Fast Inverse using Nested Dissection for NEGF

S. Li, S. Ahmed*, and E. Darve
Institute for Computational and Mathematical Engineering, Stanford University
496 Lomita Mall, Durand building, Room 265, Stanford, CA 94305-4040, U.S.A.
*Network for Computational Nanotechnology, Purdue University, W. Lafayette, IN 47907, USA.
e-mail: darve@stanford.edu

ABSTRACT

An accurate and efficient algorithm for computing non-equilibrium Green's functions has been developed and used in the simulations of a novel nanoscale MOSFET device structure. The method is based on the principle of nested dissection and demonstrates significant performance improvement from both speed and memory requirements points of view as compared to the state of the art recursive Green's function approach.

INTRODUCTION

In recent years, nanoscale MOS transistors as well as nanowires and molecular electronic devices have been actively studied [1]. It is now possible to manufacture transistors with channel lengths as small as 10 nm and below. At these scales, quantum effects are significant and have to be included in the simulation model. Despite the fact that transport issues for nano-transistors, nanowires and molecular electronic devices are very different from one another, they can be treated with the common formalism consisting of the nonequilibrium Green's function (NEGF) equations solved self-consistently with Poisson's equation. However, accurate and reliable modeling of the future nanoscale devices requires huge computational efforts, yet the current algorithms are prohibitively expensive.

In this work, a computationally efficient algorithm named Fast Inverse using Nested Dissection (FIND) has been developed and used to compute the required components of non-equilibrium Green's functions in the simulations of nanoscale devices. The numerical problem consists in computing the diagonal elements of the matrix $\boldsymbol{G} = [E\boldsymbol{I} - \boldsymbol{H} - \boldsymbol{\Sigma}]^{-1}$ (retarded Green's function) and $\boldsymbol{G}^< = \boldsymbol{G}\boldsymbol{\Sigma}^<\boldsymbol{G}^\dagger$ (less-than Green's function, $\dagger$ denotes the transpose conjugate of a matrix), where the energy level E, Hamiltonian matrix $\boldsymbol{H}$, and self energies $\boldsymbol{\Sigma}$ and $\boldsymbol{\Sigma}^<$ (see Svizhenko [2] for those notations) are in this work considered to be given. We now describe how to compute the diagonal of $\boldsymbol{G}$ and $\boldsymbol{G}^<$ efficiently. FIND can be derived for devices of arbitrary geometry and for arbitrary boundary conditions; however for simplicity we will focus in this paper on 2D rectangular devices (a typical geometry used for modeling MOSFETs [2]).

BRIEF DESCRIPTION OF THE ALGORITHM

Let's first consider $\boldsymbol{G}$. The basic idea of our algorithm is to perform many LU factorizations on the given matrix to compute the diagonal elements of its inverse. By performing the LU factorization in a certain order that minimizes fill-ins [5], we can preserve the sparsity of the given matrix and thus make the LU factorization very efficient. Once the LU factorization is complete, we can easily compute the last entry on the diagonal of the inverse: for an $n \times n$ matrix $\boldsymbol{G}^{-1} = \boldsymbol{L}\boldsymbol{U}$, we have $\boldsymbol{G}_{nn} = 1/\boldsymbol{U}_{nn}$. Although we can only compute $\boldsymbol{G}_{nn}$ in this way, we can choose any node and reorder the original matrix to make the node correspond to the (n, n) entry of the reordered matrix. In this way, all the diagonal elements of $\boldsymbol{G}$ can be computed.

If we have to perform a full LU factorization for each of the n reordered matrices, the algorithm will not be computationally efficient even though each LU factorization is very fast. However, if we reorder those matrices properly, many partial factorizations for different reordered matrices turn out to be identical. We can store the results of those partial factorizations in a binary tree and reuse them many times thereby reducing considerably the computational cost.

To label the binary tree and show how the algorithm works, we use the following notations: M is

the set of all the nodes in the mesh; C is the set of all the nodes in a cluster (a subset of M); $C_o = M \backslash C$ (outer nodes); C_b is the set of nodes (boundary nodes) in C connected to C_o; and $C_i = C \backslash C_b$ (inner nodes). We use a number to label each node in the binary tree, e.g., 10 stands for C_{10}, -10 stands for $C_{-10} = M \backslash C_{10}$. The following figures show the structure of the tree and how it is constructed:

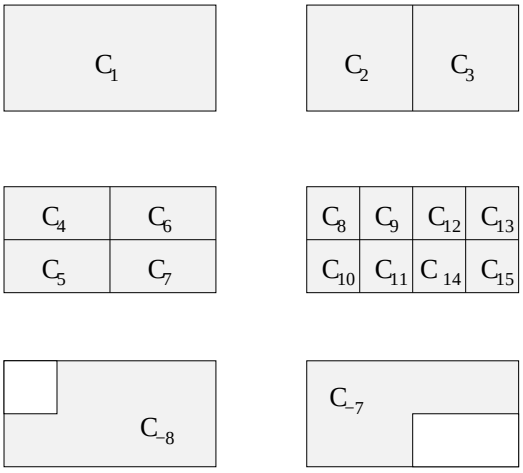

Fig. 1. The mesh and its partitions. $C_1 = M$.

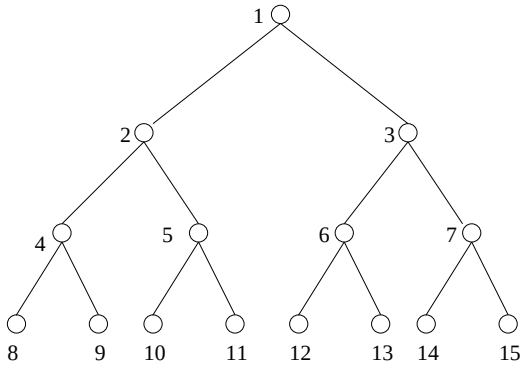

Fig. 2. The basic cluster tree of the mesh.

To compute $\boldsymbol{G}_{nn}$ of a leaf node, we need to eliminate all the other nodes in the mesh. We construct an augmented tree for each leaf node to show the steps of the elimination. For each leaf node, we go through its augmented tree and eliminate the inner nodes of each tree node from bottom up. Going through such an augmented tree is quite efficient as in a typical nested dissection algorithm [5].

The main merit of our algorithm is that the augmented trees shown above for each leaf node

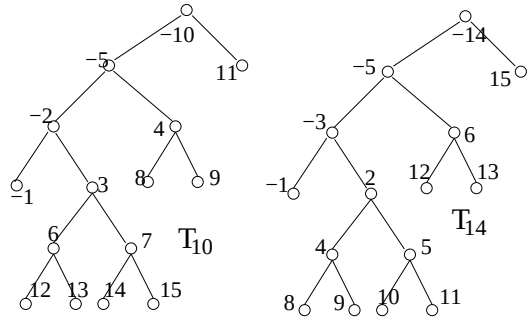

Fig. 3. Two augmented trees for clusters C_{10} and C_{14}.

considerably overlap with each other and the eliminations of the subtrees in an augmented tree are independent of each other (if they satisfy certain conditions). As a result, the cost of going through all the augmented trees is of the same order as going through one tree (detailed analysis below). The rigorous proof of the above properties is quite long and not given here.

We can also extend the above idea to computing $\boldsymbol{G}^<$. Define $\boldsymbol{R} \stackrel{def}{=} \boldsymbol{L}^{-1}\boldsymbol{\Sigma}^<\boldsymbol{L}^{-\dagger} = \boldsymbol{U}\boldsymbol{G}^<\boldsymbol{U}^\dagger$, then we have $\boldsymbol{G}^<_{nn} = \boldsymbol{R}_{nn}/|\boldsymbol{U}_{nn}|^2$. In general, $\boldsymbol{R}$ is a dense matrix which is very expensive to calculate. However, if $\boldsymbol{\Sigma}^<$ has the same pattern of nonzero entries as that in G^{-1}, the ordering used for computing G can preserve the sparsity of $\boldsymbol{\Sigma}^<$ as well. As a result, we were able to create an algorithm to compute $\boldsymbol{R}_{nn}$, whose cost is of the same order as computing $\boldsymbol{U}_{nn}$.

Computational and Memory Cost

When we perform LU factorizations, only the entries corresponding to the nodes in the boundary set will be affected. It is sufficient to store the partial results in matrices corresponding to the boundary nodes. Both the computational cost and the memory cost are mainly from dealing with such matrices.

For a squared mesh of size $N \times N$, the computational cost dominates at the top level of the binary tree. The cost of performing partial factorization at the top level is $O(N^3)$ since the size of the boundary nodes at the top level is $O(N)$. For rectangular meshes of size $N_x \times N_y$ with $N_x < N_y$, we keep cutting the mesh by half in the x direction until the clusters become squared. The total cost is thus proportional to $\Sigma_{i=1}^{L}(N_x^3 \times 2^i) + N_x^3$, where $L = \log_2(N_y/N_x)$ is the number of levels before the

clusters become square. So the total computational cost is $O(N_x^2 N_y)$. Now we can see that the total cost of performing the LU factorizations on all the n reordered matrices is of the same order as performing the LU factorization on one matrix as promised earlier. Computing $\boldsymbol{G}^<$ is more costly since it involves more computation in each step, but it is of the same order as computing $\boldsymbol{G}$ and has the same asymptotic behavior.

As a comparison, the state-of-the-art Recursive Green's Function (RGF) algorithm given by Svizhenko *et al.* [2] (see Lake [3] for an earlier version in 1D) is based on computing all the diagonal blocks of $\boldsymbol{G}$ and $\boldsymbol{G}^<$ using a forward and backward recurrence along the y axis of the device. The cost of RGF is $O(N_x^3 N_y)$ since the inverse of N_y matrices of size N_x needs to be computed. We can see by analysis that FIND is asymptotically much better than RGF: FIND takes $O(N_x^2 N_y)$ time while RGF takes $O(N_x^3 N_y)$ time.

Unlike the computing cost that dominates at the top level, the memory cost is about the same at each level of the tree. As explained earlier, we need to store the partial results of the LU factorization in the form of matrices. At each level, the memory cost is proportional to the total area of the mesh, which is $N_x\, N_y$. Thus the total memory cost is $O(N_x N_y \log_2(N_x))$. This is asymptotically better than RGF where the memory cost is $O(N_x^2 N_y)$.

RESULTS

We firstly made comparisons between FIND and RGF in a standalone environment. Fig. 4 shows a comparison of running time between the two algorithms. We can see that our algorithm uses much less time when the mesh size exceeds 50×100 and scales much better overall.

Fig. 5 shows the comparison of memory cost between the two algorithms. We can see that the memory cost of the two algorithms is about the same but the memory cost of our algorithm increases slower than the other algorithm so it is asymptotically better.

To assess the performance and applicability of FIND in realistic nanoscale MOSFET devices, both FIND and RGF have been applied to the simulations of a realistic two-dimensional non-conventional MOSFET device structure within a general community software framework called nanoFET. Fig. 6 below shows the physical device we simulated.

Within the framework of nanoFET, the density of states (DOS) and electron density were computed for a nanoscale dual-gate MOSFET device structure with the following specifications: the gate length is 10 nm, the channel length is 12 nm (including 1 nm source/drain extensions on each side), the source/drain length is 14 nm each, the silicon film thickness is 5 nm, the top/bottom gate oxide thickness is 1.5 nm each, the channel is undoped, the doping of the source/drain regions equals $1 \times 10^{20} cm^{-3}$, and the gate is assumed to be a metal gate with an affinity adjusted to 4.4227 eV.

Fig. 7 and fig. 8 show that the RGF and FIND algorithms produce identical density of states and electron density. Fig. 9 shows a comparison of simulation time between the FIND and the RGF algorithms. In the experiment, N_x was increased while keeping N_y constant at 149. FIND shows a considerable speed-up for N_x beyond around 120. We also estimated the rate of increase with respect to N_x based on the slope of $\log(time) - \log(N_x)$ plot. We have the slope for RGB equal to 2.98 and the slope for FIND equal to 1.93, which are consistent the theoretical estimation of the computation costs: $O(N_x^3 N_y)$ and $O(N_x^2 N_y)$, respectively.

CONCLUSION

We have developed an efficient method of computing the diagonal entries of the retarded Green's function (density of states) and the diagonal of the less-than Green's function (density of charges). The algorithm is exact and uses Gaussian eliminations. A simple extension allows computing off diagonal entries for current density calculations. For a 2D rectangular nano-transistor discretized with a mesh of size N_x by N_y, $N_x < N_y$, the cost is $O(N_x^2 N_y)$, which improves over the result of RGF [2], whose algorithm scales as $O(N_x^3 N_y)$. This allows simulating larger and more complex devices using a finer mesh and with a small computational cost. Our algorithm is also applicable to nanowires, nanotubes, molecules and 3D transistors, with arbitrary shape.

ACKNOWLEDGMENT

Thank Dr. M. P. Anantram and Dr. Gerhard Klimeck for their advice on the comparison with

the RGF algorithm and realistic device simulation. Thanks for the NSF funding support.

REFERENCES

[1] Y. Xue, S. Datta, and M. A. Ratner, "First-principles based matrix Green's function approach to molecular electronic devices: general formalism," *Chemical Physics* **281**(2/3), pp. 151-70, 2002

[2] A. Svizhenko, M. P. Anantram, T.R. Govindan, B. Biegel and R. Venugopal, "Two-dimensional quantum mechanical modeling of nanotransistors," *Journal of Applied Physics* **91**(4), pp. 2343-2354, 2002

[3] R. Lake, G. Klimeck, R.C. Bowen, and D. Jovanovic, "Single and multiband modeling of quantum electron transport through layered semiconductor devices," *Journal of Applied Physics* **81**(12), pp. 7845-69, 1997

[4] S. Datta, "Nanoscale device modeling: the Green's function method," *Superlattices and microstructures* **28**(4), pp. 253-278, 2000

[5] A. George, "Nested dissection of a regular finite element mesh," *SIAM Journal on Numerical Analysis* **10**(**2**), pp. 345-63, 1973

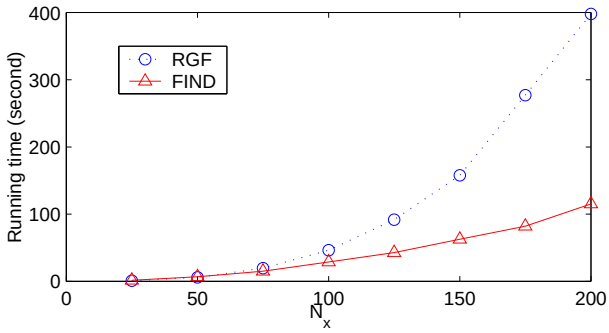

Fig. 4. Comparison of running time. $N_y = 100$

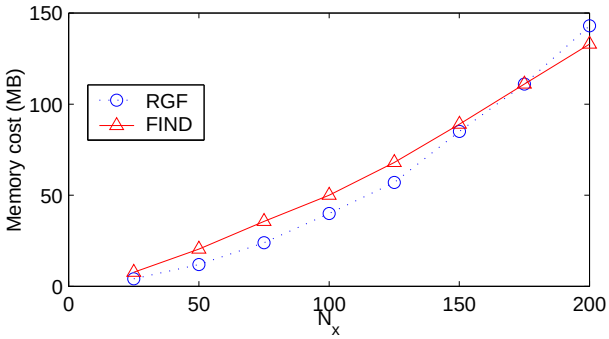

Fig. 5. Comparison of memory cost. $N_y = 100$.

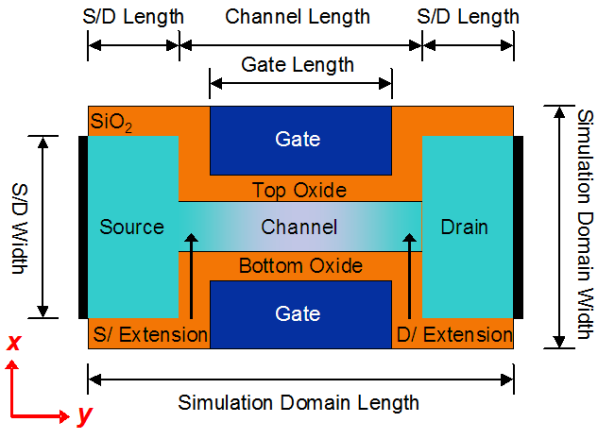

Fig. 6. Illustration of the device.

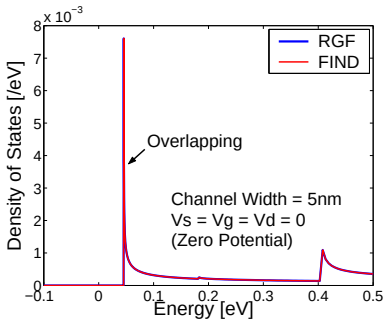

Fig. 7. Comparison of the density of states.

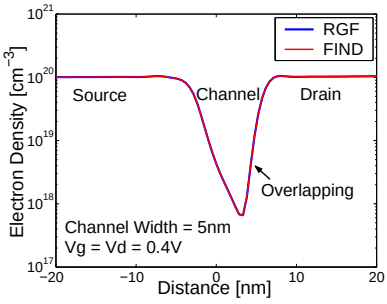

Fig. 8. Comparison of the electron density.

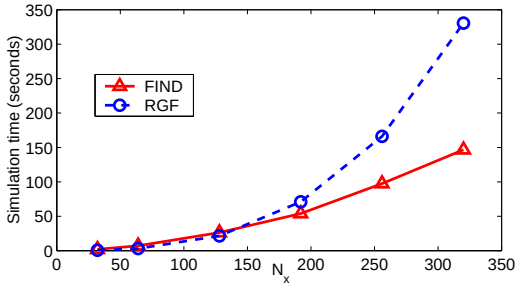

Fig. 9. Comparison of simulation time per energy level.