A Linear-Time Eigenvalue Solver for Finite-Element-Based Analysis of Large-Scale Wave Propagation Problems in On-Chip Interconnect Structures

Jongwon Lee*, Venkataramanan Balakrishnan, Cheng-Kok Koh, and Dan Jiao
School of Electrical and Computer Engineering, Purdue University
465 Northwestern Avenue, West Lafayette, IN 47907

Introduction
The analysis and design of next-generation VLSI circuits using accurate electromagnetics-based models result in numerical problems of very large scale. Typically, the solution of a problem with \( N \) parameters requires at least \( O(N) \) computation. With next generation VLSI circuits, however, even \( O(N) \) is prohibitively high since \( N \) is very large. In [1], a method that partially addresses this issue was developed for full-wave modeling of large-scale interconnect structures. In this method, a number of seeds (a seed has a unique cross section) are first recognized from an interconnect structure. In each seed, the original wave propagation problem is represented as a generalized eigenvalue problem. The complexity of solving 3D interconnects of \( O(N) \) is then overcome by seeking the solution of a few 2D seeds, which is then post-processed to obtain the solution of the original 3D problem through the development of an on-chip mode-matching technique. The procedure is rigorous, and entails no approximation. The size of the system matrix constructed by this method, \( M \), is the number of unknowns in a 2D seed residing on either the \( x-y \) or \( y-z \) plane (\( y \) is the stack growth direction).

Taking a test-chip interconnect as an example, \( M \) is 6678 while \( N \) is 10.1 million. With a larger example, \( M \) is 222k while \( N \) is 336 million. While the \( M \) parameter model was successfully constructed in [1], the solution of the associated modeling problem with \( O(M) \) complexity still remains open. The computational bottleneck is the solution of a generalized eigenvalue problem. Efficient algorithms such as ARPACK [2] still require \( O(M^2) \) storage and operations due to a dense matrix-vector multiplication. We present an algorithm that provides a solution to the generalized eigenvalue problem with \( O(M) \) complexity, thus paving the way for the full-wave simulation of next generation VLSI circuits.

Formulation
The generalized eigenvalue problem that results from a finite-element-based analysis of inhomogeneously filled waveguides [5] can be written as:

\[
\begin{bmatrix}
A & 0 \\
0 & 0 \\
\end{bmatrix}
\begin{bmatrix}
e_t \\
e_z \\
\end{bmatrix} = \gamma
\begin{bmatrix}
B & B_x \\
B_z & 0 \\
\end{bmatrix}
\begin{bmatrix}
e_t \\
e_z \\
\end{bmatrix}
\]

in which the eigenvalues correspond to the propagation constants \( \gamma \), and the eigenvectors characterize the transverse electric field \( e_t \) and longitudinal electric field \( e_z \). Matrices \( A \) and \( B \) are complex-valued due to the penetration of fields into on-chip conductors. The entries of \( A \) and \( B \) are given by
\begin{align}
A_{n,ij} &= \int_{\Omega} \left[ \frac{1}{\mu_r} \left( \nabla \times N_i \right) \cdot \left( \nabla \times N_j \right) - k_0^2 \varepsilon_r N_i \cdot N_j \right] d\Omega, \quad B_{n,ij} = \int_{\Omega} \frac{1}{\mu_r} N_i \cdot N_j d\Omega \\
B_{z,ij} &= \int_{\Omega} \frac{1}{\mu_r} N_i \cdot \nabla \varepsilon_r d\Omega, \quad B_{z,ij} = \int_{\Omega} \frac{1}{\mu_r} \nabla \varepsilon_r \cdot N_i d\Omega \\
B_{z,ij} &= \int_{\Omega} \left[ \frac{1}{\mu_r} \left( \nabla \varepsilon_r \right) \cdot \left( \nabla \varepsilon_r \right) - k_0^2 \varepsilon_r \varepsilon_r \varepsilon_r \right] d\Omega
\end{align}

where \( \varepsilon_r \) denotes the complex permittivity that accounts for conductivity, \( N \) represents the edge basis function [5] used to expand the transverse field, \( \xi \) is the node basis function used to expand the longitudinal one, and \( \Omega \) is the computational domain.

Eqn. (1) can be compactly expresses as

\[ A x = \lambda B x. \]  

Matrices \( A \) and \( B \) are sparse and of size \( O(M) \). Since the number of modes that can propagate in an on-chip structure is generally much less than \( M \), the main computational problem here is to find a few eigenpairs of the large sparse matrix system shown in (3). The Arnoldi iteration is particularly suited for this task. When \( k \) significant eigen-pairs are sought for a standard eigenvalue problem \( G x = \lambda x \), a \( k \)-step Arnoldi process only requires \( 2Mk^2 \) flops when \( G \) is sparse. However, when the generalized eigenvalue problem (3) is translated to a standard eigenvalue problem \( G x = \lambda x \), the matrix \( G = B^{-1}A \) is dense because \( B^{-1} \) is dense. Therefore, the number of flops associated with each Arnoldi iteration rises to \( O(M^2) \) due to the computation of a dense matrix-vector multiplication \( Gv \) at each iteration step. The key contribution in this paper is the reduction of this computation to \( O(M) \). We will first reduce the system matrix from 2D to 1D, then solve the reduced system and recover other unknowns. Two efficient algorithms will be developed to accomplish these two tasks with \( O(M) \) complexity.

A. Reduction from 2D to 1D lines in complexity much less than \( O(M) \)

In order for accelerated convergence, we will consider the standard shift-Arnoldi iteration, wherein, (3) is transformed to \((A - \tau B)^{-1}B x = (\lambda - \tau)^{-1} x\), in which \( \tau \) is shift parameter to improve the convergence rate. At each iteration step of the Arnoldi process, there is a need to compute \( w = (A - \tau B)^{-1} B v \), which can be solved from

\[ (A - \tau B)w = Bv. \]

Although both \( A \) and \( B \) are sparse matrices, their sparse patterns are not amenable for direct use in our computational technique. Therefore, we will transform \((A - \tau B)\) to a banded matrix by permuting the ordering of the underlying variables. Further regularity of structure can be realized by discretizing the computational domain into rectangular elements. Since typical on-chip interconnects have Manhattan geometry, rectangular elements are indeed natural for use. We discretize the computational domain into \( N_x \) segments along \( x \) and \( N_y \) segments along \( y \). We denote the \( y \)-direction edge unknowns as \( e_y \), \( x \)-direction edge unknowns as \( e_x \), and \( z \)-direction node unknowns as \( e_z \). We then first order \( e_y \) of line 1 (\( y \)-orientated), \( e_z \) on line 1; and along \( x \) we proceed to \( e_x \) between line 1 and line 2; \( e_y \) of line 2, \( e_z \) on line 2, and so on. By doing so, we generate a banded matrix formed by submatrices in all segments. In each submatrix, the matrix block
formed by \( e_x \) unknowns is \( T \), and the matrix block formed between \( e_y/e_z \) and \( e_x \) is \( D \). We then eliminate all the \( e_x \) unknowns to reduce the system matrix from 2D to multiple 1D lines. The cost of this step can be reduced to the evaluation of \( DT^{-1}D^T \) for each unique \( x \)-segment of unit thickness by exploring matrix properties. The same matrix properties we have derived in [3] apply here although the details are different. The dimension of \( T \) is \( N_y+1 \). The operation of \( DT^{-1}D^T \) could cost \( O(N_y^3) \), which is expensive. We now show how the complexity of this operation can be reduced to \( O(N_y^2) \). A careful examination of \( T \) reveals that \( T \) is a tridiagonal matrix. It is well-known that there exist two sequences \( \{ u_i \}, \{ v_i \}, i = 1, 2, \ldots, N_y+1 \) such that for \( j \geq i \), \((T^{-1})_{ij} = u_i v_j \). In other words, although the inverse of \( T \) is dense and therefore can comprise up to \((N_y+1)^2 \) parameters, it can be compactly represented by \( 2N_y+1 \) numbers containing in \( \{ u_i \} \) and \( \{ v_i \} \) \((u_i = 1\) without loss of generality). In addition, the sequences \( \{ u_i \}, \{ v_i \}, i = 1, 2, \ldots, N_y+1 \) can be generated in \( O(N_y) \) operations. Furthermore, \( D \) is sparse. Hence the cost of \( DT^{-1}D^T \) scales as \( O(N_y^2) \). Since the direction \( y \) is the stack growth direction, \( N_y \) is a constant. In addition, it is much less than \( M \). For example, \( N_y \) is 20. Hence, compared to \( O(M) \), the cost of \( O(N_y^2) \) is negligible.

**B. Solving reduced system matrix in \( O(M) \) Complexity**

The reduced system matrix forms a block tridiagonal matrix of order \((2N_y+1)(N_x+1)\), which can be denoted as \( S = tri(X_{1:N_x+1}, Y_{1:N_y}) \). Here each \( X_i, Y_i \in \mathbb{C}^{(2N_y+1)(N_x+1)}, \) Thus \( S \in \mathbb{C}^{((2N_y+1)(N_x+1)\times(2N_y+1)(N_x+1))} \), with \( N_y+1 \) diagonal blocks of size \((2N_y+1)\) each. Since the right-hand side of (5) changes at each iteration step of an Arnoldi process, we are specifically interested in its direct solution. Elegant theoretical results exist which describe the inverses of block tridiagonal matrices. For a symmetric \( m \times m \) block tridiagonal matrix \( S \), there exist two sequences of \( m \times m \) matrices \( \{ U_i \}, \{ V_i \}, \) such that for \( j \geq i \), \((S^{-1})_{ij} = U_i V_j^T \). While theoretically elegant, the computation of parameters \( \{ U_i \} \) and \( \{ V_i \} \) is beset by numerical problems for even modest-sized problems. Here, we will adopt a ratio-based approach which is numerically stable [4],

\[
U_i = R_i U_{i+1}, \quad V_i^T = V_{i+1}^T S_i, \quad i = 1, 2, \ldots, N_x
\]

\[
R_i = X_i^{-1} Y_i, \quad R_i = (X_i - Y_i^T R_{i+1})^{-1} Y_i, \quad i = 2, \ldots, N_x
\]

\[
S_{N_x} = Y_{N_x} X_{N_x+1}^{-1}, \quad S_i = Y_i (X_i + S_{i+1} Y_{i+1}^T)^{-1}, \quad i = N_x - 1, \ldots, 1
\]

As can be seen from (6), the computational cost of obtaining \( \{ U_i \} \) and \( \{ V_i \} \) is \( O(N_x N_y^3) \). Since \( \{ U_i \} \) and \( \{ V_i \} \) constitute a compact representation of the inverse of a block tridiagonal matrix, the matrix vector multiplication \( S^{-1} b \) can be performed in \( O(N_y N_x^2) \) operations. The overall computational complexity is \( O(MN_x^2 + kMN_y) \) in CPU time. As \( N_y \) is a constant that is much less than \( M \) and the number of significant modes, \( k \), is small, it scales as \( O(M) \). The memory cost is \( O(MN_x) = O(M) \). Hence, the computation of (3) scales with \( M \) linearly.

**Numerical Results**

First, the proposed linear-time eigenvalue solver was validated on a shield microstrip line shown in Fig. 7.8 in [5]. The dispersion curves obtained by the proposed algorithms agree very well with the reference results given in Fig. 8.9 of
[5], as shown in Fig. 1. In this simulation, the number of Arnoldi iterations, and hence the dimension of the upper Hessenberg matrix was 9, which was chosen based on the convergence of the eigenvalue solution. The dimension of the eigenvalue system in (1) was 289. The value of \( \tau \) in (3) was chosen as 1.5. In fact, numerical experiments show that the choice of \( \tau \) can be arbitrary in the region bounded by the minimum and maximum relative permittivity, and does not affect the computational performance. In Fig. 2, the CPU times of the proposed algorithm for varying numbers of unknowns were plotted, demonstrating a linear complexity. Next, a larger-scale wave propagation problem that consists of 300 propagation modes was simulated. The dimensions of this problem were set according to typical geometrical dimensions. There were in total 300 interconnect wires placed. Each wire was 0.4975 \( \mu \)m wide. The frequency of interest was 1 GHz. The 300 eigenvalues extracted by the proposed eigenvalue solver have shown an excellent agreement with those generated by Matlab. In this simulation, the number of Arnoldi iterations was 320. The value of \( \tau \) was chosen as 3.5. The overall CPU time of the proposed solver was shown to be 1.5 times faster than that of Matlab, which relies on ARPACK [2], a state-of-the-art large-scale sparse eigenvalue solver, for eigenvalue computation. For a fair comparison, we provided Matlab with the same \( \tau \) and required it to compute only 300 eigenvalues.

References


Fig. 1. Eigenvalues simulated by the proposed solver in comparison with the reference results.

Fig. 2. CPU time of the proposed solver v.s. the number of unknowns.