FAST COMPUTATIONAL METHODS FOR COMPUTING QUANTUM TRANSPORT IN NANOWIRES AND NANOTUBES

Eric Darve*, and Song Li
Stanford University and Army High Performance Computing and Research Center
Stanford CA 94305

Dan Erik Petersen, and Kurt Stokbro
University of Copenhagen
Copenhagen, Denmark

ABSTRACT

Nanowires and nanotubes are promising building blocks for designing nanoscale devices for sensing warfare agents. Computer models are key to designing and improving such sensors. Due to their nanoscale size, quantum models are needed to model the transport of electrons in this type of device. Few methods exist to accurately calculate quantum transport for systems comprising hundreds of thousands of atoms. The approach we are taking is based on the non equilibrium Green's function approach[1, 2, 3] (NEGF) and is an exact order N method to calculate the charge density and the IV characteristic of a device. We are presenting a novel numerical technique which allows computing in parallel the Green's function. The computational cost scales linearly with the number atoms and the parallel efficiency on benchmarks problems is nearly optimal. Based on our benchmarks results this method will enable modeling devices of unprecedented size.

1 INTRODUCTION

In NEGF, we need to consistently solve for the eigenvectors of the single-particle Schrödinger Hamiltonian $\boldsymbol{H}$:

$$(\boldsymbol{H} - E\boldsymbol{S})c = 0$$

(where $\boldsymbol{S}$ is the so-called overlap matrix) and a Poisson equation for the electrostatic potential. Computationally, the most expensive step is obtaining the retarded and less-than Green's functions defined by:

$$\boldsymbol{G}^{\mathrm{r}} = (\boldsymbol{H} - \epsilon\boldsymbol{S})^{-1}, \text{ and } \boldsymbol{G}^{<} = \boldsymbol{G}^{\mathrm{r}}\,\boldsymbol{\Gamma}\,[\boldsymbol{G}^{\mathrm{r}}]^{*}, \quad (1)$$

where $\epsilon = E + i\eta$, with η infinitesimal, and $\boldsymbol{\Gamma}$ is the complex broadening matrix and can be assumed to have the same non-zero sparsity pattern as $\boldsymbol{H}$. The infinitesimal η is necessary in order to specify the boundary conditions. Because of the requirement that the Schrödinger equation is solved consistently with the Poisson equation (which requires converging the charge density) Equation (1) must be solved repeatedly at all relevant energy levels until convergence. A key aspect of this calculation is that only the diagonal entries of $\boldsymbol{G}^{\mathrm{r}}$ and $\boldsymbol{G}^{<}$ are actually needed to compute the charge density. Since the size of the matrices scales linearly with the number of atoms, this problem is computationally very expensive and even on large parallel computers it is difficult to solve problems with more than a few hundred atoms, which is far below the size of new emerging sensing devices.

Table 1: notations used in this paper.

$\mathbf{A}$	matrix
n	number of blocks in $\mathbf{A}$
d	block size
$\mathbf{A}^T$	transpose matrix
$\mathbf{A}^\dagger$	transpose conjugate matrix
$\mathbf{A}^{-\dagger}$	transpose conjugate of the inverse matrix
$\mathbf{L}$, $\mathbf{D}$, $\mathbf{U}$	LDU factorization of $\mathbf{A}$
$\mathbf{a}_{ij}$	block entry (i,j) of matrix $\mathbf{A}$
$\mathbf{A}(i1:i2,j)$	block column vector containing rows $i1$ through $i2$ in column j
$\mathbf{A}(i,j1:j2)$	block row vector containing columns $j1$ through $j2$ in row i
$\mathbf{A}(i1:i2,j1:j2)$	sub-matrix containing rows $i1$ through $i2$ and columns $j1$ through $j2$
$\mathbf{I}$	Identity matrix
$\mathbf{0}$	All-zero matrix

In order to expand the current simulation capabilities, we have developed a novel approach to solve Equation (1) on parallel computers. In principle, if $\boldsymbol{H}$ has size $N \times N$, the cost of computing $\boldsymbol{G}^{\mathrm{r}}$ and $\boldsymbol{G}^{<}$ is

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
DEC 2008	**N/A**	**-**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Fast Computational Methods For Computing Quantum Transport In Nanowires And Nanotubes	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Stanford University and Army High Performance Computing and Research Center Stanford CA 94305**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES
See also ADM002187. Proceedings of the Army Science Conference (26th) Held in Orlando, Florida on 1-4 December 2008, The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**UU**	**7**	

$O(N^3)$. However, we only need the diagonal entries of $\boldsymbol{G}^{\mathrm{r}}$ and $\boldsymbol{G}^{<}$. We demonstrated that these entries can be computed in $O(N)$ steps only[4]. This algorithm will be described first. Then we will present our parallelization strategy and numerical results.

2 FAST RECURRENCE TO COMPUTE THE GREEN'S FUNCTION

Consider a general matrix $\mathbf{A}$ written in block form:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} \tag{2}$$

The $\mathbf{L}$ and $\mathbf{U}$ factors in a Gaussian elimination are given by: $\mathbf{L}^{\mathrm{S}} = \mathbf{A}_{21}\mathbf{A}_{11}^{-1}$ and $\mathbf{U}^{\mathrm{S}} = \mathbf{A}_{11}^{-1}\mathbf{A}_{12}$. The Schur complement is $\mathbf{S} = \mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12}$. The following equation holds for the inverse matrix $\mathbf{G} = \mathbf{A}^{-1}$:

$$\mathbf{G} = \begin{bmatrix} \mathbf{A}_{11}^{-1} + \mathbf{U}^{\mathrm{S}}\,\mathbf{S}^{-1}\,\mathbf{L}^{\mathrm{S}} & -\mathbf{U}^{\mathrm{S}}\,\mathbf{S}^{-1} \\ -\mathbf{S}^{-1}\,\mathbf{L}^{\mathrm{S}} & \mathbf{S}^{-1} \end{bmatrix} \tag{3}$$

This equation can be verified by direct multiplication by $\mathbf{A}$. It allows computing the inverse matrix using backward recurrence formulas. These formulas can be obtained by considering step i of the LDU factorization of $\mathbf{A}$, which has the following form:

$$\begin{aligned} \mathbf{A} = &\begin{bmatrix} \mathbf{L}(1:i-1,1:i-1) & \mathbf{0} \\ \mathbf{L}(i:n,1:i-1) & \mathbf{I} \end{bmatrix} \\ &\times \begin{bmatrix} \mathbf{D}(1:i-1,1:i-1) & \mathbf{0} \\ \mathbf{0} & \mathbf{S}^i \end{bmatrix} \\ &\times \begin{bmatrix} \mathbf{U}(1:i-1,1:i-1) & \mathbf{U}(1:i-1,i:n) \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \end{aligned} \tag{4}$$

From Equation (4), the first step of the LDU factorization of $\mathbf{S}^i$ is the same as the $i+1$th step in the factorization of $\mathbf{A}$:

$$\begin{aligned} \mathbf{S}^i = &\begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{L}(i+1:n,i) & \mathbf{I} \end{bmatrix} \\ &\begin{bmatrix} \mathbf{d}_{ii} & \mathbf{0} \\ \mathbf{0} & \mathbf{S}^{i+1} \end{bmatrix} \begin{bmatrix} 1 & \mathbf{U}(i,i+1:n) \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \end{aligned} \tag{5}$$

Combining Equation (5) and (3) applied to $\mathbf{A} = \mathbf{S}^i$, we arrive at:

$$[\mathbf{S}^i]^{-1} = \begin{bmatrix} \mathbf{d}_{ii}^{-1} + \mathbf{U}(i,i+1:n)\,[\mathbf{S}^{i+1}]^{-1}\,\mathbf{L}(i+1:n,i) & -\mathbf{U}(i,i+1:n)\,[\mathbf{S}^{i+1}]^{-1} \\ -[\mathbf{S}^{i+1}]^{-1}\,\mathbf{L}(i+1:n,i) & [\mathbf{S}^{i+1}]^{-1} \end{bmatrix}$$

From Equation (3), we have:

$$[\mathbf{S}^i]^{-1} = \mathbf{G}(i:n,i:n)$$

and

$$[\mathbf{S}^{i+1}]^{-1} = \mathbf{G}(i+1:n,i+1:n)$$

We have therefore proved the following backward recurrence relations:

$$\begin{aligned} &\mathbf{G}(i+1:n,i) = \\ &\quad -\mathbf{G}(i+1:n,i+1:n)\,\mathbf{L}(i+1:n,i) \\ &\mathbf{G}(i,i+1:n) = \\ &\quad -\mathbf{U}(i,i+1:n)\,\mathbf{G}(i+1:n,i+1:n) \\ &\mathbf{g}_{ii} = \\ &\quad \mathbf{d}_{ii}^{-1} + \mathbf{U}(i,i+1:n)\,\mathbf{G}(i+1:n,i+1:n) \\ &\quad \mathbf{L}(i+1:n,i) \end{aligned} \tag{6}$$

starting from $\mathbf{g}_{nn} = \mathbf{d}_{nn}^{-1}$. A recurrence for $i = n-1$ down to $i = 1$ can therefore be used to calculate $\mathbf{G}$. See Fig. 1.

By itself, these recurrence formulas do not lead to any reduction in the computational cost. However the following result shows that even though the matrix $\mathbf{G}$ is full, we do not need to calculate all the entries[5, 6, 7]. We denote $\mathbf{L}^{\mathrm{sym}}$ and $\mathbf{U}^{\mathrm{sym}}$ the lower and upper triangular factors in the symbolic factorization of $\mathbf{A}$. The non-zero structure of $(\mathbf{L}^{\mathrm{sym}})^T$ and $(\mathbf{U}^{\mathrm{sym}})^T$ is denoted by Q: this is the set of all pairs (i,j) such that $\mathbf{l}_{ji}^{\mathrm{sym}} \neq 0$ or $\mathbf{u}_{ji}^{\mathrm{sym}} \neq 0$. Then:

$$\mathbf{g}_{ij} = \begin{cases} -\displaystyle\sum_{l>j,\,(i,l)\in Q} \mathbf{g}_{il}\,\mathbf{l}_{lj} & \text{if } i > j \\ -\displaystyle\sum_{k>i,\,(k,j)\in Q} \mathbf{u}_{ik}\,\mathbf{g}_{kj} & \text{if } i < j \\ \mathbf{d}_{ii}^{-1} + \displaystyle\sum_{k>i,\,l>i,\,(k,l)\in Q} \mathbf{u}_{ik}\,\mathbf{g}_{kl}\,\mathbf{l}_{li} & \text{if } i = j \end{cases} \tag{7}$$

where $(i,j) \in Q$.

Proof. Assume that $(i,j) \in Q$, $i > j$, then $\mathbf{u}_{ji}^{\mathrm{sym}} \neq 0$. Assume also that $\mathbf{l}_{lj}^{\mathrm{sym}} \neq 0$, $l > j$, then by properties of the Gaussian elimination $(i,l) \in Q$. This proves the first case. The second case can be proved similarly.

For the third case, assume that $\mathbf{u}_{ik}^{\mathrm{sym}} \neq 0$ and $\mathbf{l}_{li}^{\mathrm{sym}} \neq 0$, then by properties of the Gaussian elimination $(k,l) \in Q$. □

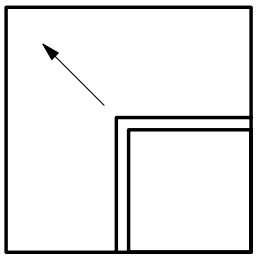

Fig. 1: Schematics of how the recurrence formulas are used to calculate $\mathbf{G}$.

This implies that we do not need to calculate all the entries in $\mathbf{G}$ but rather only the entries in $\mathbf{G}$ corresponding to indices in Q. This represents a significant reduction in computational cost. Using this algorithm, the cost of computing entries in $\mathbf{G}$ in set Q is the same (up to a constant factor) as the LDU factorization.

3 Parallel Algorithm

For a nanowire or nanotube type structure, the matrix $\boldsymbol{H} - \epsilon \boldsymbol{S}$ can be written as a block tri-diagonal matrix. Even though our algorithm can be extended to general sparse matrices, in with work, we focus on computing $\boldsymbol{G}^{\mathrm{r}}$ for tri-diagonal block matrices to simplify the discussion. The matrix $\mathbf{A}$ is assumed to be an $n \times n$ block matrix, and in block tridiagonal form as shown by

$$\mathbf{A} = \begin{pmatrix} \mathbf{a}_{11} & \mathbf{a}_{12} & & & \\ \mathbf{a}_{21} & \mathbf{a}_{22} & \mathbf{a}_{23} & & \\ & \mathbf{a}_{32} & \mathbf{a}_{33} & \mathbf{a}_{34} & \\ & & \ddots & \ddots & \ddots \end{pmatrix}. \tag{8}$$

where each block element $\mathbf{a}_{ij}$ is a dense complex matrix. In order to develop a parallel algorithm, we assume that we have at our disposal a total of $\mathcal{P}$ processes, each labeled p_0, p_1, ..., $p_{\mathcal{P}-1}$. The block tridiagonal structure $\mathbf{A}$ is then distributed among these processes in a *row-striped* manner, as illustrated in Fig. 2.

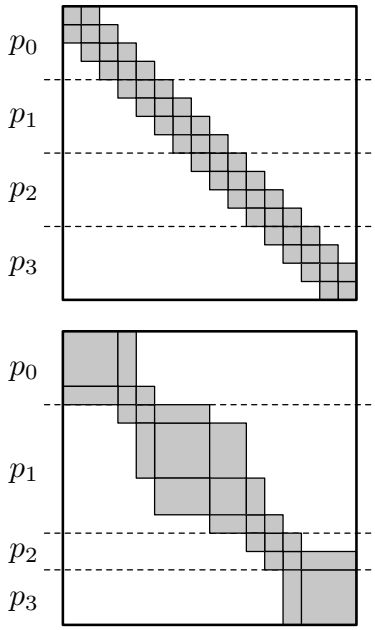

Fig. 2: Two different block tridiagonal matrices distributed among 4 different processes, labeled p_0, p_1, p_2 and p_3.

Thus each process is assigned ownership of certain contiguous rows of $\mathbf{A}$. This ownership arrangement, however, also extends to the calculated blocks of the inverse $\mathbf{G}$, as well as any LU factors determined during calculation in $\mathbf{L}$ and $\mathbf{U}$. The manner of distribution is an issue of load balancing, and will be addressed later in the paper.

Furthermore, for illustrative purposes, we present the block matrix structures as having identical block sizes throughout. The algorithm presented has no condition for the uniformity of block sizes in $\mathbf{A}$, and can be applied to a block tridiagonal matrix as presented on the right in Fig. 2. Block sizes throughout $\mathbf{A}$, however, does have consequences for adequate load balancing.

For the purposes of electronic structure applications, we only require the portion of the inverse $\mathbf{G}$ with the same block tridiagonal structure structure of $\mathbf{A}$. We express this portion as $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$.

The parallel algorithm is a *hybrid* technique in the sense that it combines the techniques of cyclic reduction and Schur block decomposition, but where we now consider individual elements to be dense matrix blocks. The steps taken by the hybrid algorithm to produce $\mathrm{Trid}_{\mathbf{A}}\{\mathbf{G}\}$ is outlined in Fig. 3.

The algorithm begins with our block tridiagonal matrix $\mathbf{A}$ partitioned across a number of processes, as

indicated by the dashed horizontal lines. Each process then performs what is equivalent to a Schur decomposition on the rows/blocks that it owns, leaving us with a *reduced* system that is equivalent to a smaller block tridiagonal matrix $\mathbf{A}^{\text{Schur}}$. This phase, which we name the Schur reduction phase, is entirely devoid of interprocess communication.

It is on this smaller block tridiagonal structure for which we perform block cyclic reduction (BCR), leaving us with a single block of the inverse, $\mathbf{g}_{kk}$. This block cyclic reduction phase involves interprocess communication.

From $\mathbf{g}_{kk}$ we then produce the portion of the inverse corresponding to $\mathbf{G}^{\text{BCR}} = \text{Trid}_{\mathbf{A}^{\text{Schur}}}\{\mathbf{G}\}$ in what we call the block cyclic production phase. Finally, using $\mathbf{G}^{\text{BCR}}$, we can then determine the full tridiagonal structure of $\mathbf{G}$ that we desire without any further need for interprocess communication through a so-called Schur production phase. The block cyclic production phase and Schur production phase are a parallel implementation of the backward recurrences in Equation (6).

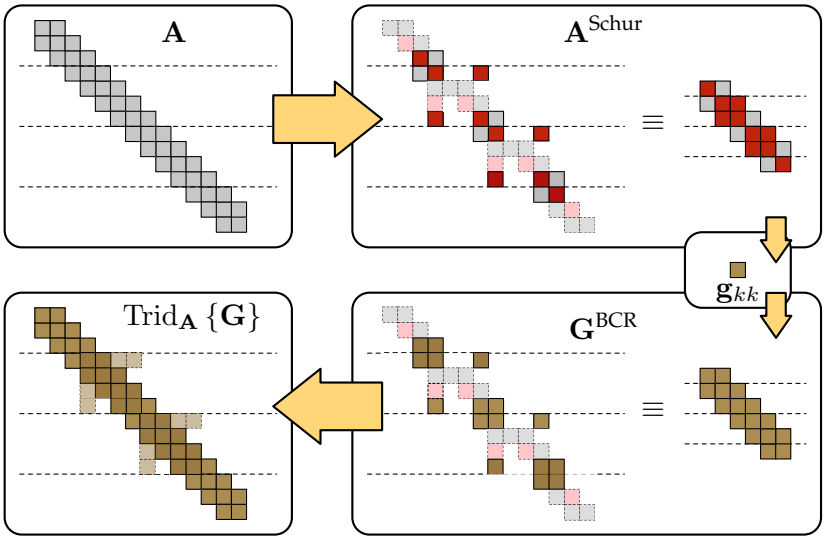

Fig. 3: This figure illustrates the distinct phases of operations performed by the hybrid method in determining $\text{Trid}_{\mathbf{A}}\{\mathbf{G}\}$, the block tridiagonal portion of $\mathbf{G}$ with respect to the structure of $\mathbf{A}$. The block matrices in this example are partitioned across 4 processes, as indicated by the horizontal dashed lines.

Assume now that we have P processors at our disposal. The N blocks can be split into P clusters of contiguous blocks, each assigned to a processor. The parallel algorithm is decomposed into 3 steps. Step 1 is an embarrassingly parallel step (it requires no communication) and consists in eliminating all the blocks inside the clusters except the block at the right end of the cluster. Step 2 requires $2\log_2 P$ passes and involves communication. Step 3 produces all the diagonal entries of $\boldsymbol{G}^{\text{r}}$ and $\boldsymbol{G}^{<}$ using a sequence of operations similar to Step 1, again requiring no communication.

4 NUMERICAL RESULTS

By performing an operation count under the assumption of equal block sizes throughout $\mathbf{A}$, and that an LU factorization costs on the order of $\frac{2}{3}d^3$ operations and a matrix-matrix multiplication costs $2d^3$ operations for a matrix of dimension d, we can estimate the ratio α of number of rows assigned to the first and last processes vs. central processes (see Fig. 2). An analysis of our implemented algorithm predicts $\alpha = 2.636$ to be an optimal choice. For the sake of completeness, we investigate the case of $\alpha = 1$, where each process is assigned the same number of rows, while the values $\alpha = 2$ and $\alpha = 3$ are chosen to bracket the optimal choice.

Execution time was measured on implemented ver-

sions of a pure block cyclic reduction algorithm and of our hybrid algorithm. The wall time measurements for running these algorithms on a block tridiagonal matrix $\mathbf{A}$ with $n = 512$ block rows with blocks of dimension $m = 256$ are given in Fig. 4[a-d] for the four different load balancing values of $\alpha = \{1, 2, 2.636, 3\}$. A total number of processors used for execution was $\mathcal{P} = \{1, 2, 4, 8, 16, 32, 64\}$ in all cases.

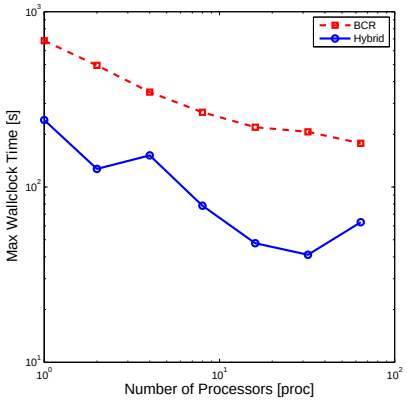

Fig. 4a: $\alpha = 1$.

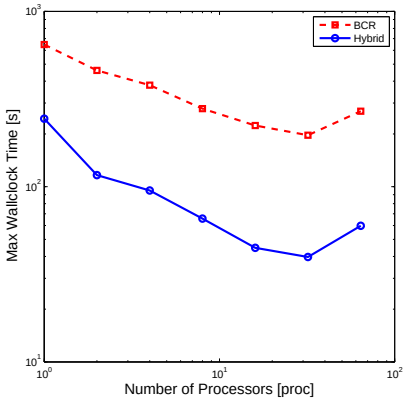

Fig. 4b: $\alpha = 2$.

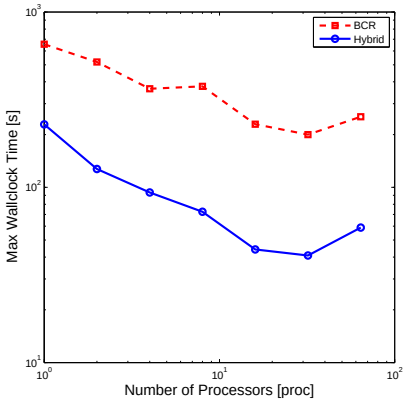

Fig. 4c: $\alpha = 2.636$.

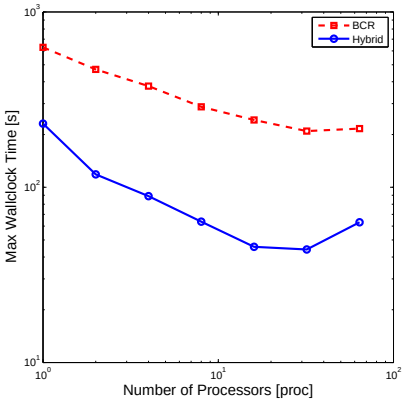

Fig. 4d: $\alpha = 3$.

This figure illustrates the wall time curves for the application of our hybrid algorithm and pure BCR under differing values of α as a basic load balancing parameter. The curves are produced by applying the different algorithms on a block tridiagonal matrix $\mathbf{A}$ with $n = 512$ diagonal blocks, each of dimension $m = 256$.

The speedup results for the corresponding wall time measurements are given in Fig. 5[a-d] for the four different load balancing values of α. For a uniform distribution where $\alpha = 1$, a serious performance hit is experienced for $\mathcal{P} = 4$. This is due to a poor load balance, as the first and last processes p_0 and p_3 terminate their Schur production/reduction phases much sooner than the central processes p_1 and p_2. It is then observed that choosing a basic load balancing parameter of $\alpha = 2$, 2.636 or 3 eliminates this dip in speedup.

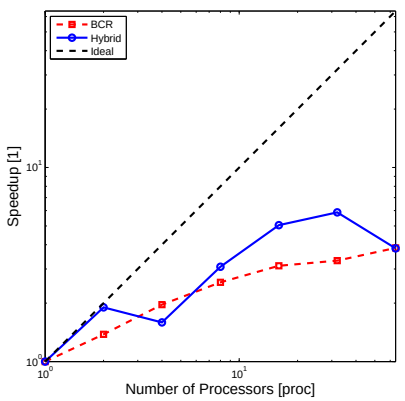

Fig. 5a: $\alpha = 1$.

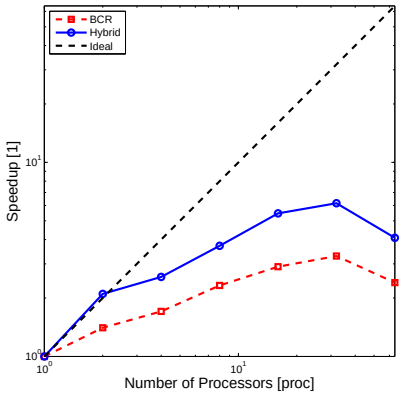

Fig. 5b: $\alpha = 2$.

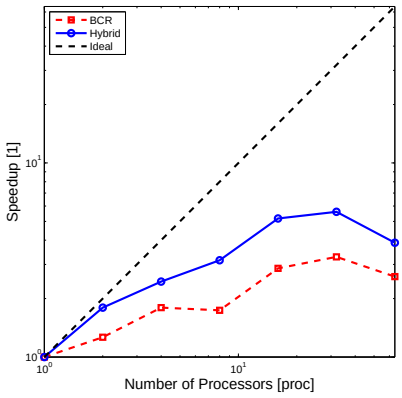

Fig. 5c: $\alpha = 2.636$.

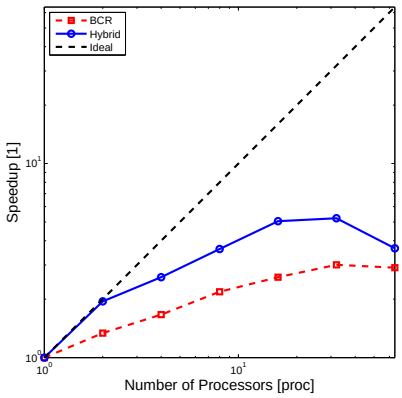

Fig. 5d: $\alpha = 3$.

This figure illustrates the speedup curves for the application of our hybrid algorithm and pure BCR under differing values of α as a basic load balancing parameter. The curves are produced by applying the different algorithms on a block tridiagonal matrix $\mathbf{A}$ with $n = 512$ diagonal blocks, each of dimension $m = 256$.

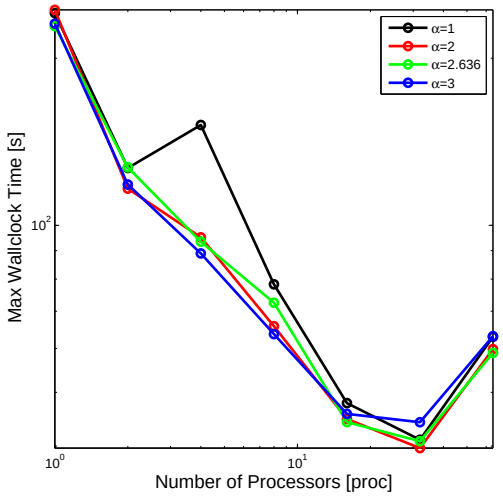

Fig. 6: The total execution time for our hybrid algorithm is plotted against the number of processes $\mathcal{P}$. The curves were produced for an example block tridiagonal matrix $\mathbf{A}$ with $n = 512$ diagonal blocks, each of dimension $m = 256$.

It can also be seen that as the number of processors $\mathcal{P}$ increases for a fixed n, a greater portion of execution time is attributed to the BCR phase of our hybrid algorithm. Ultimately higher communications and computational costs of the BCR over the embarrassingly parallel Schur phase of the algorithm dominate, and the speedup curves level off and drop, regardless of α.

In an effort to determine which load balancing parameter α is optimal, we compare the wall time execution measurements for our hybrid algorithm in Fig. 6. From this figure, we can conclude that a uniform distribution with $\alpha = 1$ leads to generally poorer execution times. Other values lead to improved execution times over $\alpha = 1$, particularly in the range $\mathcal{P} = 4 \ldots 8$, which is a common core count in high-end desktop computers. This suggests that a certain amount of flexibility can be afforded in deviating from the theoretically ideal load balancing of $\alpha = 2.636$.

5 CONCLUSION

We have presented a new algorithm and parallelization strategy to calculate the diagonal entries of Green's functions. This has application in quantum transport in nano-devices, for example nano-sensors using nanowires and carbon nanotubes. An optimization strategy has been proposed to improve the load balancing and the speed-up. The speed-up improves as

the length of the device increases (parameter n). Further improvements can be obtained by employing a different strategy to parallelize the BCR part of the algorithm which allows reducing the amount of communication by reducing the number of passes in the algorithm. This will be presented in a future paper.

References

[1] S. Datta, Electronic Transport in Mesoscopic Systems, Cambridge University Press, Cambridge, UK, 1997.

[2] H. Haug, A.-P. Jauho, Quantum Kinetics in Transport and Optics of Semiconductors, Berlin Springer Verlag, 1996.

[3] M. Brandbyge, J.-L. Mozos, P. Ordejón, J. Taylor, K. Stokbro, Density-functional method for nonequilibrium electron transport, Phys. Rev. B 65 (16) (2002) 165401.

[4] S. Li, S. Ahmed, G. Klimeck, E. Darve, Computing entries of the inverse of a sparse matrix using the find algorithm, J. Comput. Phys. In press, available online.

[5] K. Takahashi, J. Fagan, M.-S. Chin, Formation of a sparse bus impedance matrix and its application to short circuit study, in: 8th PICA Conf. Proc., Minneapolis, Minn., 1973, pp. 63–69.

[6] A. M. Erisman, W. F. Tinney, On computing certain elements of the inverse of a sparse matrix, Numerical Mathematics 18 (3) (1975) 177–79.

[7] H. Niessner, K. Reichert, On computing the inverse of a sparse matrix, Int. J. Num. Meth. Eng. 19 (1983) 1513–1526.