Computer Physics Communications 207 (2016) 105–113

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

A partitioned shift-without-invert algorithm to improve parallel eigensolution efficiency in real-space electronic transport

Baruch Feldman [a,*], Yunkai Zhou [b]

[a] Department of Materials and Interfaces, Weizmann Institute of Science, Rehovoth 76100, Israel
[b] Department of Mathematics, Southern Methodist University, Dallas, TX, 75093, USA

ARTICLE INFO

Article history:
Received 19 January 2016
Received in revised form
18 April 2016
Accepted 22 May 2016
Available online 31 May 2016

Keywords:
Eigensolution
Electronic structure
Electronic transport
High-performance computing
Parallel computing
Shift–invert

ABSTRACT

We present an eigenspectrum partitioning scheme without inversion for the recently described real-space electronic transport code, TRANSEC. The primary advantage of TRANSEC is its highly parallel algorithm, which enables studying conductance in large systems. The present scheme adds a new source of parallelization, significantly enhancing TRANSEC's parallel scalability, especially for systems with many electrons. In principle, partitioning could enable super-linear parallel speedup, as we demonstrate in calculations within TRANSEC. In practical cases, we report better than five-fold improvement in CPU time and similar improvements in wall time, compared to previously-published large calculations. Importantly, the suggested scheme is relatively simple to implement. It can be useful for general large Hermitian or weakly non-Hermitian eigenvalue problems, whenever relatively accurate inversion via direct or iterative linear solvers is impractical.

1. Introduction

Recently a real-space approach has been developed for Green's function-based ab-initio electronic conductance calculations, called TRANSEC [1]. TRANSEC inherits a number of intrinsic advantages associated with real-space electronic-structure calculations, including favorable parallelizability and no requirement of an explicit basis-set [1–3].

Within this approach the bottleneck in computing the electronic transmission function $T(E)$ at energy E is the partial diagonalization of a complex-symmetric matrix according to the equation

$$(H_{KS} - i\Gamma)\, U_k = \epsilon_k U_k, \quad \epsilon_k \in \mathbb{C}. \tag{1}$$

As described in detail previously [1], we perform this step as an abbreviated, cheaper intermediate to the inversion

$$G(E) \equiv \{E\mathbb{1} - (H_{KS} - i\Gamma)\}^{-1},$$

which is needed to compute transmission:

$$T(E) = \mathrm{Tr}\left\{G(E)\,\Gamma_R G^{\dagger}(E)\,\Gamma_L\right\}.$$

In Eq. (1), H_{KS} is the Hermitian Kohn–Sham (KS) Hamiltonian obtained from the PARSEC Density Functional Theory (DFT) code [2,3] after self-consistency is reached; the $i\Gamma \equiv i(\Gamma_L + \Gamma_R)$ is a sum of imaginary, diagonal absorbing potentials with Gaussian form at the two ends of the simulation cell, where $i^2 = -1$; and the ϵ_k and U_k are a pair of eigenvalue and eigenvector, respectively, of $H_{KS} - i\Gamma$ [1]. To calculate conductance in an implicit real-space basis, we use a simulation cell having a finite volume V, so the dimension of the matrix H_{KS} is given by $N \approx V/h^3$. Here h is the grid spacing of the real-space lattice and N is typically 10^4 to 10^6 or greater. To correct for the finite volume V, the imaginary absorbing potentials, $i\Gamma$, are tuned to absorb outgoing electrons and prevent reflections at the boundaries of the simulation cell [1]. Since H_{KS} is real-symmetric, the presence of $i\Gamma$ results in a complex-symmetric eigenproblem. This partial diagonalization is the most computation-intensive part of TRANSEC, and can take many hundreds or thousands of core-hours of computation on standard supercomputers [1].

TRANSEC is parallelized mainly by partitioning the real-space grid over computing cores during matrix–vector application.[1] This monolithic source of parallelization results in less-than-optimal

* Corresponding author.
E-mail addresses: baruch.feldman@weizmann.ac.il (B. Feldman), yzhou@smu.edu (Y. Zhou).

http://dx.doi.org/10.1016/j.cpc.2016.05.015
0010-4655/

[1] Note that symmetry and k-point sampling are not implemented because of the non-periodic geometry of conductance calculations, so parallelization over these dimensions is unavailable.

parallel scaling when, for instance, the number N_e of electrons under simulation increases in tandem with, or even faster than, the supercell volume V [1].

In this paper, we develop a further dimension of parallelization by partitioning over the eigenspectrum of Eq. (1). This scheme can significantly improve the iterative solution of the eigenproblem (1), leading to markedly better scalability in TRANSEC. As we will demonstrate in Section 3, super-linear parallel speedup, or net savings in computing time, are possible. The present scheme is conceptually and operationally simpler than shift–invert [4–7] and similar algorithms, yet still effective. It can accelerate the solution of large eigenvalue problems, both Hermitian and non-Hermitian, when matrix inversion via matrix decompositions such as sparse LU or via iterative linear solvers is impractical. We emphasize that (1) is weakly non-Hermitian,[2] as the eigenvectors of (1) are bi-orthogonal with respect to a standard inner product [1,8]; in contrast, for Hermitian eigenvalue problems, the eigenvectors are orthogonal. While iterative methods have been studied [9,10] for solution of linear equations with complex-symmetric matrices, few have been proposed for the corresponding eigenvalue problems, such as (1). Robust production code for the solution of sparse large-scale complex-symmetric eigenvalue problems such as Eq. (1), especially when many eigenpairs are needed, is sorely missing. The algorithm we propose in this paper can be used to solve such complex-symmetric eigenvalue problems.

This paper is organized as follows. The remainder of this section introduces the no-inverse approach and outlines some of its advantages. In Section 2 we describe in greater detail the shift-without-invert method, including comparisons to other partitioned approaches, and some heuristics towards an automated partition method. In Section 3 we present several large-scale calculations illustrating the potential for improved parallelization and reduced computational time of the proposed method, including two major applications originally reported in [1]. Our current calculations show better than five-fold net savings in computational time compared to our previous method.

The method proposed here partitions the eigenvalue problem (1) into parallel sub-problems, then rigidly shifts the operator

$$H_{KS} - i\Gamma \tag{2}$$

into each different partition of the eigenspectrum. In contrast to more commonly used shift–invert approaches, such as [4–7], we avoid matrix inversions. Thus our algorithm involves no linear equation solving via *LU* decomposition or iterative linear equation solvers.

Existing spectrum partition algorithms (e.g. [6,7]) partition the large eigenproblem into parallel sub-problems, but must still face the complexity associated with shift-inverse. For our problem, the matrix (2) is very large, and furthermore, the matrix H_{KS} in TRANSEC is never explicitly computed or stored [1,2], even in a standard sparse format; it is instead represented as a matrix–vector product formula. These facts make inversion *via ILU* factorization infeasible (even if it may be possible to store the sparse H_{KS} explicitly, the cost of factorization can still be prohibitive due to the matrix size). Sparse inversion using an approximate inverse (AINV) preconditioner [11,12] may be another option, but both the AINV and the ILU preconditioners are not straightforward to apply; moreover, even with the increased coding complexity and the expected increase in memory cost, the overall CPU time is not necessarily reduced.

TRANSEC works with the Hamiltonian H_{KS} from PARSEC, which is defined on a real-space grid [2]. The computational cost of the iterative diagonalization of Eq. (1) using a Krylov-type method such as ARPACK scales like $O(Nn_r^2)$, where n_r is the total number of eigenpairs found [13,14]. Because of the sparsity of the Hamiltonian from a real-space method, multiplying the Hamiltonian by a trial vector contributes only linearly in the Hamiltonian dimension N to the cost of (1). Yet this matrix–vector application is the critical source of parallelization in real-space methods [2]. By contrast, the quadratic scaling in n_r is associated with orthogonalizing the growing subspace of eigenvectors, as well as with subspace rotations necessary to carry out the Rayleigh–Ritz refinement process [15]. Whereas the Hamiltonian dimension depends on both the system volume V (number of atoms) and the grid spacing h as $N \approx V/h^3$, the number of eigenpairs n_r needed to converge the $T(E)$ calculation typically scales only with the number N_e of electrons in the system. Because the system volume, number of atoms, and number of electrons typically scale together, the cost of our original method [1] grows cubically with system size V, whereas the parallelizability improves only linearly with V. Worse still, parallel scalability deteriorates when a large n_r is needed for a given N, for example in systems with a high average electron density per unit volume, N_e/V.

To overcome these problems, we present here a spectrum partitioning method that enables the solution of Eq. (1) with better parallel scalability. To avoid the problems associated with inverse of large matrices mentioned above, we approach this problem by shifting without inversion. The no-inverse approach allows us to parallelize our transport method over energy as well as over the real-space grid, significantly improving the overall parallel performance, as will be shown in Section 3.

2. The shift-without-invert partition algorithm

Note that we need to compute the n_r eigenvalues with smallest-real-part[3] and their associated eigenvectors of Eq. (1). To partition this computation, one main feature of the present scheme is to shift the matrix (2) by some strategically placed real rigid shifts $E_{S,j}$ and transform Eq. (1) into a sequence of p sub-problems:

$$\left(H_{KS} - i\Gamma - E_{S,j}\right) U_k = (\epsilon_k - E_{S,j})U_k, \quad 1 \le j \le p, \tag{3}$$

which can be solved parallelly. For each shift $E_{S,j}$, we solve for $n_{r,j}$ number of eigenvalues closest to the shift, where each $n_{r,j}$ is only a fraction of n_r. After combining a sequence of such partitions j with different values of the shifts $E_{S,j}$ and solving for $n_{r,j} < n_r$ eigenpairs on each partition, we obtain the desired total n_r eigenpairs. To compute all n_r eigenpairs, it is clearly necessary that

$$\sum_j^p n_{r,j} \ge n_r. \tag{4}$$

The number $n_{r,j}$ of eigenvalues computed in each sub-problem can in theory be made much smaller than the total n_r, hence we mitigate the $O(Nn_r^2)$ complexity into a sequence of p parallel sub-problems, each with only $O(Nn_{r,j}^2)$ complexity.

By breaking (1) into a sequence of mostly independent sub-problems, the present scheme greatly enhances the parallelization of our real-space method, as we will demonstrate. Moreover, partitioning allows "continuation" runs to expand a domain of

[2] By "weakly", we mean both that the anti-Hermitian part $-i\Gamma$ is restricted to the diagonal, and that it is typically small, as discussed around Eq. (5).

[3] The reason is that to first order, conductance is a property of a small range of eigenpairs around the Fermi energy $E_F \equiv \epsilon_{N_F}$, the N_Fth-lowest eigenvalue in the spectrum of H_{KS} [1]. Typically in TRANSEC we choose n_r a few times N_F, and $N_F \sim N_e/2 \ll N$, where N_e is the number of electrons in the simulation, and $N \approx V/h^3$ is the dimension of H_{KS}. Hence in practice the required eigenpairs are identical to the n_r eigenpairs with smallest-real-part. (In this paper any "ordering" of the eigenvalues always refers to ordering by the real parts.)

previously-solved eigenpairs from n_r to a total $n'_r > n_r$, by computing just the $n'_r - n_r$ previously unsolved eigenpairs as a new partition.

A single partition's complexity is quadratic in $n_{r,j}$, thus it is theoretically possible to solve for a total of n_r eigenpairs in a time that scales linearly in n_r, i.e. $O(n_r)$, when we partition the spectrum into as many finer parts as necessary. In particular, this in theory can lead to far better than the "ideally" linear parallel speedup of non-partitioned methods. In practice, though, our approach faces challenges owing to the increased computational demand associated with computing interior eigenpairs. Our scheme must also cope with uncertainty in choosing the shifts $E_{S,j}$, since the eigenvalues and their density distribution are unknown *a priori*.

The applications we present in Section 3 illustrate in practice both the potential, and the possible setbacks, of the present partitioning scheme. These results show the possibility of super-linear parallel speedup, or equivalently, saving CPU time *via* partitioning, even compared to un-parallelized calculations. In particular, Sections 3.2 and 3.3 present two major applications containing Au(111) nanowires from Ref. [1], each of which we compute here in far less CPU time than in Ref. [1], on the same hardware. The third major application from Ref. [1], a C_{60} molecule between Au(111) leads, is not presented here. In this case, one interior partition out of four had difficulty converging using the same options passed to PARPACK.

The main reason is that Lanczos-type methods (including PARPACK without inversion) are better suited to converging exterior eigenvalues. For eigenvalues located far interior to the spectrum, Lanczos-type methods may suffer from slow convergence, which can worsen when the interior eigenvalues are clustered. With our partitioned method, we call PARPACK with the 'SM' (smallest magnitude) option, iterating over a much smaller dimension subspace compared to a non-partitioned method. Thus our approach is most effective when eigenvalues are not highly clustered.[4] For the unconverged partition in the C_{60} test application, the requested number of eigenpairs per Ry is over 2200. In comparison, the average requested eigenpairs per Ry for the Au nanowire examples is less than 1340, while for the C chain it is less than 250. The more eigenpairs requested per Ry, the more clustered some interior eigenvalues can become. This helps explain why we encountered convergence difficulty for one partition in C_{60} yet had excellent results for the other applications. For problems with highly clustered interior eigenvalues, a pure no-invert approach as developed here may not be optimal. In this case we should resort to applying shift-with-invert, that is, apply the more complicated 'inverting' techniques only when the no-invert approach encounters difficulty converging some partitions. This combination of no-invert and invert will be of future development. We emphasize that, for many problems in quantum transport where the interior eigenvalues are not highly clustered, our no-invert approach provides a greatly simplified alternative to shift–invert.

We implement the shift-without-inverse scheme to study quantum transport with the TRANSEC [1] code. Here, the anti-Hermitian part $-i\Gamma$ of (2) is a relatively small perturbation to H_{KS}, so the imaginary parts $\Im\{\epsilon_k\}$ of the eigenvalues are bounded, and much smaller than the magnitudes of the real parts,

$$|\Im\{\epsilon_k\}| \ll |\Re\{\epsilon_k\}|.$$

Specifically, the complex eigenvalues in our TRANSEC applications are given by

$$\epsilon_k = \langle U_k|H_{KS} - i\Gamma|U_k\rangle \approx \epsilon_k^{KS} - i\langle U_k^{KS}|\Gamma|U_k^{KS}\rangle. \tag{5}$$

[4] Of course, other factors like the assignment of grid points to cores, numerical roundoff, and convergence tolerance also influence convergence in practice.

Here the U_k^{KS} are unperturbed KS eigenvectors (i.e., those of H_{KS} alone),

$$\epsilon_k^{KS} \equiv \langle U_k^{KS}|H_{KS}|U_k^{KS}\rangle \in \Re$$

are the corresponding unperturbed KS eigenvalues, and the approximate equality can be justified by first-order perturbation theory. Therefore

$$0 \le -\Im\{\epsilon_k\} \le \max\{\Gamma\},$$

that is, the whole spectrum lies near the real axis, and TRANSEC's eigenproblem Eq. (1) is only weakly non-Hermitian. Consequently, we only find it necessary to choose appropriate shifts $E_{S,j}$ on the real axis. This choice of real shifts simplifies the partitioning of the spectrum.

It is worth mentioning that our scheme should also be applicable to more strongly non-Hermitian eigenproblems. In such cases one needs to choose complex shifts to compute eigenvalues with larger imaginary parts, and a two-dimensional partitioning $\{E_{S,j,k}, n_{r,j,k}\}$, where the j and k indexes represent real and imaginary parts of the shifts, should be necessary to cover the whole eigenspectrum.

2.1. Rationale of a "without-invert" approach

Our partitioned algorithm belongs to the divide-and-conquer methodology. As is well-known, a divide-and-conquer method in theory is more suitable for parallel computing than non-partitioned counterparts. The rationale behind our partitioned method for eigenvalue problems is also to exploit the potential parallel scaling efficiency. More specifically, since a standard non-partitioned sparse iterative eigenalgorithm for computing n_r eigenpairs of a dimension N matrix has complexity $O(Nn_r^2)$, if we partition the wanted spectrum into p parts, then for each part the complexity reduces to $O(N\left(\frac{n_r}{p}\right)^2)$, so in theory the total complexity reduces to

$$p \cdot O\left(N\left(\frac{n_r}{p}\right)^2\right) = O\left(N\,\frac{n_r^2}{p}\right). \tag{6}$$

We note these theoretical results reference only the number p of partitions, not the number of cores, so (6) could theoretically be attained even with p serial runs on a single core. But there is very little cost to parallelizing the shift-without-invert scheme because the p partitions can be computed in embarrassingly parallel (i.e., entirely independently of each other), as indeed was assumed to derive Eq. (6). The only coordination or communication required among partitions is to combine the results after the eigensolution steps, and subsequently to fill in any missed eigenpairs at the interfaces among partitions, as described below. Therefore, we may also consider p as the number of available processors or CPU cores, which on a modern supercomputer can readily reach over a few thousand (or we could choose p larger than the number of cores by running additional parts in series in order to benefit from the $\sim 1/p$ scaling). So the ideal complexity could reach even

$$O\left(Nn_r\right), \tag{7}$$

when $p = O(n_r)$.

The superior theoretical scaling efficiency associated with partitioned methods is the driving force behind the eigenvalue partition algorithms, represented by [6,16], and [7].

Both Refs. [6] and [7] utilize the shift–invert operation within the framework of the Lanczos method; the earlier such decomposition idea traces back to [17]. The inverse operations require solving linear equations, which are usually realized by calling either an iterative linear equation algorithm such as CG, or

a sparse direct solver [18], in particular the MUMPS package [19]. However, it is well-known that solving the shifted linear equations involves significant efforts, both in computational cost[5] and especially in the algorithm development. A sparse direct solver is the more stable and more straightforward choice when the dimension is not large; however, for very large dimension linear equations, a direct solver becomes impractical and an iterative solver often is needed. But any iterative linear equation solver is not a black-box method that can be used straightforwardly. Rather, there are several parameters related to a given solver that must be tuned [20,21], and such task of choosing parameters is further complicated by the fact that the shifted equations are often ill-conditioned. The so-called "preconditioned" eigen-algorithms are usually more complicated to use than the preconditioned linear solvers they employ to solve the shifted linear equations.

Furthermore, the Lanczos method employed in [17,6,7] may suffer from missing eigenvalues, especially when there are clustered eigenvalues and when subspace-restart is used to save memory; therefore, quite complicated post-processing operations are required to find any eigenvalues missed by a previous Lanczos run within a partition. To guarantee no eigenvalues are missed in a partition, accurate eigenvalue counts on an interval must be calculated in [6,7]. This is done by resorting to the Sylvester inertia theorem, which would require a sequence of Cholesky decompositions. The $O(N^3)$ complexity associated with a Cholesky decomposition thus can significantly restrict the dimension of eigenvalue problems to which the partitioned methods in [6,7] can be applied.

To avoid these difficulties associated with inversion, the spectrum slicing method in [16] opts to apply Chebyshev–Jackson polynomial filters. However, very high order of degree (such as $\geq$1000) polynomials are used to achieve the desired filtering, resulting in significant computational cost for the filtering. The method in [16] seeks to avoid expensive Cholesky decompositions for counting eigenvalues. This saves computational cost, but increases the complexity of the algorithmic design, since several post-processing procedures are needed to guarantee finding all eigenvalues on a partition slice. The method in [16] is applied to only a relatively small number of partitions; extending it to many partitions could encounter difficulties, owing to the complexity of applying high order degree polynomial filters, and the requirement of finding all eigenvalues when eigenvalue counts are unknown.

As discussed earlier, the main advantage of a partitioned eigenvalue algorithm should be its applicability to as many partitions as possible, therefore, we adopt a different approach here than in [16] to avoiding inversion. In addition, the partitioned eigen-methods cited above are all restricted to Hermitian eigenproblems, whereas our method is applicable to non-Hermitian eigenproblems, such as Eq. (1) in TRANSEC.

Although the scheme we propose here could be used with any $O(Nn_r^2)$ iterative eigensolver, our implementation makes use of the well-received eigenvalue package ARPACK [13,22], which arguably remains the best public domain solver for large non-Hermitian eigenvalue problems. ARPACK can solve a standard eigenvalue problem such as (1) without performing inverse, simply by applying implicit polynomial filters [23]. In TRANSEC we actually call PARPACK—the parallel version of ARPACK using MPI.

As mentioned earlier, the $O(Nn_r^2)$ scaling complexity of PARPACK leads to inefficiency when n_r is large. Our solution is to decompose the spectrum into smaller chunks and solve parallelly on each chunk for only a small number of eigenvalues. When computing a relatively small number of eigenvalues, PARPACK enjoys excellent scalability due to fewer basis vectors needing re-orthogonalization (and thus fewer inner-products). This, coupled with the overall stability of ARPACK, makes PARPACK the best available choice for our partitioned subproblem (3). In (3) the number $n_{r,j}$ of requested eigenvalues is only a very small fraction of the dimension of the Hamiltonian matrix.

However, we encounter an immediate difficulty with the partition approach: although PARPACK provides options to specify which eigenvalues to compute, such as SR/LR (smallest/largest real part) and SM/LM (smallest/largest magnitude), the SR/LR/LM are all for computing exterior eigenvalues, and cannot be used to compute the eigenvalues in, for example, an interior partition; and the remaining SM option is only for computing eigenvalues closest to zero.

We overcome this difficulty by combining shifts and the SM option in PARPACK. That is, we strategically place shifts as in Eq. (3) in an estimated region of the spectrum, then request PARPACK to compute the eigenvalues closest to each of these shifts by using the SM option on the shifted operator. With this choice, we can converge both the exterior and the interior eigenvalues, by placing shifts at suitable locations of the spectrum. A downside is that some interior eigenvalues may be very slow to converge without using inverse; if that happens, the overall scalability of our scheme would deteriorate. A possible remedy is to partially integrate inverse operations when such a situation is detected to happen. Partially utilizing inverse is viable because the number of eigenvalues to be computed around a shift is small. The mixed no-inverse plus shift-inverse approach is still expected to be less expensive than using shift–invert on the full wanted spectrum, and will be the subject of our future work. The current paper focuses on the shift-no-inverse approach.

2.2. Computational structure of the shift-without-invert algorithm

Existing partitioned eigenvalue algorithms, including [6,16,7] and our method, all face the challenge of how to partition the spectrum. Since the spectrum is unknown at the start of computation, it is not straightforward to know how to partition it into smaller parts, and harder still to partition in such a way that each chunk would have similar workload for ideal load balancing. This is one of the intrinsic difficulties of any partitioned approach; another difficulty is the handling of partition boundaries (or interfaces) between adjacent partitions, including removing redundant eigenvalues and recovering the wanted eigenvalues that may have been missed on all the partitions. These difficulties, in our opinion, may be the main reason why there exist rather few partitioned eigen-algorithms, even though such an approach can theoretically reach excellent linear scaling complexity. Progress toward an automated algorithm to cope with these difficulties would be a meaningful step forward for approaching the improved theoretical scaling efficiency promised by a partitioned method.

In the next sections we suggest simple techniques to address the two intrinsic difficulties mentioned above. Note that the approach as it is sketched here is neither optimized, nor robust enough to run entirely without human input. Thus our approach is not yet a fully automated algorithm. But the present heuristics are straightforward to implement and may serve as an initial step toward a fully automated partitioned eigen-algorithm. Note also that the results in Section 3 below are special in that we obtained them with foreknowledge of the eigenspectrum from our previous work [1], and therefore did not need to resort to these approximate methods to estimate the $n_{r,j}$ and $E_{S,j}$.

2.2.1. Choosing shifts and partitions

We choose in advance the total number p of partitions, based on how many cores are available for parallelization, and based on the

[5] In fact, we have in the first place chosen eigensolution as an efficiency-enhancing intermediate step towards our actual goal of inverting $E-(H_{KS}-i\Gamma)$, as discussed around Eq. (1) and further in Ref. [1].

general consideration that performance improves as p increases (as seen for actual calculations in Section 3) until the overhead of choosing the partitions begins to dominate. Each partition is associated with a shift $E_{S,j}$ together with a number $n_{r,j}$ of eigenpairs to solve; in the jth partition, we compute the $n_{r,j}$ eigenpairs closest to the shift $E_{S,j}$.

To estimate initial locations for the partitions, we call PARPACK to compute just two exterior eigenvalues—the one with the largest real part and the one with the smallest. This is not expensive since PARPACK is generally very efficient for converging a small number of exterior eigenvalues. Our goal in TRANSEC is to compute the lower end of the spectrum (see footnote 3), so once we estimate these two eigenvalues (denote their real parts as $r_{\min}$ and $r_{\max}$), we can estimate the interval to be partitioned as a certain proportion of the full eigenspectrum located at the lower end, for example:

$$\left[r_{\min},\ r_{\min} + \frac{n_r}{N} \cdot (r_{\max} - r_{\min})\right].$$

However, this estimate assumes uniform density of the eigenspectrum, so it could lead to too many or too few eigenpairs within the partitioned range. To improve, we can estimate the density of eigenpairs by solving for a few eigenpairs at several sample points throughout the spectral region of interest.

We next discuss the challenge of load balancing. Because PARPACK is based on the restarted Arnoldi/Lanczos method, interior eigenvalues are more expensive to converge than exterior ones when we call it without inversion. Clearly, using equal $n_{r,j} \equiv n_r/p$ on each partition j would result in poor load balancing. Instead, it is strongly advisable to reduce $n_{r,j}$ as the partitions move into the interior of the spectrum. As a simple heuristic, we use the formulas at lines 4–8 in the pseudocode shown in the next section to do this. After choosing $n_{r,j}$, we must estimate $E_{S,j}$ accordingly, as for example in line 9. Here `round()` and `ceil()` are the standard rounding and ceiling functions, respectively. The formulas, derived by approximate fitting to actual timing data, provide cheap estimates of the optimal $n_{r,j}$ for given j and p that result in far improved load balancing. Still, these formulas are only heuristics, so we expect further refinements such as sampling the local eigenvalue density would improve load-balancing significantly.

We now address the two issues mentioned above relating to boundaries between partitions: first, some eigenvalues may be computed twice in adjacent partitions, resulting in redundancy. Second, some wanted eigenvalues may be outside the range of all partitions, leading to holes in the computed spectrum. In practice, these two issues compete; tolerating slight redundancy may be preferable so as to minimize the risk of holes. Holes are the more severe problem because filling them necessitates further PARPACK calls.

To address the redundancy problem, we first identify eigenpairs having the same eigenvalue within a small numerical tolerance, and then perform a Gram–Schmidt process to bi-orthogonalize their computed eigenvectors. As a result of the Gram–Schmidt process, any eigenvector linearly dependent on the previous eigenvectors is removed, in this case we also remove its associated redundant eigenvalue. The search and subsequent Gram–Schmidt process can run through all eigenpairs, or only over adjacent pairs of partitions.[6]

The problem of missing eigenvalues is harder to address. Since we avoid any Cholesky decompositions, we cannot apply the Sylvester inertia theorem to get eigenvalue counts on a partition. Aside from the high cost of Cholesky decompositions, our eigenvalue problems are non-Hermitian, so the Sylvester inertia theorem does not apply. In addition, TRANSEC and PARSEC avoid explicitly computing or storing H_{KS} or other $N \times N$ matrices, so a matrix decomposition such as Cholesky is inapplicable. Some techniques have been recently proposed [24,25] to estimate the number of eigenvalues in a given interval without computing them. But these provide only approximate counts, and are again intended for Hermitian eigenproblems, thus cannot apply to (3). Next, we instead propose simple heuristics to identify and fill holes.

2.2.2. Pseudocode of the partitioned shift-without-invert algorithm

Here we present first heuristics towards more automated hole-filling approaches. The structure of our algorithm is presented as Algorithm 1, which we have designed with the quantum transport application of TRANSEC in mind. Adjustment may be called for to optimize the algorithm for other applications.

Some useful inputs to the algorithm include the total dimension N of the Hamiltonian and the total eigenspectrum range $r_{\min}$ and $r_{\max}$, as mentioned above. In quantum transport applications, one can also use the lowest Kohn–Sham eigenvalues ϵ_k^{KS} (those of H_{KS} only, without $i\Gamma$) to estimate the eigenvalue spacing and the lowest eigenvalue of (2). These are necessary prerequisites to quantum transport, obtained when solving the KS equations of DFT, and are valid approximations to ϵ_k, as discussed around Eq. (5). Still, the algorithm would benefit from non-automatic human insight to gauge the validity of such an approximation, or further corrections such as higher-order perturbation theory to improve on it.

Following Algorithm 1, one first can use the $n_{r,j}$ from lines 4 to 8 to estimate optimal shifts $E_{S,j}$, as in line 9. Ideally, these shifts would be chosen to minimize redundancy while still avoiding holes, both between partitions, and at the edges of the overall range of interest. Or one can improve on line 9 by estimating the eigenspectrum distribution, as described above. Any such knowledge of the spectral distribution could prove important to the quality of the initial shift choices.

Next, one computes the solution to Eq. (3) using the chosen $n_{r,j}$ and $E_{S,j}$. The Gram–Schmidt process shown in Algorithm 2 should then be performed to remove redundancies. As mentioned above, in practice a few redundancies are desirable to minimize holes. In fact, we find a reliable heuristic to detect holes is the absence of even a single redundancy between adjacent partitions. If the Gram–Schmidt process detects holes, we next apply additional iterations to "fill in" the holes, by inserting new partitions between existing adjacent ones that lack redundancy, as in lines 17–22 of Algorithm 1. This approach is reminiscent of an adaptive grid algorithm.

Hole-filling carries computational costs and overheads, but these must be weighed against the more expensive probing that would be needed to get an accurate count of the number of eigenvalues in advance.

3. Application of the partitioned shift-without-invert algorithm

In this Section, we present benchmark timing results of the shift-without-invert scheme compared to the standard single-partition TRANSEC method of Ref. [1] for several large $T(E)$ calculations.

We define parallel speedup η according to the usual convention, except with two generalizations. First, because the partitions j run independently, we report speedup either based on total CPU time, or on the longest elapsed wall-time:

$$\eta_{CPU} \equiv \frac{T_{CPU,0}}{T_{CPU}}, \qquad \eta_{wall} \equiv \frac{T_{wall,0}}{\max_j\{T_{wall,j}\} \cdot N_{cores}}. \tag{8}$$

[6] In theory, a brute-force search for degenerate eigenvalues across all partitions could cost quadratically in n_r, just as does the orthogonalization step in PARPACK's Arnoldi algorithm. But one must carry out this search and the Gram–Schmidt step only as post-processing steps *after* each solution of Eq. (3), rather than iteratively within Eq. (3). So in practice, removing redundancy consumes only a small fraction of CPU time. Moreover, one can restrict this search to adjacent partitions, since redundancy normally does not extend beyond the nearest partition.

Algorithm 1 Partitioned shift-without-invert eigen-algorithm, with automated hole-filling heuristic

1: p := Total number of partitions, $\{E_{\min}, E_{\max}\}$:= extremal eigenvalues
2: **for** $j = 1 \to p$ **do**
3: Estimate number of target eigenpairs $n_{r,j}$ for partition j:
4: **if** $j \le p/2$ **then**
5: $n_{r,j} = \texttt{round}\left(\frac{2}{1.5+\frac{j-1}{p}} \cdot \frac{n_r}{p}\right)$
6: **else**
7: $n_{r,j} = \texttt{ceil}(\frac{2n_r}{p}) - n_{r,(p-j+1)}$
8: **end if**
9: Estimate energy shift $E_{S,j}$ for partition j: $E_{S,j} = E_{S,j-1} + \left(\frac{E_{\max}-E_{\min}}{N}\right)\left(\frac{n_{r,j}+n_{r,j-1}}{2}\right)$
10: Solve eigenvalue problem (3) with the updated values $n_{r,j}$, $E_{S,j}$
11: **end for**
12:
13: Set a conditional flag "holes" on each interface between partitions to true
14: **repeat**(while any "holes" flag is true)
15: Combine partitions and check for holes (Algorithm 2)
16:
17: **for all** interfaces j on which the flag has value "holes" **do**
18: Create a new partition j' between partitions $j-1$, j; $p = p+1$;
19: Choose $E_{S,j'} = \frac{1}{2}\left[\max_i \Re\{\lambda_i^{(j-1)}\} + \min_i \Re\{\lambda_i^{(j)}\}\right]$, where i indexes all eigenvalues $\lambda_i^{(k)}$ on partition k;
20: Choose $n_{r,j'} = \frac{3}{2}\left[\frac{\min_i \Re\{\lambda_i^{(j)}\} - \max_i \Re\{\lambda_i^{(j-1)}\}}{(E_{\max}-E_{\min})/N}\right]$,
21: Solve (3) using $n_{r,j'}$, $E_{S,j'}$.
22: **end for**
23: **until** No "holes" remain.

Algorithm 2 Combine partitions and remove redundancies/test for holes

1: **for all** Computed eigenpairs $\{\lambda_k, v_k\}$, across all partitions **do**
2: Check whether eigenvalue $\lambda_m = \lambda_k$ for any $m < k$:
3: **for all** Computed eigenpairs $m < k$ **do**
4: **if** $\lambda_m = \lambda_k$ **then**
5: Modified Gram–Schmidt orthogonalization: $v_k = v_k - \texttt{Proj}_{v_m} v_k$
6: **end if**
7: **end for**
8: Compute norm $a_k \equiv |v_k|$ after Gram–Schmidt process;
9: **if** $a_k <$ tolerance **then**
10: Remove redundant eigenpair $\{\lambda_k, v_k\}$;
11: Set flag holes$_{k-1}$ = false ;
12: **else**
13: Normalize: $v_k = v_k/a_k$;
14: **end if**
15: **end for**

Here T_{CPU} is the total CPU time for the parallel run, $T_{CPU,0}$ is the total CPU time for the reference serial run, $T_{wall,j}$ is the elapsed wall time of the jth partition, $T_{wall,0} \approx T_{CPU,0}$ the elapsed wall time of the reference serial run, and N_{cores} the total number of cores in the parallel job (cumulative over all partitions). Note the difference between η_{CPU} and η_{wall} is due to imperfect load-balancing among the partitions, i.e. $T_{wall,k} \neq T_{wall,j}$. Second, because some calculations are too large to run practically on a single core, we sometimes replace the single-*core* reference times $T_{CPU,0}$ and $T_{wall,0}$ in Eq. (8) with single-*partition* reference times $T_{CPU,1}$ and $N_1 \cdot T_{wall,1}$, respectively, that are still parallelized over grid points to N_1 cores.

For ideal parallelization, the speedup factors defined above approach $\eta_{CPU} = \eta_{wall} = 100\%$. In this Section, we will report cases where $\eta > 100\%$ because multiple partitions can actually *reduce* CPU time compared to a single partition, as discussed around Eqs. (6) and (7).

Our shift-without-invert scheme requires a relatively small additional run to combine the results of the separate partitions (Algorithm 2). We neglect this in most timings reported in this section, but including it would not change the qualitative picture we present. A more important caveat is the necessity to choose $E_{S,j}$ and $n_{r,j}$ appropriately, something made harder when one lacks foreknowledge of the distribution of eigenvalues. As mentioned in Section 2.2, the calculations in this Section portray the potential of a somewhat idealized shift-without-invert scheme because we do possess such foreknowledge from Ref. [1]. Therefore, hole-filling was avoided, and the timings we present here are simply cumulative times for the successful partitioned runs. But we note the calculations *do* reflect realistic difficulties such as load-balancing and redundancy.

In general, we sought to equal or exceed the total eigenvalue counts n_r of Ref. [1], choosing $n_{r,j}$ in accordance with Eq. (4), and using lines 4–8 of Algorithm 1 as a starting point for load-balancing.[7] We divided the single-partition eigenspectra obtained in Ref. [1] into consecutive intervals j containing $n_{r,j}$ eigenvalues, and chose $E_{S,j}$ as the midpoint of each interval. We then typically increased the actual eigenpair requests $n_{r,j}$ by ∼5% or 10%, and in some cases[8] further adjusted the partitions, in order to eliminate holes. We combined the partitions according to Algorithm 2, and proceeded to compute transmission $T(E)$ as in Ref. [1]. Because $T(E)$ is sometimes sensitive to the number of eigenpairs used [1], we typically might discard eigenpairs in the combined spectra that were in excess of the number found in the corresponding single-partition runs. In addition to the $T(E)$ comparisons shown below, we also usually compared the final eigenvalue spectra to the single-partition results as another rigorous consistency check.

3.1. C monatomic chain

We applied Algorithm 1 in TRANSEC to compute the transmission $T(E)$ in an identical C monatomic chain structure presented in Ref. [1]. The geometry, shown in Fig. 1(a), consisted of 14 C atoms per electrode, with atomic spacing of 2.6 a_0, and a gap of 4.7 a_0 between the electrodes and central atom, where a_0 is the Bohr radius. The calculation made use of norm-conserving Troullier–Martins pseudopotentials with s/p cutoff radii of 1.46/1.46 a_0 for C. As in Ref. [1], we used Gaussian imaginary absorbing potentials Γ centered on the first and last atoms in the chains, of strength 265 mRy and standard deviation 10.4 a_0.

Following Ref. [1], we used a converged grid spacing $h = 0.6a_0$, resulting in $N = 22{,}100$ grid points. In order to investigate a case with large n_r/N, we requested the lowest $n_r/N = 10\%$ of eigenpairs instead of 5% as in Ref. [1] for a total of $n_r = 2210$ eigenpairs. The eigenvalues range from $E_F - 1.036$ Ry to $E_F + 9.229$ Ry, where E_F is the Fermi level. On a single core with all n_r eigenvalues in a single partition, the calculation took about $T_0 = 14.5$ h. Parallelized to 4 cores via the standard TRANSEC procedure (i.e., a single partition, parallelized over N only), it took

[7] When doubling the number of partitions, this typically means that each partition j is split into two smaller ones.

[8] The C chain and the 2- and 4-partition BDT calculations were successfully partitioned on the first attempt.

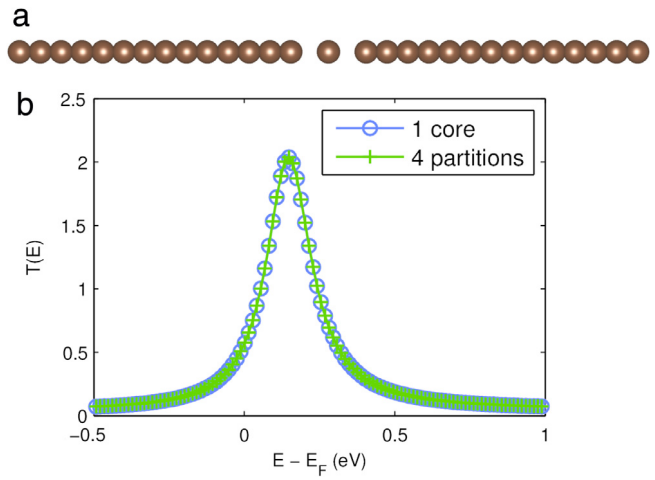

Fig. 1. (a) Structure of the C monatomic chain system considered here and in Ref. [1]. In this work we use 10% of total eigenpairs rather than 5%. (b) TRANSEC calculated results for transmission, $T(E)$, for the C monatomic chain system shown in (a). Results obtained using the partitioned shift-without-invert scheme agree with the single-partition results to better than two decimal places.

$T_{wall,1} = 10$ wall-clock hours, or about $T_{CPU,1} = 40$ core-hours (parallel speedup of just $\eta = 36\%$).

With the shift-without-invert algorithm with 4 partitions on 1 core each (total of 4 cores), the calculation took a total of $T_{CPU} = 9$ core-hours, giving parallel speedup $\eta_{CPU} = 160\%$ compared to the single-core run. The result was obtained in wall-clock time $\max_j\{T_{wall,j}\} = 3$ h and including Algorithm 2, a full $T(E)$ result was obtained within 3.5 h of starting the calculation (parallel speedup of $\eta_{wall} = 290\%$ compared to the non-partitioned parallel run with same 4 cores, or $\eta_{wall} > 100\%$ compared to the single core). The partition parameters were $n_{r,j} = (760, 630, 459, 424)$ and $E_{S,j} - E_F = (1.726, 5.678, 7.496, 8.678)$ Ry. The $T(E)$ results agree with the results obtained with the non-partitioned PARPACK package in Ref. [1], as shown in Fig. 1(b).

3.2. Transmission in Au(111) nanowire electrodes

To gauge the shift-without-invert method's usefulness in practice, we next applied Algorithm 1 to one of our primary test systems of Ref. [1], consisting of Au(111) nanowire electrodes with an Au atomic point contact as the scattering region, and a gap of 9.3 a_0 between the central Au atom and each lead. The system's structure, shown in Fig. 2(a), is identical to that used in Ref. [1]. As in Ref. [1], we used Gaussian absorbing potentials centered at the ends of the two electrodes, with strength 100 mRy and standard deviation 8.5 a_0; and we used a norm-conserving Troullier–Martins pseudopotential for Au with electronic configuration of $5d^{10}6s^{1}6p^{0}$ and s/p/d cutoff radii of 2.77/2.60/2.84 a_0. The real-space grid had $N = 234{,}500$ grid points, of which the lowest $n_r = 2930$ eigenpairs, roughly 1% of the total, were computed. The eigenvalues ranged from $E_F - 0.549$ Ry to $E_F + 1.638$ Ry. As reported in Ref. [1], this single-partition calculation took about $T_{wall,1} = 41$ wall-clock hours on two nodes of Intel E5-2630 machines, each node with two hex-core CPUs (a total $N_1 = 24$ cores). Thus, the total CPU time was $T_{CPU,1} = 980$ core-hours.

In the current work, we have performed the same calculation in four separate partitions of six cores (one hex-core CPU) per partition on the same type of processors, giving a total again of $N_{cores} = 24$ cores. The total CPU time for the four runs reduced to only $T_{CPU} = 320$ core-hours, about three times faster than the single-partition run parallelized only via PARPACK over grid points. The longest of the four partitioned runs took $\max_j\{T_{wall,j}\} = 17$ h, a factor ∼2.5 less elapsed wall-clock time than the PARPACK-only

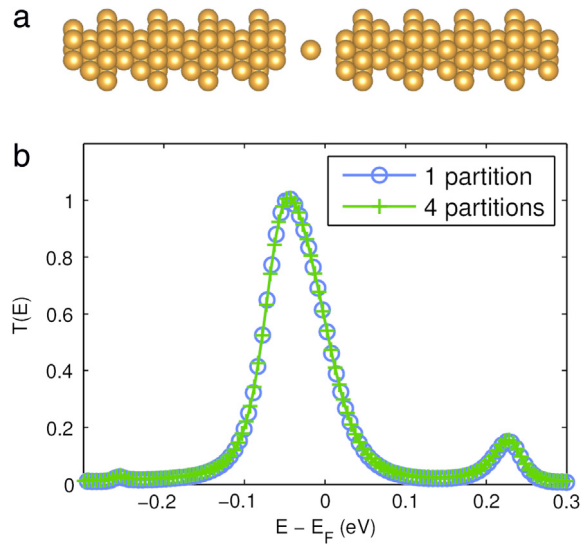

Fig. 2. (a) Structure of the Au(111) nanowire/atom/nanowire system considered here and in Ref. [1]. (b) TRANSEC calculated results for transmission, $T(E)$, for the Au(111) nanowire/atom/nanowire system shown on top. Results obtained using the partitioned shift-without-invert scheme agree with our results obtained with non-partitioned PARPACK in Ref. [1] to better than two decimal places.

run. The partition parameters were $n_{r,j} = (1019, 891, 675, 524)$ and $E_{S,j} - E_F = (-0.045, 0.795, 1.281, 1.535)$ Ry. As shown in Fig. 2(b), the $T(E)$ results obtained by both methods agree to within 0.5% of the peak height, which is well within the typical margin of error of TRANSEC $T(E)$ calculations.

To determine the portion of this observed speedup attributable to the partitioning scheme, we also performed a new *single*-partition reference calculation for the same system and same n_r with just $N_1 = 6$ cores. In this case, the calculation took $T_{wall,1} = 52$ h of elapsed wall-clock time, or a total of $T_{CPU,1} = 310$ core-hours. Compared to this 6-core reference run, the shift-without-invert scheme with 24 cores exhibited a parallel speedup of $\eta_{CPU} = 98\%$ as measured by CPU time, and $\eta_{wall} = 79\%$ measured by elapsed wall-clock time. By contrast, the original single-partition TRANSEC algorithm exhibited speedup of just 31% for 24 cores compared to the 6-core reference run.

3.3. Benzene dithiol (BDT) molecule with Au(111) nanowire electrodes

We also applied the partitioned shift-without-invert scheme to another of our principal test systems from Ref. [1], a molecular junction with the same Au(111) nanowire electrodes and absorbing potentials as in Section 3.2 above and a benzene dithiol (BDT) molecule as the scattering region. The system structure used is identical to that in Ref. [1], except that there an electrode–molecule gap of 3.2 a_0 was used to match a similar calculation by Stokbro et al. [26], and here we use a larger 6.6 a_0 gap, as shown in Fig. 3(a), to demonstrate the gap-dependence of $T(E)$. As in Ref. [1], we used norm-conserving Troullier–Martins pseudopotentials with s/p/d cutoff radii of 1.69/1.69/1.69 a_0 for S, s/p cutoff radii of 1.46/1.46 a_0 for C, and s cutoff radius of 1.28 a_0 for H. The real-space grid had $N = 257{,}000$ grid points, of which the lowest $n_r = 3{,}210$ eigenpairs, about 1% of the total, were computed. These eigenvalues ranged from $E_F - 0.977$ Ry to $E_F + 1.670$ Ry. The single-partition calculation took about $T_{wall,1} = 72$ h on $N_1 = 24$ cores (two nodes) of Intel E5-2630, for a total of $T_{CPU,1} = 1730$ CPU core-hours.

As in Section 3.2, we performed the same $T(E)$ calculation with the shift-without-invert scheme, using four partitions on six cores

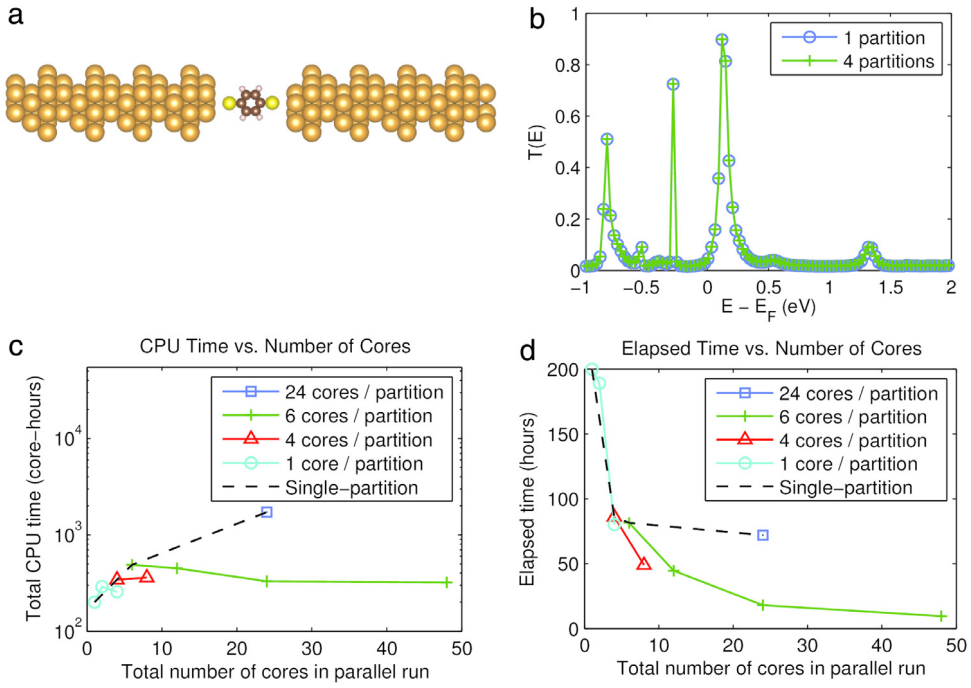

Fig. 3. (a) Structure of the benzene dithiol (BDT) molecular junction between Au(111) nanowire electrodes, as in Ref. [1] except with a 6.6 a_0 molecule–electrode gap. (b) Computed $T(E)$ with 4 partitions compared to the standard single-partition TRANSEC. (c) Total CPU time T_{CPU} vs. number of cores N_{cores} for the $T(E)$ calculation in BDT, shown in semi-log scale. T_{CPU} shown decreasing with N_{cores} implies super-linear parallel speedup, as discussed in the main text. (d) Net elapsed time $\max_j\{T_{wall,j}\}$ (wall-time of longest-running partition) vs. N_{cores}. Results are shown for runs with 24 (blue square [1]), 6 (green pluses), 4 (red triangles), and 1 (cyan circles) cores per partition. Note the leftmost point on each curve is a standard single-partition run, as in Ref. [1]; the remaining points are multiple-partition runs using the shift-without-invert scheme. To compare parallelization schemes, one can compare the elapsed time (part (c) of figure) for different runs at a fixed total number of cores in the calculation. As can be seen from (c), four partitions of six cores each (green pluses) parallelize far better than a single partition of 24 cores (blue square) [1]. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 1
Details of partitioned and non-partitioned BDT calculations, including number of partitions p, number of cores per partition, partition parameters, and timing information.

p	N_{cores}/p	$\{n_{r,j}\}$	$\{E_{S,j}\} - E_F$ (Ry)	$\{T_{wall,j}\}$ (h)	T_{CPU} (core-hours)
1	1	3209	0.34	200	200
1	4	3209	0.34	86	344
1	6	3209	0.34	82	489
1	24	3209	0.34	72	1728
2	1	(2252, 1475)	(0.06, 1.40)	(100, 193)	293
2	4	(2252, 1475)	(0.06, 1.40)	(41, 50)	362
2	6	(2252, 1475)	(0.06, 1.40)	(30, 43)	438
4	1	(1188, 1035, 807, 655)	(−0.24, 0.82, 1.29, 1.56)	(23, 77, 79, 77)	256
4	6	(1188, 1035, 807, 655)	(−0.24, 0.82, 1.29, 1.56)	(6, 18, 16, 15)	330
8	6	(561, 520, 485, 464, 446, 466, 440, 376)	(−0.58, 0.14, 0.65, 0.96, 1.18, 1.37, 1.53, 1.64)	(2, 4, 5, 6, 8, 9, 10, 9)	321

(one hex-core CPU) per partition, with the same type of nodes, for a total $N_{CPU} = 24$ cores. As shown in Fig. 3(b), the partitioned and non-partitioned $T(E)$ agree to within the margin of error of TRANSEC calculations. The total CPU time for the four partitioned runs reduced to just $T_{CPU} = 330$ CPU core-hours, a factor > 5 less than the single-partition run using PARPACK on 24 cores. The longest of the four partitioned runs took $\max_j\{T_{wall,j}\} = 18$ wall-clock hours, a factor ∼4 savings in elapsed time compared to the non-partitioned PARPACK run on the same number of cores. In addition, the overhead of the partitioned runs, i.e., combining the results together by removing redundancy, took less than 4 hours wall-time on 12 cores.

To further investigate the method's parallel performance in large-scale calculations, we performed a series of $T(E)$ calculations with one, four, and six cores per partition. Details of these BDT calculations, including the partition parameters, are given in Table 1. We have summarized the speedup data, our primary result in this work, in Fig. 3(c), showing T_{CPU} vs. N_{cores} and Fig. 3(d), showing $\max_j\{T_{wall,j}\}$ vs. N_{cores}. Each curve shown has a fixed number of cores per partition, N_{cores}/p. Thus one can compare parallelization schemes by evaluating the timings for various curves at a fixed N_{cores} position along the horizontal axis. The number p of partitions is of course given by N_{cores} divided by N_{cores}/p.

The left-most data point displayed in each curve is always a single-partition calculation (serial or parallelized over grid points). Thus, one can visualize the parallel efficiency of the shift-without-invert scheme by comparing the T_{CPU} trend of each curve to the T_{CPU} value of the left-most data point. The runs with six cores per partition (shown as green pluses) exhibit T_{CPU} decreasing with N_{cores}, or equivalently parallel speedup $\eta_{CPU} > 100\%$. For all the curves, T_{CPU} is constant or at most weakly increasing, or equivalently η is near 100% or even better. For example, comparing the four-partition *vs.* single-partition runs with $N_{cores}/p = 6$ cores per partition, we see that parallel speedup was $\eta_{CPU} = 150\%$ by CPU time and $\eta_{wall} = 120\%$ by elapsed time. Moreover, the single-partition run with 24 cores (shown as a blue square) from Ref. [1] has T_{CPU} and T_{wall} far greater than the shift-without-invert results

with the same N_{cores}. Although the original TRANSEC method has been designed to handle very large calculations, in this case our original single-partition result with 24 cores was clearly over-parallelized for the size of the given calculation.

The dashed line connects the left-most point of each curve, and so represents the speedup trend of the standard TRANSEC algorithm with parallelization only over grid points [1]. Moreover, contrasting the trend of the dashed curve to each solid curve vividly illustrates how the limitations of PARPACK-only parallelization can be overcome by shift-without-invert partitioning of the eigenspace. These speedup results are particularly noteworthy because the BDT junction between Au(111) nanowire electrodes is a challenging nano quantum transport system.

4. Conclusions

We have developed a partitioned shift-without-invert scheme that significantly improves the performance of large iterative partial diagonalization algorithms. This scheme can theoretically reduce the computational cost from $O(Nn_r^2)$ to $O(Nn_r)$, as noted in Eq. (7). In practice, we have illustrated with $n_r/N = 10\%$ (Section 3.1) that the shift-without-invert time T_{CPU} can indeed be less than the single-core time T_0, and even with $n_r/N \approx 1\%$ (green pluses in Fig. 3(c)) less than the single-partition time $T_{CPU,1}$. The proposed scheme adds another level of parallelization (over the spectrum and the spatial grid), which provides significant improvement over the already good parallelization (over spatial grid only) implemented in TRANSEC. As a result, even with the non-optimized partitions and parameters, we have readily obtained a factor >5 improvement in CPU time for a large TRANSEC calculation by switching from one to four partitions with the same number of cores, as shown in Fig. 3(c). The shift-without-invert scheme provides a far simpler overall structure compared to other partitioned methods. This is particularly true when compared with partitioned approaches that utilize inversion, which require significant additional effort related to solving shifted linear equations. Moreover, our scheme is applicable to non-Hermitian problems, for such eigenvalue problems very few parallel algorithms have been proposed. Finally, our scheme also enables continuation runs, so that previously converged eigenpairs need not be discarded, but instead we simply place new shifts in unexplored regions of the spectrum to compute desired new eigenpairs. The shift-without-invert scheme is expected to be applicable to a wide range of iterative eigenvalue problems: since we base our partitioned solver on PARPACK, it inherits the remarkable robustness and generality of the PARPACK package. Thus for a wide range of eigenvalue problems where ARPACK/PARPACK is applicable, one can adapt our partitioned scheme to improve parallel scalability.

Acknowledgments

We would like to thank Leeor Kronik and Oded Hod for helpful comments and discussions. The first author's research was supported in part by the European Research Council and the Israel Science Foundation. The second author's research was supported in part by NSF grants DMS-1228271 and DMS-1522587.

References

[1] B. Feldman, T. Seideman, O. Hod, L. Kronik, Phys. Rev. B 90 (2014) 035445.
[2] J.R. Chelikowsky, N. Troullier, Y. Saad, Phys. Rev. Lett. 72 (1994) 1240–1243.
[3] L. Kronik, A. Makmal, M.L. Tiago, M.M.G. Alemany, M. Jain, X. Huang, Y. Saad, J.R. Chelikowsky, Phys. Status Solidi b 243 (2006) 1063.
[4] K. Meerbergen, A. Spence, Math. Comp. 66 (218) (1997) 667–689.
[5] L. Bergamaschi, G. Pini, F. Sartoretto, Numer. Linear Algebra Appl. 7 (2000) 99–116.
[6] H. Zhang, B. Smith, M. Sternberg, P. Zapol, ACM Trans. Math. Software 33 (2) (2007) article 9.
[7] H.M. Aktulga, L. Lin, C. Haine, E.G. Ng, C. Yang, Parallel Comput. 40 (7) (2014) 195–212.
[8] R. Santra, L.S. Cederbaum, Phys. Rep. 368 (1) (2002) 1–117.
[9] A. Bunse-Gerstner, R. Stöver, Linear Algebra Appl. 287 (1999) 105–123.
[10] R.W. Freund, SIAM J. Sci. Stat. Comput. 13 (1992) 425–448.
[11] Y. Saad, Iterative Methods for Sparse Linear Systems, SIAM Press, 2003.
[12] M. Benzi, J. Comput. Phys. 182 (2) (2002) 418–477.
[13] R.B. Lehoucq, D.C. Sorensen, C. Yang, ARPACK User's Guide: Solution of Large Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods, SIAM, 1998.
[14] G.H. Golub, C.F. Van Loan, Matrix Computations, third ed., Johns Hopkins University Press, Baltimore, MD, 1996.
[15] Y. Zhou, Y. Saad, M.L. Tiago, J.R. Chelikowsky, Parallel self-consistent-field calculations using Chebyshev-filtered subspace acceleration, 74 (6) (2006) 066704.
[16] G. Schofield, J.R. Chelikowsky, Y. Saad, Comput. Phys. Comm. 183 (2012) 497–505.
[17] S.W. Bostic, R.E. Fulton, Comput. Struct. 25 (3) (1987) 395–403.
[18] T. Davis, Direct Methods for Sparse Linear Systems, in: Fundamentals of Algorithms, SIAM Press, Philadelphia, 2006.
[19] P.R. Amestoy, I.S. Duff, J.-Y. L'Excellent, J. Koster, SIAM J. Matrix Anal. Appl. 23 (1) (2001) 15–41.
[20] R. Barrett, M.W. Berry, T.F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H. van der Vorst (Eds.), Templates for the Solution of Linear Systems, SIAM, Philadelphia, PA, 1994.
[21] S. Balay, S. Abhyankar, M.F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, V. Eijkhout, W.D. Gropp, D. Kaushik, M.G. Knepley, L. Curfman McInnes, K. Rupp, B.F. Smith, S. Zampini, H. Zhang, PETSc Users Manual, Technical Report ANL-95/11—Revision 3.6, Argonne National Laboratory, 2015.
[22] The ARPACK-NG project. http://github.com/opencollab/arpack-ng.
[23] D.C. Sorensen, SIAM J. Matrix Anal. Appl. 13 (1992) 357–385.
[24] L. Lin, Y. Saad, C. Yang, SIAM Rev. 58 (2016) 34–65.
[25] E. Di Napoli, E. Polizzi, Y. Saad, Efficient estimation of eigenvalue counts in an interval, 2013. ArXiv e-prints, arXiv:1308.4275.
[26] K. Stokbro, J. Taylor, M. Brandbyge, J.-L. Mozos, P. Ordejón, Comput. Math. Sci. 27 (2003) 151.