEE International Symposium on*, vol. 2, Apr-3 May 1995, pp. 1420–1423 vol.2.

[26] Y. Su, E. T. Ong, and K. H. Lee, "Automatic classification of singular elements for the electrostatic analysis of microelectromechanical systems," *Journal of Micromechanics and Microengineering*, vol. 12, no. 3, pp. 307–315, 2002.

[27] M.-D. Ker and K.-C. Hsu, "Overview of on-chip electrostatic discharge protection design with SCR-based devices in CMOS integrated circuits," *Device and Materials Reliability, IEEE Transactions on*, vol. 5, no. 2, pp. 235–249, June 2005.

[28] J. Vinson and J. Liou, "Electrostatic discharge in semiconductor devices: an overview," *Proceedings of the IEEE*, vol. 86, no. 2, pp. 399–420, Feb 1998.

[29] J. A. Morrison and J. A. Lewis, "Charge singularity at the corner of a flat plate," *SIAM Journal on Applied Mathematics*, vol. 31, no. 2, pp. pp. 233–250, 1976.

[30] R. De Smedt and J. Van Bladel, "Field singularities at the tip of a metallic cone of arbitrary cross section," *Antennas and Propagation, IEEE Transactions on*, vol. 34, no. 7, pp. 865–870, Jul 1986.

[31] A. E. Beagles, J. R. Whiteman, and W. Wendland, "General conical singularities in three-dimensional poisson problems," *Mathematical Methods in the Applied Sciences*, vol. 11, no. 2, pp. 215–235, 1989.

[32] S. Natarajan, M. Breuer, and S. Gupta, "Process variations and their impact on circuit operation," in *Defect and Fault Tolerance in VLSI Systems, 1998. Proceedings., 1998 IEEE International Symposium on*, Nov 1998, pp. 73–81.

[33] D. Eppstein, "Graph-theoretic solutions to computational geometry problems," in *Graph-Theoretic Concepts in Computer Science*, ser. Lecture Notes in Computer Science, C. Paul and M. Habib, Eds. Springer Berlin Heidelberg, 2010, vol. 5911, pp. 1–16.

[34] X. Zhang, Q. Wang, and Y. Zhang, "Model-driven Level 3 BLAS performance optimization on Loongson 3A processor," in *Parallel and Distributed Systems (ICPADS), 2012 IEEE 18th International Conference on*, Dec 2012, pp. 684–691.

[35] K. Goto and R. A. van de Geijn, "Anatomy of high-performance matrix multiplication," *ACM Trans. Math. Softw.*, vol. 34, no. 3, pp. 12:1–12:25, May 2008.

Yu-Chung Hsiao received the B.S. degree in electrical engineering from National Taiwan University, Taiwan, in 2006, and the M.S. and Ph.D. degrees in electrical engineering and computer science from MIT in 2010 and 2015, respectively. His research interests include integral equation methods, predictive modeling, system identification, nonlinear dynamic systems, parallel computing, and convex optimization.

Luca Daniel received the Laurea degree (*summa cum laude*) in electronic engineering from the Universita di Padova, Italy, in 1996, and the Ph.D. degree in electrical engineering from the University of California, Berkeley, in 2003.

He is an Associate Professor in the Electrical Engineering and Computer Science Department of the Massachusetts Institute of Technology (MIT), Cambridge. His research interests include accelerated integral equation solvers and parameterized stable compact dynamical modeling of linear and nonlinear dynamical systems with applications in mixed-signal/RF/mm-wave circuits, power electronics, MEMs, and the human cardiovascular system.

Dr. Daniel received the 2014 IEEE Transactions on CAD of Integrated Circuits and Systems best paper award, the 1999 IEEE Transactions on Power Electronics best paper award, the 2003 ACM Outstanding Ph.D. Dissertation Award in Electronic Design Automation, five best paper awards in international conferences, the 2009 IBM Corporation Faculty Award, and 2010 Early Career Award from the IEEE Council on Electronic Design Automation.