A Scalable Distributed Method for Quantum-Scale Device Simulation

Stephen Cauley, Jitesh Jain, Cheng-Kok Koh, and Venkataramanan Balakrishnan
School of Electrical and Computer Engineering
Purdue University, West Lafayette, IN 47907-1285
{stcauley,jjain,chengkok,ragu}@ecn.purdue.edu

Abstract

We present an algorithm for the fast and accurate simulation of nano-scale devices not attainable using current techniques. The idea underlying the algorithm is a novel divide-and-conquer method based on the non-equilibrium 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 speed-up, as compared to the current state of the art algorithm. We conclude that 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.

1 Introduction

The accurate and efficient modeling and simulation of nano-scale devices has become a problem of increasing importance. Due to the scaling of device sizes, previously employed macro-modeling 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 non-equilibrium 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 [3, 6, 10] 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. \(10^6 \times 10^6\)).
In [13], 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;
- 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 [13]. 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 [8, 4], and the two-dimensional simulation of nanotransistors. We conclude that our algorithm allows for the simulation of nano-scale devices not attainable with current techniques.

2 Problem Description

Consider the two-dimensional model of a nano-scale transistor, shown in Fig. 1. The body of the transistor is projected onto a two-dimensional non-uniform spatial grid of dimension $N_x \times N_y$, 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 $N_x$ and $N_y$ could result in problem sizes ($N_x \cdot N_y$) 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:

$$H_b(r) = -\frac{\hbar^2}{2m^b_z} \left( \frac{d}{dx} \left( \frac{1}{m^b_x} \frac{d}{dx} \right) + \frac{1}{m^b_y} \frac{d}{dy} \frac{d}{dy} + \frac{1}{m^b_z} \frac{d}{dz} \frac{d}{dz} \right) + V(r),$$

(1)

where $(m^b_x, m^b_y, m^b_z)$ are the components of effective mass in valley $b$. The equation of motion for the retarded Green’s function ($G'_b$) and less-than Green’s function ($G_b^<$) are [5]:

$$\begin{align*}
    &\left[ E - \frac{\hbar^2 k^2_z}{2m_z} - H_b(r_1) \right] G'_b(r_1, r_2, k_z, E) - \int dr \Sigma'_b(r_1, r_2, k_z, E) G'_b(r_1, r_2, k_z, E) = \delta(r_1 - r_2), \\
    &= \int dr \Sigma'_b(r_1, r_2, k_z, E) G'_b(r_1, r_2, k_z, E). 
\end{align*}$$

(2)

$$\begin{align*}
    &\left[ E - \frac{\hbar^2 k^2_z}{2m_z} - H_b(r_1) \right] G_b^<(r_1, r_2, k_z, E) - \int dr \Sigma_b^<(r_1, r_2, k_z, E) G_b^<(r_1, r_2, k_z, E) \\
    &= \int dr \Sigma_b^<(r_1, r_2, k_z, E) G_b^<(r_1, r_2, k_z, E). 
\end{align*}$$

(3)

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

$$N(r, k_z, E) = \sum_b N_b(r, k_z, E) = -\frac{1}{\pi} \text{Im} \left[ G'_b(r, r, k_z, E) \right],$$

(4)

$$\rho(r, k_z, E) = \sum_b \rho_b(r, k_z, E) = -i \left[ G_b^<(r, r, k_z, E) \right].$$

(5)

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 Fig. 2 and Fig. 3. In the case of the sp\(^3\)d\(^5\)s\(^*\) model, 10 orbitals are considered, 20 if spin orbit coupling is included. For this case the value of \(N_x\) is dependent on the lattice structure, i.e. the number of atoms and orbitals in a given layer, and the value of \(N_y\) 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 nano-scale devices, the self-consistent solution of the Green’s function is often the most-time intensive step in the simulation process. In [13], 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 [9] 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 \(G^r\), all results shown herein can be easily extended to the case of the less-than Green’s function \(G^<\).

### 3 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 \(G^r\) [5], where \(KG^r = I\) and \(K\) is a block tridiagonal matrix of the form

\[
K = \begin{pmatrix}
A_1 & -B_1 & & & \\
-B_1^T & A_2 & -B_2 & & \\
&& \ddots & \ddots & \\
& -B_{N_y-2} & A_{N_y-1} & -B_{N_y-1} & \\
& -B_{N_y-1}^T & A_{N_y} & & \\
\end{pmatrix},
\]

where each \(A_i, B_i \in \mathbb{C}^{N_x \times N_x}\). Thus \(K \in \mathbb{C}^{N_y N_x \times N_y N_x}\), with \(N_y\) diagonal blocks of size \(N_x\) each. We will use the notation \(K = \text{tri}(A_{1:N_y}, B_{1:N_y-1})\) 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 \(G^r\) in the case of electron densities) offers new computational challenges.

The inverse of a block tridiagonal matrix can be computed explicitly, as demonstrated in [11]. Specifically, there exists two (non-unique) sequences of matrices \(\{U_i\}, \{V_i\}\) such that for \(j \geq i\), \((K^{-1})_{ij} = U_i V_j^T\).
Consider the decomposition of the block tridiagonal matrix 

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

To address these issues we offer a divide-and-conquer approach through the use of domain decomposition.

where the diagonal blocks of the inverse, \( D \), correction term, demonstrated below:

\[ \text{Hence, } K^{-1} \text{ can be written as:} \]

\[ K^{-1} = \begin{pmatrix}
D_1 & D_1 S_1 & D_1 S_1 S_2 & \cdots & D_1 \prod_{i=1}^{N_y-1} S_i \\
R_1 D_1 & D_2 & D_2 S_2 & \cdots & D_2 \prod_{i=2}^{N_y-1} S_i \\
R_2 R_1 D_1 & R_2 D_2 & D_3 & \cdots & D_3 \prod_{i=3}^{N_y-1} S_i \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
(\prod_{i=N_y-1}^{1} R_i) D_1 & (\prod_{i=N_y-1}^{2} R_i) D_2 & (\prod_{i=N_y-1}^{3} R_i) D_3 & \cdots & D_N_y \\
\end{pmatrix}. \]

The above result has also been shown in [10] and [2]. The \( U \) and \( V \) sequences can be constructed in a numerically stable fashion as demonstrated in [7]. Specifically, there exists two sequences of “ratio” matrices \( \{R_i\}, \{S_i\} \) such that the inverse of \( K \) can be written as:

\[ R_1 = A_1^{-1} B_1, \quad R_i = (A_{i-1} - B_{i-2}^T R_{i-1})^{-1} B_{i-1}, \quad i = 2, \ldots, N_y - 1, \]

\[ S_{N_y-1} = B_{N_y-1} A_{N_y-1}^{-1}, \quad S_i = B_i (A_{i+1} - S_{i+1} B_{i+1}^T)^{-1}, \quad i = N_y - 2, \ldots, 1, \]

\[ D_1 = (A_1 - B_1 S_1^T)^{-1}, \quad D_{i+1} = (A_{i+1} - B_{i+1} S_{i+1}^T)^{-1} (I + B_i T_{i}^T D_i S_i), \quad i = 1, \ldots, N_y - 2, \]

\[ D_{N_y} = A_{N_y}^{-1} (I + B_{N_y-1}^T D_{N_y-1} S_{N_y-1}). \]

The time complexity associated with determining the ratio sequences \( R_i \) and \( S_i \) is \( O(N_x^2 N_y) \), with a memory requirement of \( O(N_x^2 N_y) \). As was alluded to in §1, accurate simulation of realistic devices often involves a large number of grid-points resulting in prohibitive memory and computational requirements. To address these issues we offer a divide-and-conquer approach through the use of domain decomposition. Consider the decomposition of the block tridiagonal matrix \( K \) into two block tridiagonal sub-matrices and a correction term, demonstrated below:

\[ K = \begin{pmatrix}
\phi_1 \\
\phi_2 \\
\end{pmatrix} + XY, \]

\[ K = \begin{pmatrix}
\phi_1 \\
\phi_2 \\
\end{pmatrix}. \]
where
\[ \phi_1 = \text{tri}(A_{1;i}, B_{1;j-1}), \quad \phi_2 = \text{tri}(A_{i+1:N_i;B_{i+1:N_i}}, \quad \text{and} \]
\[ X = \begin{pmatrix} 0 & \cdots & -B_i^T & 0 & \cdots & 0 \\ 0 & \cdots & 0 & -B_i & \cdots & 0 \end{pmatrix}^T, \quad Y = \begin{pmatrix} 0 & \cdots & 0 & I & \cdots & 0 \\ 0 & \cdots & I & 0 & \cdots & 0 \end{pmatrix}. \]

Thus, the original block tridiagonal matrix can be decomposed into the sum of a block diagonal matrix (with its two diagonal blocks themselves being block tridiagonal) and a correction term parameterized by the \( N_i \times N_i \) matrix \( B_i \), which we will denote as the “bridge matrix”.

Using the standard matrix inversion lemma from linear algebra, we have
\[ K^{-1} = (\bar{K} + XY)^{-1} = \bar{K}^{-1} - (\bar{K}^{-1}X)(I + Y\bar{K}^{-1}X)^{-1}(Y\bar{K}^{-1}), \]
where
\[ \bar{K}^{-1}X = \begin{pmatrix} -\phi_1^{-1}(i,i)B_i & 0 \\ 0 & -\phi_2^{-1}(i,1)B_i^T \end{pmatrix}, \]
\[ (I + Y\bar{K}^{-1}X)^{-1} = \begin{pmatrix} I & -\phi_2^{-1}(1,1)B_i^T \\ -\phi_1^{-1}(i,i)B_i & I \end{pmatrix}^{-1}, \]
\[ Y\bar{K}^{-1} = \begin{pmatrix} 0 & \phi_2^{-1}(i,1)^T \\ \phi_1^{-1}(i,1)^T & 0 \end{pmatrix}, \]
and \( \phi_1^{-1}(i,i) \) and \( \phi_2^{-1}(i,1) \) denote respectively the last and first block columns of \( \phi_1^{-1} \) and \( \phi_2^{-1} \).

Expression (8) illustrates how the inversion of a block tridiagonal matrix can be divided into smaller problems, whose solutions can then be combined to solve the original problem. Recall that in the context of (4), “solving” the original problem refers to finding only the diagonal entries of \( K^{-1} \). Due to the fact that \( \bar{K}^{-1} \) is block diagonal, the diagonal entries of \( K^{-1} \) can be found by modifying the diagonal entries of each sub-matrix by a portion of the second term on the right-hand side of (8). Namely, the solution of each sub-problem need only be modified by the diagonal entries of

\[ \begin{pmatrix} \bar{K}^{-1}X & (I + Y\bar{K}^{-1}X)^{-1} \end{pmatrix} \]
\[ = \begin{pmatrix} \phi_1^{-1}(i,i)B_i & 0 \\ 0 & \phi_2^{-1}(i,1)B_i^T \end{pmatrix} \begin{pmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{pmatrix} \begin{pmatrix} \phi_1^{-1}(i,i)^T & 0 \\ \phi_2^{-1}(i,1)^T & 0 \end{pmatrix}, \]

where
\[ J = \begin{pmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{pmatrix} = (I + Y\bar{K}^{-1}X)^{-1}. \]

It is important to note that the diagonal entries of (9) are divided implicitly based upon which sub-problem solution they are modifying, e.g. the \( k^{th} \) diagonal entry for the solution of sub-problem \( 1 \) is
modified by the term $\phi_1^{-1} (k,i) B_i J_1 \phi_1^{-1} (k,i)^T$ in $O(N_2^2)$ operations. Furthermore, the “adjustment” term, $(I + YK^{-1}X)^{-1}$, depends only on corner blocks, i.e. $\phi_2^{-1} (1,1)$ and $\phi_1^{-1} (i,i)$, of the inverses of the sub-matrices. 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 sub-problem solutions in both a memory and computationally efficient manner.

4 Divide-and-Conquer Algorithm

4.1 Overview

The procedure begins with separating the block tridiagonal matrix $K$ into $D$ sub-matrices, each of which are 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 sub-problem 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 computation. Adjustments to the matrix maps at each combining stage are not constant and must be modified to follow the procedure (8), detailed in §3. These maps can then be used in the final stage to transform the sub-problem solutions into the global solution.

Matrix maps are created to produce the cumulative effect of each combining step associated with a particular sub-problem. In this context the term “cumulative effect” refers, not only to the adjustment of the diagonal entries of a particular sub-matrix into a corresponding section of the global, but also the ability to outwardly project, from a given sub-problem, 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 $\phi_{1 \sim 2}$ and $\phi_{3 \sim 4}$. For simplicity of illustration we will assume each sub-problem to be of equal size, where $N$ denotes the number of blocks for the corresponding sub-matrices. Thus, the two sub-matrices $\phi_1$ and $\phi_2$ can be used to form $\phi_{1 \sim 2}$ via the bridge matrix $B_N$. The
first (last) block row (column) of $\phi_{1 \sim 2}$ can then be described in the following manner:

$$
\phi_{1 \sim 2}^{-1}(1,:) = \tilde{\phi}_{1 \sim 2}^{-1}(1,:) - (-\phi_{1}^{-1}(1,N)B_{N}J_{12}\phi_{1}^{-1}(::,N)^T - \phi_{1}^{-1}(1,N)B_{N}J_{11}\phi_{2}^{-1}(1,:)),
$$

(10)

$$
\phi_{1 \sim 2}^{-1}(::,2N) = \tilde{\phi}_{1 \sim 2}^{-1}(::,2N) - (-\phi_{2}^{-1}(N,1)B_{N}^{T}J_{22}\phi_{1}^{-1}(::,N)^T - \phi_{2}^{-1}(N,1)B_{N}^{T}J_{21}\phi_{2}^{-1}(1,:))^T,
$$

and the $r^{th}$ diagonal block of $\phi_{1 \sim 2}^{-1}$ can be described using the following relationships:

$$
\phi_{1 \sim 2}^{-1}(r,r) = \tilde{\phi}_{1 \sim 2}^{-1}(r,r) - (-\phi_{1}^{-1}(r,N)B_{N}J_{12}\phi_{1}^{-1}(r,N)^T)
\quad r \leq N,
$$

(11)

$$
\phi_{1 \sim 2}^{-1}(r,r) = \tilde{\phi}_{1 \sim 2}^{-1}(r,r) - (-\phi_{2}^{-1}(r,1)B_{N}^{T}J_{21}\phi_{2}^{-1}(1,r))^T
\quad r > N,
$$

given the adjustment matrix:

$$
J = \begin{pmatrix}
I & -\phi_{1}^{-1}(1,1)B_{N}^{T} \\
-\phi_{1}^{-1}(N,N)B_{N} & I
\end{pmatrix}^{-1}.
$$

Similar to the example from §3 the term $\tilde{\phi}_{1 \sim 2}^{-1}$ represents the combination of the individual sub-matrix inverses. It is important to note that all the updates shown above are dependent only on the first (last) block row (column) of the sub-problems, only as a function of the sub-matrix 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 sub-problems, the necessity for the preservation of the effects to the first block row and last block column (which includes all corner blocks of $\phi_{1 \sim 2}$) is demonstrated in the following section.

4.2 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 sub-matrix. 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 $Q_{1} = \phi_{1 \sim 2}$ and $Q_{2} = \phi_{3 \sim 4}$. 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.
It should be noted that both \( Q_1^{-1} \) and \( Q_2^{-1} \) will have formulation (7) since \( \phi_{1 \sim 2} \) and \( \phi_{3 \sim 4} \) 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 braces above, i.e. the corner blocks for each of the joined sub-matrices. 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 sub-problem solution into a given section of the global.

### 4.3 Matrix Maps

The effects of combining multiple sub-problems will be captured, in a recursive fashion, through the use of matrix maps. These linear mappings facilitate the generation, through the use of sub-problem ratio sequences (7), of any information necessary to propagate the sub-problem 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 sub-problems and those to Lower sub-problems, 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 §4.2. The adjustment matrix, \( J \), for the combining step is defined as follows:

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

\[
P = (I - Z_1 Z_2)^{-1},
\]

\[
J = \begin{pmatrix} P & -PZ_1 \\ -Z_2 P & I + Z_2 P Z_1 \end{pmatrix} = \begin{pmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{pmatrix}.
\]

The process of combining sub-problems \( Q_1 \) 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:

\[
K^{-1}(1,:) = \tilde{K}^{-1}(1,:) - ( -Q_1^{-1}(1,2N)B_{2N}J_{12}Q_1^{-1}(,2N)^T - Q_1^{-1}(1,2N)B_{2N}J_{11}Q_2^{-1}(1,:))
\]

\[
= ([I]Q_1^{-1}(1,:) + [Q_1^{-1}(1,2N)B_{2N}J_{12}]Q_1^{-1}(,2N)^T [Q_1^{-1}(1,2N)B_{2N}J_{11}]Q_2^{-1}(1,:),
\]
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 sub-problem. 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 sub-problems \(Q_1\) 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 \((K^{-1})\).

Initially,

\[
M_i = \begin{cases} 
0 & i = 2, 3, 5, 6, 7, 8 \\
I & i = 1, 4
\end{cases}
\]

for each of the sub-problems. The matrices \(M_1\) and \(M_4\) are initialized to the identity due to the fact that when combining sub-problems at the lowest level the Upper sub-problem contains existing first block row information and the Lower sub-problem contains existing last block column information. The dependencies for the relationships shown in (13) can be written explicitly in terms of the matrix maps for sub-problems \(Q_1\) and \(Q_2\), as follows:

\[
K^{-1}(r) = \tilde{K}^{-1}(r) - \left(-Q_2^{-1} (2N, 1) B_{2N} J_{22} Q_1^{-1} (2N) + Q_2^{-1} (2N, 1) B_{2N} J_{21} Q_2^{-1} (1)\right)^T
\]

\[
K^{-1}(r, r) = \tilde{K}^{-1}(r, r) - \left(-Q_2^{-1} (r, 2N) B_{2N} J_{12} Q_1^{-1} (r, 2N)\right)^T \quad r \leq 2N,
\]

\[
K^{-1}(r, r) = \tilde{K}^{-1}(r, r) - \left(-Q_2^{-1} (r, 1) B_{2N} J_{21} Q_2^{-1} (r, 1)\right)^T \quad r > 2N.
\]
\[ K^{-1}(1,:) = (M_{1:Q_1}Q_1^{-1}(1,:) + M_{2:Q_1}Q_2^{-1}(1,:; 2N)^T, \quad M_{1:Q_2}Q_2^{-1}(1,:) + M_{2:Q_2}Q_2^{-1}(1,:; 2N)^T), \]

\[ K^{-1}(::4N) = (M_{3:Q_1}Q_1^{-1}(1,:) + M_{4:Q_1}Q_1^{-1}(1,:; 2N)^T, \quad M_{3:Q_2}Q_2^{-1}(1,:) + M_{4:Q_2}Q_2^{-1}(1,:; 2N)^T)^T, \]

\[ K^{-1}(r,r) = \tilde{K}^{-1}(r,r) - [Q_1^{-1}(r,1)M_{5:Q_1}Q_1^{-1}(1,r) + Q_1^{-1}(r,1)M_{6:Q_1}Q_1^{-1}(r,2N)^T + Q_1^{-1}(r,2N)M_{7:Q_1}Q_1^{-1}(1,r) + Q_1^{-1}(r,2N)M_{8:Q_1}Q_1^{-1}(r,2N)^T] \quad r \leq 2N, \]

\[ K^{-1}(r,r) = \tilde{K}^{-1}(r,r) - [Q_2^{-1}(r,1)M_{5:Q_2}Q_2^{-1}(1,r) + Q_2^{-1}(r,1)M_{6:Q_2}Q_2^{-1}(r,2N)^T + Q_2^{-1}(r,2N)M_{7:Q_2}Q_2^{-1}(1,r) + Q_2^{-1}(r,2N)M_{8:Q_2}Q_2^{-1}(r,2N)^T] \quad r > 2N. \]

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

The updates to the matrix maps for the Upper sub-problem \((Q_1)\) is summarized below:

\[ M_5 - M_5^T (B_{2N}J_{12})M_3 \rightarrow M_5, \]
\[ M_6 - M_6^T (B_{2N}J_{12})M_4 \rightarrow M_6, \]
\[ M_7 - M_7^T (B_{2N}J_{12})M_3 \rightarrow M_7, \]
\[ M_8 - M_8^T (B_{2N}J_{12})M_4 \rightarrow M_8, \]
\[ M_4 + (Q_1^{-1}(1,2N)B_{2N}J_{12})M_3 \rightarrow M_1, \]
\[ M_2 + (Q_1^{-1}(1,2N)B_{2N}J_{12})M_4 \rightarrow M_2, \]
\[ (Q_2^{-1}(1,2N)B_{2N}J_{22})M_3 \rightarrow M_3, \]
\[ (Q_2^{-1}(1,2N)B_{2N}J_{22})M_4 \rightarrow M_4. \]

Those associated with the Lower sub-problem \((Q_2)\) are exactly a reflection of the updates to Upper...
sub-problem:

\[ M_5 - M_1^T (B_{2NJ21}^2) M_1 \rightarrow M_5, \]
\[ M_6 - M_1^T (B_{2NJ21}^2) M_2 \rightarrow M_6, \]
\[ M_7 - M_2^T (B_{2NJ21}^2) M_1 \rightarrow M_7, \]
\[ M_8 - M_2^T (B_{2NJ21}^2) M_2 \rightarrow M_8, \]
\[ (Q_1^{-1})(1,2N)B_{2NJ11}^2 M_1 \rightarrow M_1, \]
\[ (Q_2^{-1})(1,2N)B_{2NJ11}^2 M_2 \rightarrow M_2, \]
\[ M_3 + (Q_2^{-1})(1,2N)B_{2NJ21}^2 M_1 \rightarrow M_3, \]
\[ M_4 + (Q_2^{-1})(1,2N)B_{2NJ21}^2 M_2 \rightarrow M_4. \]  

(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 sub-problems. Each stage in the combining process requires the generation of the combined sub-matrix corner blocks, through their respective matrix maps. Specifically, the sub-problems who 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 sub-problems 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 \((N_x \times N_x)\) corner blocks and the bridge matrix in order to perform updates to the matrix maps for a sub-problem.

5 Parallel Implementation

Using the divide-and-conquer framework of §4 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 sub-problems 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 3 diagonals and two dense \((N_x \times N_x)\) 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 §4.2 the entry would be of the form: \([\text{start}, \text{stop}, \text{bridge point}] = [1, 4, 2]\). Before the accumulation 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 sub-matrix. 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 sub-matrix \(j\):
Figure 6: Structured matrix associated with nanotransistor ($K$).

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

\[
= \left( \phi_j^{-1}(1,1) \quad \phi_j^{-1}(1,1)S_1 \quad \cdot \cdot \cdot \quad \phi_j^{-1}(1,1)(\prod_{k=1}^{N-2} S_k) \quad \phi_j^{-1}(1,N) \right).
\]

Next, the matrix maps for each sub-problem will be modified according to the procedure in section §4.3. 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 ($N_x \times N_x$) matrices required for the update of matrix maps for any combining stage. The computers can easily distinguish their responsibilities based upon 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 & II and computers III & IV, where the shaded blocks represent the bridge matrices between the respective divisions. Stage 2 shows the pseudo-combined problems, shown by the dotted lines, for which the sub-problem matrix maps 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 sub-matrices. 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 $O\left(\frac{N_x^2N_x}{p} + N_x^3 \log_2 D\right)$, with memory consumption $O\left(\frac{N_x^2N_x}{D} + \frac{N_x^2D}{p}\right)$. The first term ($\frac{N_x^2N_x}{p}$) 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 ($N_x^3 \log_2 D$) is
dependent on the number of levels needed to gather combining information for $D$ sub-problems, 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 sub-problem governed.
**Pseudo-Code**

1. Decompose the matrix \( K \) into \( D \) divisions

2. Each sub-problem from \( \{1, 2, ..., D\} \) is assigned to a single computer in \( \{1, 2, ..., p\} \):
   - Set governing flags on each computer: \( \text{gov}[i] = \begin{cases} \text{TRUE} & \text{if computer governs sub-problem } i \\ \text{FALSE} & \text{else} \end{cases} \)

3. For each governed sub-problem a computer will independently:
   - Determine ratio sequences
   - Determine diagonal blocks of inverse
   - Construct corner blocks of inverse

4. Generate list of sub-problem combinations:

   ```
   size = 1
   k = 0
   while (size < D) {
     i = 0
     while (i + size < D) {
       k = k + 1
       list(k).index = \([i + 1, \min(i + 2 \times size, D), i + size]\)
       i = i + 2 \times size
     }
     size = size * 2
   }
   ```

5. Each computer will adjust maps for any combining step involving a governed sub-problem:

   ```
   k = 1
   while (k < D) {
     [\text{start, stop, bridge point}] = list(k).index
     if (\text{gov[start]} \lor \text{gov[stop]} \lor \cdots \lor \text{gov[stop+1]}) == \text{TRUE}){
       if (\text{gov[stop]} \lor \text{gov[bridge point]} \lor \text{gov[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 according to (15) and (16)
     }
     k = k + 1
   }
   ```

6. For each governed sub-problem a computer will independently:
   - Apply matrix maps to transform sub-problem solutions into corresponding section of the global solution

7. Gather updated local solutions to form global solution
6 Results and Discussion

6.1 Preface

The algorithm in [13] calculates the diagonal blocks \( (D_i) \) of \( K^{-1} \) in the following manner.

\[
\begin{align*}
G_1 &= A_1^{-1}, \\
G_{i+1} &= (A_{i+1} - B_iG_iB_i)^{-1}, \quad i = 1, 2, \ldots, N_y - 1, \\
D_{N_y} &= G_{N_y}, \\
D_i &= G_i + G_iB_iD_{i+1}B_iG_i, \quad i = N_y - 1, N_y - 2, \ldots, 1.
\end{align*}
\]  

(17)

The time complexity of this algorithm (we will refer to as RGF, Recursive Green’s Function approach) was shown to be \( O(N_y^3N_y) \), with a memory requirement of \( O(N_y^2N_y) \). 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 upon 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).

6.2 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 [12], or special purpose hardware for simulation. Our divide-and-conquer algorithm offers a direct Green’s function approach that is applicable to these nano-scale device simulations. Through the general framework shown in [9] we can construct the retarded Green’s function matrix equation:

\[
KG' = (EI - H - \Sigma_1 - \Sigma_2)G' = I
\]

(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:

$$T = \text{Trace}[\Gamma_1 G \Gamma_2 G^t], \quad \Gamma_{1,2} = i[\Sigma_{1,2} - \Sigma_{1,2}^t].$$

(19)

This can be seen easily by recalling that the matrix map formulation shown in §4.3 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 wire of size $2\text{nm} \times 2\text{nm} \times 20.6\text{nm}$, considering 10 orbitals per atom, across 90 energy points. The transmission and density of states are shown in Fig. 9, and were generated using $D = p = 16$. Although this example requires less than 1GB 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 3GB of memory for a $2.5\text{nm} \times 2.5\text{nm} \times 102.6\text{nm}$ nanowire, and 6GB for a $3\text{nm} \times 3\text{nm} \times 102.6\text{nm}$ 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 $N_y = 756$. Our algorithm facilitates simulation for these devices without the use of special purpose hardware. The DOS and transmission for these geometries are shown in Fig. 10 and Fig. 11, again assuming $D = p = 16$. It is important to note that these simulations considered only 10 atomic orbitals, if spin was considered the memory requirements described above would increase by a factor.
Figure 10: Transmission and DOS for Si [100], 2.5nm × 2.5nm × 102.6nm.

Figure 11: Transmission and DOS for Si [100], 3nm × 3nm × 102.6nm.
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 nano-scale device simulations can be performed. An analysis of the computational performance for our algorithm is provided in the following sections.

6.3 Peak Performance

In order to compare our algorithm against the current state of the art, we consider the simulation of nanotransistors based upon a MIT well-tempered 25nm device-like structure [1]. Our algorithm (PDIV), 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, \((N_x, N_y) = (100, 4992)\) 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 1 shows the run-time of RGF, and PDIV implemented for \(p = 2, 4, \cdots, 64\). Here, the number of divisions is given by \(D = \frac{N_y}{p}\), the single computer case \((p = 1)\) uses RGF, and the remaining cases \((p > 1)\) utilizes 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 \(O\left(\frac{N^2 N_y}{p} + N^3 \log_2 D\right)\). As \(p \rightarrow D\), the second term in the computational cost begins to dominate, resulting in the gradual decay of speed-up shown in Table 1.

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 \(N_y\), i.e. \(O(N^2 \log_2 N)\). Table 2 shows the speed-up for \(p = 4, 8, \cdots, 64\) while keeping the ratio of grid points constant, i.e. \(D = p = \frac{N_y}{400}\). As can be seen in Table 2, for the case \(p = 64\), PDIV was able to achieve a speed-up of 20.90× as compared to the time estimate of RGF.

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 \((N_x, N_y) = (100, 512000)\) 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 \(D > p\) sub-problems subject to the memory constraints imposed by only using \(p\) computers. In the case of \(D > p\), multiple sub-problems should be assigned to the computers based upon spatial location in order to reduce unnecessary communication.

6.4 Scalability

While it seems clear from §6.3 that the divide-and-conquer algorithm offers benefits in both memory and computational complexity, 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.

Fig. 12 shows the time needed to find DOS for the case $N_x = 100$, while varying $N_y$ 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 $p = 32$ due to superior time as compared to that of $p = 16$. 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 speed-up 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. Fig. 14 and 15 show the same analyses for the case $N_y = 624$ and varying $N_x$.

### Table 1: Run time comparison $(N_x, N_y) = (100, 4992)$.

<table>
<thead>
<tr>
<th>$p$</th>
<th>(RGF)</th>
<th>1</th>
<th>2</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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>137.88</td>
<td>168.99</td>
<td>85.40</td>
<td>42.57</td>
<td>22.95</td>
<td>12.90</td>
<td>9.64</td>
<td></td>
</tr>
<tr>
<td>Speed-up</td>
<td>1</td>
<td>0.81</td>
<td>1.61</td>
<td>3.23</td>
<td>6.00</td>
<td>10.68</td>
<td>14.32</td>
<td></td>
</tr>
</tbody>
</table>

### Table 2: Peak performance; Ratio of run time (Extrapolated-RGF / PDIV)

<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>Speed-up</td>
<td>1.47</td>
<td>3.00</td>
<td>5.81</td>
<td>11.29</td>
<td>20.90</td>
</tr>
</tbody>
</table>
Figure 12: Time for PDIV for varying $N_y$ with $N_x = 100$ and $p = D = 2, 4, \cdots, 32$.

Figure 13: Speed-up for PDIV for varying $N_y$ with $N_x = 100$ and $p = D = 2, 4, \cdots, 32$. 
Figure 14: Time for PDIV for varying $N_x$ with $N_y = 624$ and $p = D = 2, 4, \cdots, 32$.

Figure 15: Speed-up for PDIV for varying $N_x$ with $N_y = 624$ and $p = D = 2, 4, \cdots, 32$. 
*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), Gerhard Klimeck (NCN Purdue University) for providing nanowire simulation resources.

**This material is based on work supported by the NASA, under Award NCC 2-1363, and the Semiconductor Research Corporation (SRC).

References


