A scalable distributed method for quantum-scale device simulation

Stephen Cauley, a) Jitesh Jain, b) Cheng-Kok Koh, c) and Venkataramanan Balakrishnan d)
School of Electrical and Computer Engineering, Purdue University, West Lafayette, Indiana 47907-2305

(Received 11 January 2007; accepted 4 May 2007; published online 29 June 2007)

We present an algorithm for the fast and accurate simulation of nanoscale devices. The idea underlying the algorithm is a divide-and-conquer method based on the nonequilibrium Green’s function formalism. This formalism has provided a unifying conceptual framework for the analysis of quantum transport in nanodevices and the computations therein are of significant interest across many areas of research. We offer two applications, the atomistic level simulation of silicon nanowires and the two-dimensional simulation of nanotransistors, which highlight the benefits of the divide-and-conquer framework. The inherently parallel algorithm presented here allows for computing resources to be flexibly allocated toward either solving problems of larger sizes in comparable time or speeding up the solution of a problem for a given size. Our algorithm facilitates the solution of problems orders of magnitude larger and, in most cases, was able to achieve substantial speedup, as compared to the current state of the art algorithm. Thus, the method presented here allows for large-scale simulation problems that can now be realized without the use of special purpose hardware or approximation methods. © 2007 American Institute of Physics. [DOI: 10.1063/1.2748621]

I. INTRODUCTION

The accurate and efficient modeling and simulation of nanoscale devices have become a problem of increasing importance. Due to the scaling of device sizes, previously employed macromodeling techniques are no longer sufficient. This has prompted the development of more accurate models from the fundamental principles of quantum mechanics. Models that capture such quantum effects tend to involve the solution of large systems of linear equations. This results in extraordinary demands on memory, and simulation with these models requires a prohibitive amount of computation. It is fair to say that the lack of scalability of most existing techniques presents a significant bottleneck in the simulation of such systems.

In this paper, we consider both the two-dimensional quantum mechanical modeling of nanotransistors and the atomistic level simulation of silicon nanowires. The problem of computing electron densities through the nonequilibrium Green’s function formalism reduces to a mathematical problem of finding the diagonal entries of the inverse of a block tridiagonal matrix. Block tridiagonal matrices and their inverses are encountered in a number of areas from engineering to applied mathematics and physics. There are several existing methods for finding the inverse of a block tridiagonal matrix in a computationally efficient manner. However, all of these suffer from being either numerically unstable or heavily memory intensive and hence impractical for problems of very large sizes (e.g., 10^6 × 10^6).

In Ref. 4 the authors presented an algorithm which in addition to being computationally efficient is also numerically stable for large problems, making it perhaps among the best-known algorithms for quantum-scale simulations. However, the demands on memory of this algorithm are still high and it is also of a recursive nature and hence cannot be readily made parallel. We have developed an algorithm that addresses these computational challenges by being

- computationally efficient,
- numerically well conditioned, and
- able to distribute computation across multiple processors, due to its inherent ability to be parallelized.

We provided a direct comparison of our algorithm with that of Ref. 4. Specifically, the ability to trade off computing resources with both simulation time and problem size is illustrated for two applications. We consider both the atomistic simulation of silicon nanowires, using the sp^3d^5s^* tight-binding model, and the two-dimensional simulation of nanotransistors. Although the method from Ref. 4 is directly applicable to these applications, the simulation of a modest 3 × 3 × 102.6 nm^3 silicon nanowire would require up to 24 Gbytes of memory using the approach. We conclude that our algorithm allows for the simulation of nanoscale devices not attainable with current techniques.

II. PROBLEM DESCRIPTION

Consider the two-dimensional model of a nanoscale transistor, shown in Fig. 1. The body of the transistor is projected onto a two-dimensional nonuniform spatial grid of dimension N_x × N_y, where N_x (N_y) denote the number of grid points along the depth (length) of the device. The values of N_x (N_y) will vary with device geometry, given a fixed spatial resolution. Increasing the values will improve the accuracy of the simulation by defining a fine grid across the region of interest. In the case of a nanotransistor, typical values of N_x

---

a)Electronic mail: stcauley@ecn.purdue.edu
b)Electronic mail: jjain@ecn.purdue.edu
c)Electronic mail: chengkok@ecn.purdue.edu
d)Electronic mail: ragu@ecn.purdue.edu
and \(N_z\) could result in problem sizes \(\left( N_x \times N_z \right)\) of the order of millions. We shall now provide a brief review of the governing physics for this device.

The Hamiltonian of a valley \(b\) for electrons, associated with the device under consideration, is as follows:

\[
H_b(\mathbf{r}) = \frac{-\hbar^2}{2m^b_x} \left( \frac{\partial}{\partial x} \left( \frac{1}{m^b_x} \frac{\partial}{\partial x} \right) + \frac{\partial}{\partial y} \left( \frac{1}{m^b_y} \frac{\partial}{\partial y} \right) + \frac{\partial}{\partial z} \left( \frac{1}{m^b_z} \frac{\partial}{\partial z} \right) \right)
+ V(\mathbf{r}),
\]

(1)

where \((m^b_x,m^b_y,m^b_z)\) are the components of effective mass in valley \(b\). The equations of motion for the retarded Green’s function \(G^\text{r}\) and less-than Green’s function \(G^\text{<}\) are

\[
\begin{align*}
E - \frac{\hbar^2 k^2}{2m^b_x} - H_b(\mathbf{r}) & \quad G^\text{r}(\mathbf{r}_1, \mathbf{r}_2, \mathbf{k}_z, E) \\
- \int d\mathbf{r} \Sigma^b_1(\mathbf{r}_1, \mathbf{r}_2, \mathbf{k}_z, E)G^\text{r}(\mathbf{r}_1, \mathbf{r}_2, \mathbf{k}_z, E) & = \delta(\mathbf{r}_1 - \mathbf{r}_2),
\end{align*}
\]

(2)

\[
\begin{align*}
E - \frac{\hbar^2 k^2}{2m^b_x} - H_b(\mathbf{r}) & \quad G^\text{<}(\mathbf{r}_1, \mathbf{r}_2, \mathbf{k}_z, E) \\
\int d\mathbf{r} \Sigma^b_1(\mathbf{r}_1, \mathbf{r}_2, \mathbf{k}_z, E)G^\text{r}(\mathbf{r}_1, \mathbf{r}_2, \mathbf{k}_z, E) & = \int d\mathbf{r} \Sigma^b_1(\mathbf{r}_1, \mathbf{r}_2, \mathbf{k}_z, E)G^\text{<}(\mathbf{r}_1, \mathbf{r}_2, \mathbf{k}_z, E).
\end{align*}
\]

(3)

Here \(G^\text{r}\) is the advanced Green’s function, \(\Sigma^b_1\) (\(\Sigma^\text{<}_b\)) are the self-energy terms corresponding to electron (phonon) scattering, and the position vector and wave vector are denoted \(\mathbf{r}\) and \(\mathbf{k}\), respectively. Given \(G^\text{r}\) and \(G^\text{<}\) the density of states and the charge density can be written as a sum of contributions from the individual valleys,

\[
N(\mathbf{r}, k_z, E) = \sum_b N^b(\mathbf{r}, k_z, E) = \frac{1}{\pi} \text{Im}[G^\text{r}(\mathbf{r}, \mathbf{r}, k_z, E)],
\]

(4)

\[
\rho(\mathbf{r}, k_z, E) = \sum_b \rho^b(\mathbf{r}, k_z, E) = -i [G^\text{<}(\mathbf{r}, \mathbf{r}, k_z, E)].
\]

(5)

III. MATHEMATICAL FORMULATION

Initially, we will turn our attention to the matrix constructed from (2), using the spatial discretization shown in Fig. 1. It can be shown that the problem of computing electron densities in a nanotransistor can be reduced to finding the diagonal blocks of \(G^\text{r}\), where \(KG^\text{r} = I\) and \(K\) is a block tridiagonal matrix of the form

\[
\begin{pmatrix}
K_{11} & K_{12} & 0 \\
K_{21} & K_{22} & K_{23} \\
0 & K_{32} & K_{33}
\end{pmatrix}
\]

FIG. 1. Two-dimensional spatial grid for a nanotransistor.

FIG. 2. (Color online) Cross section of Si nanowire [100].

FIG. 3. (Color online) Device model using Si nanowire [100].

In addition to the two-dimensional nano transistor problem detailed above, we investigate the atomistic simulation of silicon nanowires. This is a two-dimensionally confined transport problem, where each atom is represented by a set of orbitals, see Figs. 2 and 3. In the case of the \(sp^3d^5s^*\) model, ten orbitals are considered, 20 if spin-orbit coupling is included. For this case the value of \(N_z\) is dependent on the lattice structure, i.e., the number of atoms and orbitals in a given layer, and the value of \(N_z\) is dictated by the total number of layers. Although the method in this paper can be extended to handle other materials and geometries, for simplicity of illustration we examine only square cross-section Si nanowires in the [100] direction.

When considering the computation of electron densities for nanoscale devices, the self-consistent solution of the Green’s function is often the most-time intensive step in the simulation process. In Ref. 4, it was shown that the approximation by a block tridiagonal structure for the left-hand side in (2) and (3) facilitates the efficient calculation of the electron density. Similarly, through the general framework shown in Ref. 8 we can again exploit block tridiagonal structures in order to determine the desired physical quantities for the atomistic simulation of nanowires. In this paper we will consider only the case of the retarded Green’s function \(G^\text{r}\); all results shown herein can be easily extended to the case of the less-than Green’s function \(G^\text{<}\).
where each $A_i, B_i \in \mathbb{C}^{N_i \times N_i}$. Thus $K \in \mathbb{C}^{N_N \times N_N}$, with $N_N$ diagonal blocks of size $N_i$ each. We will use the notation $K = \text{tri}(A_1, N_1, B_1, B_{i-1})$ to compactly represent such a block tridiagonal matrix. Solving large systems of linear equations, with structure matching that of $K$, is a well studied problem across several areas of research. However, the problem of determining only select entries of $K^{-1}$ (diagonal entries of $G'$ in the case of electron densities) offers computational challenges.

The inverse of a block tridiagonal matrix can be computed explicitly, as demonstrated in Ref. 9. Specifically, there exist two (nonunique) sequences of matrices $\{U_i\}, \{V_i\}$ such that for $j \geq i$, $(K^{-1})_{ij} = U_i V_j^T$. Hence, $K^{-1}$ can be written as:

\[
K^{-1} = \begin{pmatrix}
U_1 V_1^T & U_1 V_2^T & U_1 V_3^T & \cdots & U_1 V_{N_N}^T \\
V_2 U_1^T & U_2 V_2^T & U_2 V_3^T & \cdots & U_2 V_{N_N}^T \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
V_{N_N} U_1^T & V_{N_N} U_2^T & V_{N_N} U_3^T & \cdots & V_{N_N} U_{N_N}^T
\end{pmatrix}.
\]

The above result has also been shown in Refs. 3 and 10. The $U$ and $V$ sequences can be constructed in a numerically stable fashion, as demonstrated in Ref. 11. Specifically, there exist two sequences of “ratio” matrices $\{R_i\}, \{S_i\}$ such that the inverse of $K$ can be written as:

\[
K^{-1} = \begin{pmatrix}
D_1 & D_1 S_1 & D_1 S_1 S_2 & \cdots & D_1 S_1 S_2 \cdots S_{N_N-1} \\
R_1 D_1 & D_2 & D_2 S_2 & \cdots & D_2 S_2 \cdots S_{N_N-1} \\
R_2 R_1 D_1 & R_2 D_2 & D_3 & \cdots & D_3 \cdots S_{N_N-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
D_{N_N} = A_{N_N}^{-1} (I + B_{N_N}^T D_1 S_1 S_2 \cdots S_{N_N-1}).
\end{pmatrix}
\]

where the diagonal blocks of the inverse $D_i$ and the ratio sequences satisfy the following relationships:

\[
R_i = A_i^{-1} B_i, \quad R_i = (A_{i-1} - B_{i-1}^T R_{i-1})^{-1} B_{i-1},
\]

\[
i = 2, \ldots, N_N - 1,
\]

\[
S_{N_N-1} = B_{N_N-1} A_{N_N}^{-1}, \quad S_i = B_i (A_{i+1} - S_{i+1} B_{i+1}^T)^{-1},
\]

\[
i = N_N - 2, \ldots, 1,
\]

\[
D_1 = (A_1 - B_1 S_1^T)^{-1},
\]

\[
D_{i+1} = (A_{i+1} - B_{i+1} S_{i+1}^T)^{-1} (I + B_{i+1}^T D_i S_i),
\]

\[
i = 1, \ldots, N_N - 2,
\]

\[
D_{N_N} = A_{N_N}^{-1} (I + B_{N_N}^T D_{N_N-1} S_{N_N-1}).
\]
a correction term parametrized by the $N_x \times N_x$ matrix $B$, which we will denote as the "bridge matrix."

Using the standard matrix inversion lemma from linear algebra, we have

$$K^{-1} = (\tilde{K} + XY)^{-1} = \tilde{K}^{-1} - (\tilde{K}^{-1}X)(I + Y\tilde{K}^{-1}X)^{-1}(Y\tilde{K}^{-1}),$$

where

$$\tilde{K}^{-1}X = \left( \begin{array}{cc} -\phi_1^{-1}(::,i)B_i & 0 \\ 0 & -\phi_2^{-1}(i,1)B_i^T \end{array} \right),$$

$$(I + Y\tilde{K}^{-1}X)^{-1} = \left( \begin{array}{cc} I & -\phi_1^{-1}(1,1)B_i^T \\ -\phi_1^{-1}(i,i)B_i & I \end{array} \right)^{-1},$$

$$(\tilde{K}^{-1}X)(I + Y\tilde{K}^{-1}X)^{-1}(Y\tilde{K}^{-1}) = \left( \begin{array}{cc} -\phi_1^{-1}(::,i)B_i & 0 \\ 0 & -\phi_2^{-1}(i,1)B_i^T \end{array} \right) \left( \begin{array}{cc} I & -\phi_1^{-1}(1,1)B_i^T \\ -\phi_1^{-1}(i,i)B_i & I \end{array} \right)^{-1} \left( \begin{array}{cc} 0 & \phi_2^{-1}(::,1)^T \\ \phi_1^{-1}(::,i)^T & 0 \end{array} \right)$$

$$= \left( \begin{array}{cc} -\phi_1^{-1}(::,i)B_i & 0 \\ 0 & -\phi_2^{-1}(i,1)B_i^T \end{array} \right) \left( \begin{array}{cc} J_{11} & J_{12} \\ J_{21} & J_{22} \end{array} \right) \left( \begin{array}{cc} 0 & \phi_2^{-1}(::,1)^T \\ \phi_1^{-1}(::,i)^T & 0 \end{array} \right)$$

$$= \left( \begin{array}{cc} -\phi_1^{-1}(::,i)BJ_{12}\phi_1^{-1}(::,i)^T & -\phi_1^{-1}(i,i)BJ_{11}\phi_2^{-1}(::,1)^T \\ -\phi_2^{-1}(i,1)BJ_{22}\phi_1^{-1}(::,i)^T & -\phi_2^{-1}(::,1)BJ_{21}\phi_2^{-1}(::,1)^T \end{array} \right),$$

where

$$J = \left( \begin{array}{cc} J_{11} & J_{12} \\ J_{21} & J_{22} \end{array} \right) = (I + Y\tilde{K}^{-1}X)^{-1}.$$

It is important to note that the diagonal entries of (9) are divided implicitly based on which subproblem solution they are modifying, e.g., the $k$th diagonal entry for the solution of subproblem 1 is modified by the term $\phi_1^{-1}(k,::)BJ_{12}\phi_1^{-1}(k,::)^T$ in $O(N_x^2)$ operations. Furthermore, the "adjustment" term, $(I + Y\tilde{K}^{-1}X)^{-1}$, depends only on corner blocks, i.e., $\phi_2^{-1}(i,1)$ and $\phi^{-1}(i,i)$, of the inverses of the submatrices. These observations provide the basis for our formulation of the global solution in the more general case. Specifically, the proposed algorithm will address the issue of how to combine multiple subproblem solutions in both a memory and computationally efficient manner.

IV. DIVIDE-AND-CONQUER ALGORITHM

A. Overview

The procedure begins with separating the block tridiagonal matrix $K$ into $D$ submatrices, each of which is joined to their neighbors by a bridge matrix. The procedure presented in the previous section motivates the formulation of the global solution by combining the subproblem solutions in a simple radix-2 fashion, shown for the case $D=4$ in Fig. 4. However, this method requires that every entry in the matrix needs to be updated, making it both memory and computationally intensive. Alternatively, matrix maps are created to capture the effect of each combining step without performing all of the associated computations. Adjustments to the matrix maps at each combining stage are not constant and must be modified to follow the procedure (8), detailed in Sec. III. These maps can then be used in the final stage to transform the subproblem solutions into the global solution.

Matrix maps are created to produce the cumulative effect of each combining step associated with a particular subproblem. In this context the term "cumulative effect" refers not only to the adjustment of the diagonal entries of a particular submatrix into a corresponding section of the global but also the ability to outwardly project, from a given subproblem, any information necessary to further the accumulation process. To better understand these effects, we will begin by examining the initial stages of the combining process shown in Fig. 4, i.e., forming the combined subproblems $\phi_1^{-1}$ and $\phi_1^{-1}$. For simplicity of illustration we will assume each subproblem to be of equal size, where $N$ denotes the number of blocks for the corresponding submatrices. Thus, the two submatrices $\phi_1$ and $\phi_2$ can be used to form $\phi_1^{-1}$ via the bridge matrix $B_N$. The first (last) block row (column) of $\phi_1^{-1}$ can then be described in the following manner:

$$\phi_1^{-1}(::,i) = \phi_1^{-1}(::,i) - (-\phi_1^{-1}(1,N)B_NJ_{12}\phi_1^{-1}(1,N)^T$$

$$- \phi_1^{-1}(1,N)B_NB_NJ_{11}\phi_2^{-1}(1,:)^T).$$
Similar to the example from Sec. III the term used to efficiently produce the required block entries shown the following relationships:

$$
\varphi^{-1}_{1-2}(\cdot,2N) = \bar{\varphi}^{-1}_{1-2}(\cdot,2N) - \left(- \varphi^{-1}_{2}(N,1)B_N^TJ_{21}\varphi^{-1}_{1}(\cdot,1)\right)^T,
$$

and the $r$th diagonal block of $\varphi^{-1}_{1-2}$ can be described using the following relationship:

$$
\varphi^{-1}_{1-2}(r,r) = \bar{\varphi}^{-1}_{1-2}(r,r) - \left(- \varphi^{-1}_{2}(r,N)B_NJ_{12}\varphi^{-1}_{1}(r,N)\right)^T, \quad r \leq N,
$$

$$
\varphi^{-1}_{1-2}(r,r) = \bar{\varphi}^{-1}_{1-2}(r,r) - \left(- \varphi^{-1}_{2}(r,1)B_N^TJ_{21}\varphi^{-1}_{1}(1,r)\right)^T, \quad r > N,
$$

given the adjustment matrix

$$
J = \begin{pmatrix}
I & -\varphi^{-1}_{2}(1,1)B_N^T \\
-\varphi^{-1}_{1}(N,N)B_N & I
\end{pmatrix}^{-1}.
$$

Similar to the example from Sec. III the term $\bar{\varphi}^{-1}_{1-2}$ represents the combination of the individual submatrix inverses. It is important to note that all the updates shown above are dependent only on the first (last) block row (column) of the subproblems, only as a function of the submatrix corner blocks and the bridge matrix. In addition, while it is clear that the diagonal blocks need to be updated in order to form the solution to the combined subproblems, the necessity for the preservation of the effects to the first block row and last block column (which includes all corner blocks of $\varphi^{-1}_{1-2}$) is demonstrated in the following section.

### B. Combining effect

To model the effect of any combining stage, it is necessary to know the corner block elements from the inverse of each combined submatrix. This process can be easily illustrated by continuing our example. Suppose we consider the final combing stage for the case $D=4$, where we will denote $Q_1 = \varphi_{1-2}$ and $Q_2 = \varphi_{3-4}$. The approach from above can again be used to combine $Q_1$ and $Q_2$. Here the updates will again only be dependent on the first (last) block row (column) of the inverses, as a function of the bridge matrix $B_{2N}$ and the corner blocks enclosed in the brackets shown below:

$$
\bar{K}^{-1} = \begin{pmatrix}
Q_1^{-1} & 0 \\
0 & Q_2^{-1}
\end{pmatrix} =
\begin{pmatrix}
\begin{bmatrix}
Q_1^{-1}(1,1) & \cdots & [Q_1^{-1}(1,2N)] \\
\vdots & \ddots & \vdots \\
[Q_1^{-1}(1,2N)^T] & \cdots & [Q_1^{-1}(2N,2N)]
\end{bmatrix} & 0 \\
0 & \begin{bmatrix}
Q_2^{-1}(1,1) & \cdots & [Q_2^{-1}(1,2N)] \\
\vdots & \ddots & \vdots \\
[Q_2^{-1}(1,2N)^T] & \cdots & [Q_2^{-1}(2N,2N)]
\end{bmatrix}
\end{pmatrix}.
$$

It should be noted that both $Q_1^{-1}$ and $Q_2^{-1}$ will have formulation (7) since $\varphi_{1-2}$ and $\varphi_{3-4}$ are block tridiagonal. It is also important to note that the relationships shown in (10) can be used to efficiently produce the required block entries shown in the brackets above, i.e., the corner blocks for each of the joined submatrices. This concept will be generalized into the notion of matrix maps, which offer a general framework facilitating the accumulation of all combining effects needed to
transform a subproblem solution into a given section of the global.

C. Matrix maps

The effects of combining multiple subproblems will be captured, in a recursive fashion, through the use of matrix maps. These linear mappings facilitate the generation, through the use of subproblem ratio sequences (7), of any information necessary to propagate the subproblem solutions into a section of the global. For any combining stage, the process of updating matrix maps can be broken down into two categories: adjustments to upper subproblems and those to lower subproblems, the distinction being their location with respect to the bridge point. This procedure will be illustrated using the final combining stage for the case $D=4$, as described in Sec. IV B. The adjustment matrix $J$ for the combining step is defined as follows:

$$Z_1 = -Q_2^{-1}(1,1)B_{2N}^T, \quad Z_2 = -Q_1^{-1}(2N,2N)B_{2N}.$$  

$$P = (I - Z_1 Z_2)^{-1},$$

$$J = \begin{pmatrix} P & -PZ_1 \\ -Z_2 P & I + Z_2 P Z_1 \end{pmatrix} = \begin{pmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{pmatrix}. \quad (12)$$

The process of combining subproblems $Q_1$ and $Q_2$ is completely analogous to that described in (10) and (11), with the effects on the first (last) block row (column), along with the diagonal blocks, shown below:

$$K^{-1}(1,:) = \tilde{K}^{-1}(1,:) - (-Q_1^{-1}(1,2N)B_{2N}J_{12}Q_2^{-1}(1,2N)^T)$$

$$-Q_2^{-1}(2N,1)B_{2N}^TJ_{21}Q_1^{-1}(1,2N)$$

$$+ [Q_1^{-1}(1,2N)B_{2N}J_{12}]Q_2^{-1}(1,2N)^T \times B_{2N}J_{21}Q_2^{-1}(1,2N),$$

$$K^{-1}(:,4N) = \tilde{K}^{-1}(1,:) - (-Q_2^{-1}(2N,1)B_{2N}^TJ_{21}Q_1^{-1}(1,2N)^T)$$

$$-Q_1^{-1}(2N,1)B_{2N}^TJ_{12}Q_2^{-1}(1,2N)$$

$$+ ([Q_2^{-1}(2N,1)B_{2N}J_{21}]Q_1^{-1}(1,2N)^T \times B_{2N}^TJ_{12}Q_1^{-1}(1,2N)^T + [I]Q_2^{-1}(2N,2N)^T)^T,$$

$$K^{-1}(r,r) = \tilde{K}^{-1}(r,r) - (-Q_1^{-1}(r,2N)B_{2N}J_{12}Q_2^{-1}(r,2N)^T)$$

$$-Q_2^{-1}(2N,1)B_{2N}^TJ_{21}Q_1^{-1}(r,2N), \quad r \leq 2N,$$

$$K^{-1}(r,r) = \tilde{K}^{-1}(r,r) - (-Q_1^{-1}(r,1)B_{2N}^TJ_{12}Q_2^{-1}(r,1)^T)$$

$$-Q_2^{-1}(2N,1)B_{2N}^TJ_{21}Q_1^{-1}(1,r), \quad r > 2N. \quad (13)$$

In order to capture all needed information for the relationships shown in (13), we assign a total of eight $N \times N$ matrix maps $\{M_i\}$ for each subproblem. Four of the maps describe the effects on the first block row and last block column, $M_{1,4}$, while the other four describe the effects on the diagonal blocks, $M_{5,8}$. The use of the maps is illustrated, for subproblems $Q_1$ and $Q_2$, in Fig. 5. Here the arrows describe the dependencies of the matrix maps in the construction of necessary entries of the inverse ($K^{-1}$).

![Fig. 5. Matrix maps.](image-url)
\[ K^{-1}(r,r) = \bar{K}^{-1}(r,r) - [Q^{-1}_2(r,1)M_5Q^{-1}_2(1,r) + Q^{-1}_2(r,1)M_6Q^{-1}_2(1,2N)^T + Q^{-1}_2(r,2N)M_7Q^{-1}_2(1,r) + Q^{-1}_2(r,2N)M_8Q^{-1}_2(r,2N)^T], \quad r > 2N. \tag{14} \]

Although each section of the relationships shown above is again implicitly divided based on geometry, the necessity of having four matrix maps for updates to each subproblems’ diagonal entry arises from the fact that the new first (last) block row (column) is a function of both the previous row and previous column associated with each subproblem.

The updates to the matrix maps for the upper subproblem \( (Q_1) \) are summarized below:

\[ M_5 - M_5^T(2B_{2N}J_{12})M_3 \to M_5, \]
\[ M_6 - M_6^T(2B_{2N}J_{12})M_4 \to M_6, \]
\[ M_7 - M_7^T(2B_{2N}J_{12})M_3 \to M_7, \]
\[ M_8 - M_8^T(2B_{2N}J_{12})M_4 \to M_8, \]
\[ M_1 + (Q^{-1}_1(1,2N)B_{2N}J_{12})M_3 \to M_1, \]
\[ M_2 + (Q^{-1}_1(1,2N)B_{2N}J_{12})M_4 \to M_2, \]
\[ (Q^{-1}_2(1,2N)B_{2N}^TJ_{12})M_3 \to M_3, \]
\[ (Q^{-1}_2(1,2N)B_{2N}^TJ_{12})M_4 \to M_4. \tag{15} \]

Those associated with the lower subproblem \( (Q_2) \) are exactly a reflection of the updates to upper subproblem:

\[ M_5 - M_5^T(2B_{2N}J_{21})M_1 \to M_5, \]
\[ M_6 - M_6^T(2B_{2N}J_{21})M_2 \to M_6, \]
\[ M_7 - M_7^T(2B_{2N}J_{21})M_1 \to M_7, \]
\[ M_8 - M_8^T(2B_{2N}J_{21})M_2 \to M_8, \]
\[ (Q^{-1}_1(1,2N)B_{2N}J_{11})M_1 \to M_1, \]
\[ (Q^{-1}_1(1,2N)B_{2N}J_{11})M_2 \to M_2, \]
\[ M_3 + (Q^{-1}_2(1,2N)B_{2N}^TJ_{21})M_1 \to M_3, \]
\[ M_4 + (Q^{-1}_2(1,2N)B_{2N}^TJ_{21})M_2 \to M_4. \tag{16} \]

The above procedure, shown in (15) and (16), for modifying the matrix maps can be recursively repeated for each of the \( (D-1) \) combining stages beginning at the lowest level with the individual subproblems. Each stage in the combining process requires the generation of the combined submatrix corner blocks through their respective matrix maps. Specifically, the subproblems that correspond to the first, last, above bridge point, and below bridge point divisions govern the required information. On completion the maps can then be used to generate the diagonal entries of \( K^{-1} \). It is important to note that this scheme fits nicely into a parallel framework due to the fact that systems handling either upper or lower subproblems would only have to trade the limited amount of information shown in (12), (15), and (16) to modify the matrix maps they are governing. Namely, each system would need only the four \( (N_s \times N_s) \) corner blocks and the bridge matrix in order to perform updates to the matrix maps for a subproblem.

**V. PARALLEL IMPLEMENTATION**

Using the divide-and-conquer framework of Sec. IV we will now provide details on the parallel implementation of the algorithm. The process begins by initializing the program on \( p \) computers. Each computer is then assigned a subset from the \( D \) divisions of the matrix \( K \). The sizes of the divisions and the number of divisions assigned to each computer are determined in order to meet memory requirements, while aiming to decompose the problem as evenly as possible. For each computer “governing” flags are used to indicate which of the subproblems have been assigned. Given this information, each computer loads the proper sections of the matrix \( K \). Specifically, the matrices encountered in the simulation of nanotransistors will have the sparsity structure shown in Fig. 6. The matrix under consideration is complex valued and symmetric but not Hermitian. It can be specified by three diagonals and two dense \( (N_s \times N_s) \) boundary condition blocks, due to coupling with the device contacts, assigned to the first and last divisions.

Next, a list of the \( (D-1) \) combining stages needed in order to form the global solution is created. Each entry in the list specifies the first, last, and bridge point of the combining stage, i.e., for the example in Sec. IV B the entry would be of the form \([\text{start}, \text{stop}, \text{bridge}_\text{point}]=[1,4,2]\) Before the ac-
cumulation of the combining effects can begin, each of the \( p \) computers first determines the corner blocks of the inverses for all divisions that it governs. This is a two step process beginning with computation of the compact representation of the inverse for each submatrix. This representation can then be used to solve for the diagonal blocks and the remaining corner block, i.e., if we consider the first block row of the inverse for submatrix \( j \):

\[
\phi_j^{-1}(1,:) = (U_1V_1^T U_1V_2^T \cdots U_1V_N^T)
\]

\[
= \phi_j^{-1}(1,1) \phi_j^{-1}(1,1)S_1 \cdots \phi_j^{-1}(1,1)
\]

\[
\times \left( \prod_{k=1}^{N-2} S_k \phi_j^{-1}(1,N) \right).
\]

Next, the matrix maps for each subproblem will be modified according to the procedure in Sec. IV C. This process begins by looping through the combination list to identify any combining stages involving divisions being governed. For any such encountered, the computer is responsible for generating any necessary information for all computers involved in that stage to update their maps. Therefore, the only transmission of information is that necessary to form the adjustment matrix (12) and that required to perform the operations shown in (15) and (16). Specifically, there are four \((N_x \times N_y)\) matrices required for the update of matrix maps for any combining stage. The computers can easily distinguish their responsibilities based on the location of the division within the combination list entry. Computers which are responsible for divisions with indices, start, stop, bridge_point, and bridge_point+1, must use their maps to generate the required blocks and distribute them to all computers within the range [start, stop]. This communication for the case \( D=4 \) and \( p=4 \) is illustrated in Fig. 7. Here, stage 1 represents the capturing of the combining effects between computers I and II and computers III and IV, where the shaded blocks represent the bridge matrices between the respective divisions. Stage 2 shows the pseudocombined problems, shown by the dotted lines, for which the subproblem matrix maps can be used to create the information necessary for the combination across computers I–IV.

Finally, each computer can apply the matrix maps for each division governed to update the diagonal entries of the inverse for each of the submatrices. The new diagonal entries are then gathered to form the global solution. A summary of the implementation is detailed below. The time complexity of the algorithm presented is \( O(N_x^2 N_y/p + N_y^3 \log_2 D) \), with memory consumption \( O(N_x N_y/D + (N_y^2 D/p)) \). The first term \((N_x^2 N_y/p)\) in the computational complexity arises from the embarrassingly parallel nature of both determining the ratio sequences and applying the matrix maps to update the diagonal entries. The second term \((N_y^3 \log_2 D)\) is dependent on the number of levels needed to gather combining information for \( D \) subproblems, i.e., updates to the matrix maps for each division governed. Similarly, the first term in the memory complexity is due to the ratio sequences and diagonal blocks, and the second represents the memory required for the matrix maps of each subproblem governed.

**Pseudo-Code**

1. Decompose the matrix \( K \) into \( D \) divisions
2. Each subproblem from \( \{1, 2, \ldots, D\} \) is assigned to a single computer in \( \{1, 2, \ldots, p\} \):
   - Set governing flags on each computer:
     - \( \text{TRUE} \) if computer governs subproblem \( i \)
     - \( \text{FALSE} \) else
3. For each governed subproblem a computer will independently:
   - Determine ratio sequences
   - Determine diagonal blocks of inverse
   - Construct corner blocks of inverse
4. Generate list of subproblem combinations:
   - \( \text{size}=1 \)
   - \( k=0 \)
   - while \( (\text{size}<D) \)
     - \( i=0 \)
     - while \( (i+\text{size}<D) \)
       - \( k=k+1 \)
       - \( \text{list}(k) \index=[i+1, \min(i+2*\text{size}, D), i+\text{size}] \)
     - \( i=i+2*\text{size} \)
   - \( \text{size}=\text{size}+2 \)
5. Each computer will adjust maps for any combining step involving a governed subproblem:
   - \( k=1 \)
   - while \( (k<D) \)
     - \( \text{list}(k) \index=[\text{start}, \text{stop}, \text{bridge_point}]=\text{list}(k).\index \)
     - if \( \text{gov}([\text{start}]) \lor \text{gov}([\text{start}+1]) \lor \cdots \lor \text{gov}([\text{stop}]) \)
       - \( \text{TRUE} \)
         - \( \text{if} \ (\text{gov}([\text{start}]) \lor \text{gov}([\text{stop}]) \lor \text{gov}([\text{bridge_point}]) \lor \text{gov}([\text{bridge_point}+1]) = \text{TRUE} ) \)
           - Generate all governed corner blocks according to (14)
         - Gather all corner blocks from: start, stop, bridge_point, and bridge_point+1
       - Adjust matrix maps for governed divisions
     - \( k=k+1 \)
6. For each governed subproblem a computer will independently:

- Apply matrix maps to transform subproblem solutions into corresponding section of the global solution

7. Gather updated local solutions to form global solution

VI. RESULTS AND DISCUSSION

A. Preface

The algorithm in Ref. 4 calculates the diagonal blocks \((D_i)\) of \(K^{-1}\) in the following manner:

\[
G_1 = A_1^{-1},
\]

\[
G_{i+1} = (A_{i+1} - B_iG_iB_i)^{-1}, \quad i = 1, 2, \ldots, N_y - 1,
\]

\[
D_{Ny} = G_{Ny}.
\]

The time complexity of this algorithm [we will refer to as recursive Green’s function (RGF) approach] was shown to be \(O(N_x^2 N_y)\), with a memory requirement of \(O(N_x^2 N_y)\). It is important to note that each recursion, i.e., (7) and (17), can be used to determine both the diagonal and off-diagonal blocks of \(K^{-1}\) in the same computational and memory complexities. In addition, the RGF method can be extended in a similar fashion to compute additional entries of the inverse of \(K\). Therefore, the use of the ratio sequences to describe \(K^{-1}\) is chosen based on simplicity of representation, with respect to the divide-and-conquer algorithm, but should be thought of as completely analogous to the formulation shown in (17).

B. Atomistic simulation of silicon nanowires

We begin with the investigation of an application necessitating either approximation techniques, such as the wave function based method considered in Ref. 12 or special purpose hardware for simulation. Our divide-and-conquer algorithm offers a direct Green’s function approach that is appli-
cable to these nanoscale device simulations. Through the general framework shown in Ref. 8 we can construct the retarded Green’s function matrix equation:

$$KG' = (EI - H - \Sigma_1 - \Sigma_2)G' = I$$  \hspace{1cm} (18)

for a Si [100] nanowire. Here $E$ is the energy of interest, $H$ is the Hamiltonian which includes all the nearest neighbor interactions, and $\Sigma_{1,2}$ are the boundary condition matrices. Since only interactions between successive layers of the assumed lattice structure are included, the Hamiltonian will be a block tridiagonal matrix. In the case of Si in the [100] direction the block sizes will repeat every four layers, as can be seen from the example shown in Fig. 8. In addition to the density of states, the algorithm presented here can be used to calculate the transmission

$$T = \text{Trace}[\Gamma_1 G' \Gamma_2 G'^*]$$

$$\Gamma_{1,2} = i[\Sigma_{1,2} - \Sigma_{1,2}^*].$$  \hspace{1cm} (19)

This can be seen easily by recalling that the matrix map formulation shown in Sec. IV C allows for the first block row and last block column to be produced at any combining formulation shown in Sec. IV C allows for the first block row and last block column to be produced at any combining stage. Therefore, all information needed for the calculation of (19) is explicitly known.

Initially, we simulate a small square cross-section silicon nanowire of size $2 \times 2 \times 20.6 \text{ nm}^3$, considering ten orbitals per atom, across 90 energy points. The transmission and density of states (DOS) are shown in Fig. 9 and were generated using $D = p = 16$. Although this example requires less than 1 Gbyte of memory on a single processor, the availability of our algorithm becomes crucial when considering even modest device sizes. For example, a computationally efficient implementation of RGF would require approximately 3 Gbytes of memory for a $2.5 \times 2.5 \times 102.6 \text{ nm}^3$ nanowire and 6 Gbytes for a $3 \times 3 \times 102.6 \text{ nm}^3$ nanowire. For the latter case, this would correspond to a simulation with 50,085 atoms, considering the geometry of $N_x = \{720, 600, 720, 610\}$ and $N_z = 756$. Our algorithm facilitates simulation for these devices without the use of special purpose hardware. The DOS and transmission for these geometries are shown in Figs. 10 and 11, again assuming $D = p = 16$. It is important to note that these simulations considered only ten atomic orbitals; if spin was considered the memory requirements described above would increase by a factor of 4. We conclude that the parallel divide-and-conquer algorithm presented here sets the foundation for a framework under which a wide range of nanoscale device simulations can be performed. An analysis of the computational performance for our algorithm is provided in the following sections.

### C. Peak performance

In order to compare our algorithm against the current state of the art, we consider the simulation of nanotransistors based on a MIT well-tempered 25 nm device-like structure. Our algorithm (PDIV, Parallel Divide-and-Conquer) along with RGF have been implemented, in C, on a cluster consisting of 3.02 or 3.2 GHz Intel Xeon and Nocona/Irwindale processors.

To demonstrate the scaling of our algorithm it was first necessary to determine the largest sized problem that can be solved on a single workstation using RGF. For this cluster, ($N_x, N_z$)=(100,4992) was the largest problem representative of the physical dimensions with respect to an actual nanotransistor simulation, i.e., the necessary number of grid points along the depth and the length of the device. Table I shows the runtime of RGF and PDIV implemented for $p = 2, 4, \ldots, 64$. Here, the number of divisions is given by $D = N_z/p$, the single computer case ($p=1$) uses RGF, and the remaining cases ($p>1$) utilize our parallel algorithm. With the use of four computers our algorithm’s computation time was superior to the time required for RGF. The further improvement in time decays as the number of computers $p$ increases, which is consistent with increases in communication costs. This artifact can also be seen from the computational complexity of $O[(N^3_x N_z/p)+N^3_z \log_2 D]$. As $p(=D)$ increases, the second term in the computational cost begins to dominate, resulting in the gradual decay of speedup shown in Table I.

<table>
<thead>
<tr>
<th>$p$</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (s)</td>
<td>137.88</td>
<td>168.99</td>
<td>85.40</td>
<td>42.57</td>
<td>22.95</td>
</tr>
<tr>
<td>Speedup</td>
<td>1</td>
<td>0.81</td>
<td>1.61</td>
<td>3.23</td>
<td>6.00</td>
</tr>
</tbody>
</table>

Next, to show the peak performance of our algorithm it is necessary to run larger cases than those realizable on a single workstation using RGF. In order to compare run times we were able to extrapolate an approximate time for RGF by using the fact that the time complexity scaled linearly with respect to $N_z$, i.e., $O(N^2_x N_z)$. Table II shows the speedup for $p=4, 8, \ldots, 64$ while keeping the ratio of grid points constant, i.e., $D = p = N_z/400$. As can be seen in Table II, for the case $p=64$, PDIV was able to achieve a speedup of 20.90× as compared to the time estimate of RGF.

### Final remarks

Finally, while it seems clear that the addition of computing resources in a distributed computing environment will benefit not only computation time but memory consumption
as well, the flexibility to break apart the problem into more divisions should not be overlooked (i.e., the case $D > p$). For example, assume that the memory consumption of a particular problem exceeds the computer’s resources. Instead of increasing the number of processors ($p$), the number of divisions ($D$) on each computer can be increased in order to meet the memory requirements for the workstations by trading off computation time. In order to demonstrate this ability, the case $(N_x, N_y) = (100, 512\,000)$ with $D = 128$ was run across 64 computers. This translates into finding the diagonal entries of the inverse of a matrix of size $N_x \times N_y > 5 \times 10^7$. This is greater than 100 times the size of the largest problem which could be run on a single computer $(N_x \times N_y = 5 \times 10^5)$, while only using 64 computers. In general, if we exclude the cost of communication, the problem should be separated into $D \geq p$ subproblems subject to the memory constraints imposed by only using $p$ computers. In the case of $D > p$, multiple subproblems should be assigned to the computers based on spatial location in order to reduce unnecessary communication.

**D. Scalability**

While it seems clear from Sec. VI C that the divide-and-conquer algorithm offers benefits in both memory and computational complexities, several important questions remain. Specifically, with any numerical method there is always a question of accuracy when dealing with real data as opposed to simulated. Secondly, with any parallel algorithm the sensitivity to the scaling of computation time due to communication costs needs to be examined. In order to validate the algorithm, the density of states (DOS) was computed using both the PDIV and RGF methods and found to be within machine epsilon for all comparable cases. In addition, to confirm the computational complexity and further examine the scalability of the algorithm several other examples were analyzed.

Figure 12 shows the time needed to find DOS for the case $N_y = 100$, while varying $N_x$ for a specific values of $p$. It is important to note that the times considered are only those which offer improvement by the addition of computing resources, i.e., the point $(N_x, N_y) = (100, 2496)$ is not shown for

**FIG. 12.** (Color online) Time for PDIV for varying $N_y$ with $N_x = 100$ and $p = D = 2, 4, \ldots, 32$.

**FIG. 13.** (Color online) Speedup for PDIV for varying $N_y$ with $N_x = 100$ and $p = D = 2, 4, \ldots, 32$.

**FIG. 14.** (Color online) Time for PDIV for varying $N_y$ with $N_x = 624$ and $p = D = 2, 4, \ldots, 32$.

**FIG. 15.** (Color online) Speedup for PDIV for varying $N_y$ with $N_x = 624$ and $p = D = 2, 4, \ldots, 32$. 
The trade-offs between computation time, memory consumption, and computing resources can all be clearly seen from Fig. 12. Specifically, the trade-off between computation time and number of processors is emphasized in Fig. 13. The figure shows the speedup determined by the ratio of time required for PDIV to that of RGF, across varying values of \( p \). As the problem size increases, i.e., \( N_y \) increases, the relative reduction of the communication costs can be seen by its corresponding positive effect to the scalability of the algorithm.

Figures 14 and 15 show the same analyses for the case \( N_y = 624 \) and varying \( N_x \).

**ACKNOWLEDGMENTS**

We would like to acknowledge the contribution of M. P. Anantram, Senior Research Scientist NASA Ames, for his invaluable expertise in the area of nanosciences. We would also like to acknowledge the contributions of Mathieu Luisier (ETH Zurich), Rezwanuzzaman Chowdhury (Purdue University), and Gerhard Klimeck (NCN Purdue University) for providing nanowire simulation resources. In addition, we would like to acknowledge the contribution of Y. Charlie Hu (Purdue University) for providing insight into parallel computing techniques. This material is based on the work supported by the NASA, under Award No. NCC 2-1363, and the Semiconductor Research Corporation (SRC). The work was also supported by the National Science Foundation and Purdue University through cost-sharing to the Network for Computational Nanotechnology. Computational resources from nanoHUB.org were used in this work.

11. J. Jain, S. Cauley, H. Li, C.-K. Koh, and V. Balakrishnan, Purdue ECE 07–17.