J Comput Electron (2015) 14:593–603
DOI 10.1007/s10825-015-0695-z

A comprehensive study of popular eigenvalue methods employed for quantum calculation of energy eigenstates in nanostructures using GPUs

W. Rodrigues[1] · A. Pecchia[2] · M. Auf der Maur[1] · A. Di Carlo[1]

Published online: 9 April 2015

Abstract In this work, we concentrate on the graphics processing unit (GPU) implementation of three different methods that are common among peers in the electronic computational domain. We calculate the energy eigenstates of GaN/AlGaN quantum dots on GPU using the tight-binding approach with a $sp^3d^5s^*$ + spin-orbit parametrization for structures ranging from 8039 atoms to 351,600 atoms corresponding to a Hamiltonian matrix size of around 160,780–7,032,000. We perform an analysis for timing, memory occupancy and convergence on a multi-GPU workstation and a high performance computing (HPC) cluster. We also present comparisons between the multi-GPU system having 4 Nvidia Kepler graphic cards and a HPC cluster where the algorithms are benchmarked on up to 256 CPU cores.

Keywords Eigensolver · Lanczos · Jacobi–Davidson · FEAST · Tight-binding · Atomistic simulation · GPU

✉ W. Rodrigues
walter.rodrigues@uniroma2.it

A. Pecchia
pecchia@ing.uniroma2.it

M. Auf der Maur
auf.der.maur@ing.uniroma2.it

A. Di Carlo
aldo.dicarlo@uniroma2.it

[1] Department of Electronics Engineering, Università degli Studi di Roma Tor Vergata, 00133 Rome, Italy

[2] CNR-ISMN, Via Salaria Km 29,600, 00017 Monterotondo, Rome, Italy

1 Introduction

Today, the simulation of multi-million atom nanodevices are a reality due to the advancements in technology that have made high performance computing widely accessible to computational scientists. Atomistic simulation of nanostructures using the tight-binding (TB) model involves the calculations of quantum energy eigenstates of large-scale, sparse Hamiltonian matrices [1–3]. Such approach can be used, for example, to calculate the electronic states in a GaN/AlGaN quantum dot structure (see Fig. 1) which have important applications in modern nitride-based light emitting diodes [4]. From the energy eigenstates calculation, we can compute many different properties, thus providing a guide for device optimization.

Significant efforts put by computational researchers have resulted in libraries like ScaLAPACK which are collections of high performance parallelized eigensolvers. However, most solvers packages based on direct methods like QR, bisection and inverse iteration, and divide and conquer cannot be utilized as they are meant for comparatively small sized dense eigenproblems, that can be stored in the system memory and are needed to calculate all of the eigenvalues. Hence, iterative methods have to be engaged for large-scale TB calculations that facilitates the calculation of a few eigenpairs by projecting the huge problem onto a much smaller search space.

There are several approaches that can be used to calculate the needed eigen states of the Hamiltonian. Given the variety of possible methods it is still unclear which one is more suited and how their performance compares in a given scenario. However, there are few methods which are more widely used given their implementation feasibility, convergence characteristic, accuracy and reliability. Methods, such as Lanczos, Jacobi–Davidson and conjugate gradient

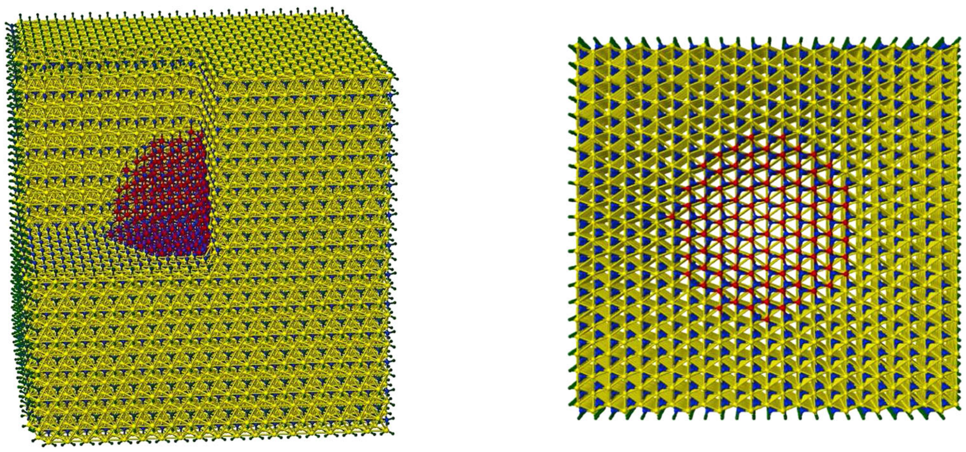
Fig. 1 (*Left*) Cubical wurtzite GaN/AlGaN quantum dot showing the core with 30 % aluminum. (*Right*) A central slice of the cube. Atomistic description: in *yellow* aluminum, in *red* gallium and in *blue* nitride (Color figure online)

are popular and widely utilized in tight-binding calculations [5–7]. Recently, a new method called FEAST based on the density-matrix operator is gaining popularity in the computational community [8,9]. Hence, benchmarking them for recent HPC architectures is of importance for the given application domain.

The Lanczos algorithm is an effective iterative method to find eigenvalues and eigenvectors of large sparse matrices by first building an orthonormal basis and then forming approximate solutions using Rayleigh projection. It reduces a large, complicated eigenvalue problem into a smaller (simpler) one [10,11], explicitly taking advantage of the symmetry of the Hamiltonian matrix. However, the Lanczos method diverges when implemented on a finite precision architecture since the Lanczos vectors inevitably lose their mutual orthogonality [11,12]. Hence, it needs a full reorthogonalization of each newly computed vector against all preceding Lanczos vectors. This, not only greatly increases the number of computations required, but also requires that all the vectors be stored. For large problems, it will be very expensive to take more than a few steps using full reorthogonalization; nevertheless, linear independence will surely be lost without some sort of corrective procedure. Selective orthogonalization interpolates between full reorthogonalization and simple Lanczos to obtain the best of both worlds. Robust linear independence is maintained among the vectors at a cost which is close to that of a simple Lanczos [13,14]. Another way to maintain orthogonality is to limit the size of the basis set and use a restarting scheme by replacing the starting vector with an improved starting vector and computing a new Lanczos factorization with the new vector.

The Jacobi–Davidson method is another popular technique to compute a few eigenpairs of large sparse matrices. It is motivated by the fact that standard eigensolvers often require an expensive factorization of the matrix to compute interior eigenvalues. Such a factorization is infeasible for large matrices in large-scale simulations. In the Jacobi–Davidson method, one still needs to solve inner linear systems, but a factorization is avoided because the method is designed so as to favor the efficient use of iterative solution techniques based on preconditioning [15]. Jacobi–Davidson method belongs to the class of subspace methods, which means that approximate eigenvectors are sought in a subspace. Each iteration of this method has two important phases: the subspace extraction in which an approximate eigenpair is sought with the approximate vector in the search space and the subspace expansion in which the search space is enlarged by adding a new basis vector to it trying to lead to a better approximate eigenpairs in the next extraction phase [16,17].

Conjugate gradient method is widely used to solve many types of problems like the linear algebraic equation, eigenvalue and minimization problem. However, it requires that the coefficient matrix to be both symmetric and positive definite. There are implementations of applying conjugate gradient method to indefinite systems but its stability continues to remain an issue and is therefore not included in this scope of our study [18].

Lately, the FEAST algorithm which takes its inspiration from the density-matrix representation and contour integration technique in quantum mechanics is also being used [19]. The algorithm deviates fundamentally from the traditional Krylov subspace iteration based techniques. This algorithm is free from any orthogonalization procedures and its main computational tasks consist of solving the inner independent linear systems with multiple right-hand sides. The FEAST algorithm finds all the eigenpair in a given search interval. It requires that one provides an estimate for the number of eigenpair within the search interval which often is not possible to obtain beforehand.

Today, larger and faster computing systems are widely accessible. Supercomputers and high-end expensive computing systems are being utilized to accelerate computation in a parallel distributed, cluster or grid computing setting. The advent of GPUs have grasped the attention of most of the scientific computation community with its huge number of computing cores making massive thread level parallel computing possible, resulting in high throughput. Developing algorithms that can ideally scale over such system is an important component for transferring the hardware feature into actual beneficial speedups. In recent times, there has been an extensive effort being put in translating algorithms initially designed for sequential processors to now days HPC system which normally deal with either single instruction multiple data (SIMD) or multiple instruction multiple data (MIMD) scenario. However, a lot of aspects need to be considered to result in speedup while dealing with parallel computing. Hence, often this sequential to parallel transition is not straight forward and requires a deeper understanding of the system architecture and method itself.

There are many challenging questions to be considered in terms of the choice of method employed. Some of these questions include: what method takes the least total computation time and is well suited for GPUs given its limited available resources? Which approach is robust in convergence when used with nanostructure having a dense energy spectrum? Also, in a multi-GPU scenario where data has to be shared among GPUs, we need to identify the implementation that deals well with hardware limitation. For example, the slow host-to-device memory transfer and the limited PCI-E transfer speed. Characteristics of the method like its ratio of compute to memory intensive operation which are needed for a good speedup in hybrid implementations also need to be considered. Finally, we need to find a method that scales best in a multi-GPU distributed setup.

Having identified the aspects that need to be taken into account and proposed a design for parallel computing, we will test and compare each of the above described algorithms for memory utilization, execution time, implementation complexity (feasibility) and convergence. We will benchmark a robust implementation of each of the algorithm on a multi-GPU system as well on a HPC cluster.

2 Implementation

GPU have a limited memory and the peak bandwidth between the device memory and the GPU is much higher than the peak bandwidth between host memory and the device memory therefore, as already shown in our previous work [20] it is crucial to minimize the data transfer between the host and the GPU by keeping the Hamiltonian matrix and the search subspace on the device memory. For this reason, the TB Hamiltonian matrix is converted to a single precision format prior to transfer to GPU's global memory. The algorithms are implemented using mixed single/double precision arithmetic to ensure highly accurate solutions. For III-V semiconductors, every atom has 4 neighbors, the TB Hamiltonian matrix is largely sparse with $sp^3d^5s^*$ parameterization [21] having an average of 40 non-zeros values per row with a standard deviation ranging from 3.0 to 4.0. Given the large and sparse nature of the matrix, we employ the compressed sparse row (CSR) format.

Given an $N \times N$ Hamiltonian matrix (H), its eigenvalues, λ_i, and corresponding eigenvectors, v_i, are defined by the relation,

$$Hv_i = \lambda_i v_i \tag{1}$$

We are interested in finding inner eigenvalues of the energy spectrum, near the energy gap of the nanostructure. In the following subsections, we give a brief overview of the three methods employed in our work.

2.1 Lanczos method

The Lanczos algorithm converges fast to the extreme eigenvalues. Different spectral transformations are used for the extraction of the inner eigenpairs, like spectrum folding or shift-and-invert. We employ the Lanczos algorithm with a simple restart along with the spectral folding technique [22,23]. This strategy avoids reorthogonalization that forces saving of several vectors. So, in general, we compute the lowest eigenpairs of the operator $A = (H - sI)^2$, where s is the chosen spectrum shift [24]. Reorthogonalizing the newly computed vector against all preceding Lanczos vectors takes a lot of resources and it is not done in our implementation. In our implementation, we perform the Lanczos iterations until orthogonality with respect to the initial vector is preserved to an error of 10^{-5}. In this way, the typical size of the tridiagonal matrix becomes of the order of 1000, which can be diagonalized easily using standard routines, obtaining the eigenvalues, λ'_i, and corresponding eigenvectors, w_i. It can be proved that the eigenvalues of the tridiagonal matrix are approximate eigenvalues of A. The projected eigenvector, v_i, can be calculated as $v_i = Q_m w_i$, where Q_m is the transformation matrix whose column vectors are recomputed on the fly by running the Lanczos iteration a second time. The advantage of this fine-tuned Lanczos algorithm is that it resides in very little memory. The detail implemented of the algorithm is described in Ref. [20].

2.2 Jacobi–Davidson method

The Jacobi–Davidson method is an iterative subspace method for computing one or more eigenpairs of large sparse matrices. In this method each iteration has two phases: the subspace extraction and the subspace expansion.

For the subspace expansion phase, given an approximate eigenpair (θ_i, u_i) close to (λ_i, v_i), with $u_i \in U$, where U is the subspace, and $\theta_i = u_i^* H u_i / u_i^* u_i$ is the Rayleigh quotient of u_i, taken as approximate eigenvalue because it minimizes the two-norm of the residual: $\|r\| = \|Hu_i - \theta_i u_i\|$. To expand U in an appropriate direction, we look for an orthogonal correction $t \perp u_i$ such that $u_i + t$ satisfies the eigenvalue equation:

$$H(u_i + t) = \lambda_i(u_i + t) \tag{2}$$

We try to find eigenvalues closest to some given target τ, initially we consider this be the same as the chosen Lanczos shift $\tau = s$. In the above equation,

$$(H - \tau I)t = -r + (\lambda_i - \theta_i)u_i + (\lambda_i - \tau)t \tag{3}$$

$\|t\|$ and $| \lambda - \tau |$ are small and can be neglected. When multiply both sides of Eq. (3) by the orthogonal projection $I - u_i u_i^*$. We have the following equation

$$(I - u_i u_i^*)(H - \tau I)(I - u_i u_i^*)t = -r \tag{4}$$

where $t \perp u_i$. We solve Eq. (4) only approximately using generalized minimal residual method (GMRES) and its approximate solution is used for the expansion of the subspace [25].

To save GPU memory, the process is enhanced by restarting the Jacobi–Davidson method with a few recently found u_i in this way, we restrict the dimension of the search subspace [26]. In order to avoid the found eigenvalues from reentering the computational process, we make sure that the new search vectors are explicitly orthogonal to the computed eigenvectors.

As stated above, we are interested in interior eigenvalues, the Ritz vectors represent poor candidates for restart since they converge monotonically towards exterior eigenvalues. One solution to this problem is using the harmonic Ritz vectors. The harmonic Ritz values are inverses of the Ritz values of H^{-1}. Since our matrix is Hermitian the harmonic Ritz values for the shifted matrix $(H - \tau I)$ converge monotonically towards $\theta_i \neq \tau$ eigenvalues closest to the target value τ. The search subspace for the shifted matrix and the unshifted matrix coincide and hence, it is possible for the computation of harmonic Ritz pairs for any shift. The harmonic Ritz vector for the shifted matrix can be interpreted as maximizing a Rayleigh quotient for $(H - \tau I)^{-1}$. It represents the best information that is available for the wanted eigenvalue and hence, is also the best candidate as a starting vector after restart [27].

The correction equation is solved to an accuracy of just 10^{-1}, it is sufficient enough to keep the number of outer iterations between 4 and 10 with internal restart set to 10. GMRES, although a bit more expensive than the other linear solvers in terms of memory and computation, is chosen as it is found to be more stable in solving the correction equation for our TB Hamiltonian [28,29]. We can further improve the computation by treating the Hamiltonian matrix with a preconditioner. However, the preconditioner will occupy similar memory as the actual matrix and also increase the crucial time consuming matrix-vector multiplications per iteration and hence may not be a wise choice for a GPU-accelerated solver where 10^{-1} accuracy is sufficient. Its development and testing is part of future research.

2.3 FEAST method

The aim of the FEAST algorithm is to actually compute the eigenvectors instead of approximating them, unlike the Lanczos and Jacobi–Davidson method. It yields all the eigenvalues and eigenvectors within a given search interval $[\lambda_{min}, \lambda_{max}]$. FEAST relies on the Rayleigh–Ritz method [19,30] for finding the eigenvector space V in some enveloping space $U \supseteq V$. Let Γ be a simply closed differentiable curve in the complex plane that encloses exactly the eigenvalues $\lambda_1, \ldots, \lambda_m$ and z be the contour point. Using the Cauchy integral theorem, it can easily be shown that

$$VV^* = \frac{1}{2\pi i}\int_{\Gamma}(zI - H)^{-1}dz = Q \tag{5}$$

Next, choose a random matrix $Y \in C^{n \times m0}$, where $m0$ is the size of the working subspace which is slightly larger than m, the number of eigenvalues within the search interval. The expression in (5) leads to a new set of $m0$ independent vectors $Q_{n\times m0} = q_1, q_2, ..q_{m0}$ obtained by solving linear systems along the contour and form $U = QY$. It follows that $U = span(U) \supseteq V$ is a candidate for the space used in the Rayleigh–Ritz method. The matrix U can be computed, for our TB Hamiltonian matrix 3 to 8 integration points are sufficient. Then for each integration point, z, a block linear system $(zI - H)^{-1}U_i = Y_i$ needs to be solved, each with $m0$ right hand sides. Notice that the matrix keeps on changing with z throughout the run.

The FEAST algorithm can be parallelized in several ways. First, the interval $[\lambda_{min}, \lambda_{max}]$ can be split, and each part can be treated separately. Also, for each contour point block linear system can be solved independently from each other, as well as each linear system in principal can be solved in parallel [31]. We haven't parallelized using any of the mentioned strategies. Instead we have parallelized the solver that finds the solution for each linear system using our parallel multi-GPU enhanced techniques since the solution to the block linear system is the most expensive part of the method.

We utilize the conjugate gradient squared method (CGS) to solve the block inner independent linear systems since the cost per iteration of CGS is cheaper than that of GMRES in terms of computation and memory [25,32]. The inner independent linear systems need to be solved to a high accuracy of at least 10^{-6}. For non-converged linear system, the solver can be stopped after a few hundreds of iteration. Often, convergence is improved by using an incomplete factorization method based on Gaussian elimination like incomplete LU (ILU) as a preconditioner matrix [33]. However, for TB Hamiltonian matrix under consideration the ILU factorization with 0 level of fill-in is not sufficient for convergence, if utilized it takes more iterations to converge compared to the case where a preconditioner is not employed and hence we need to perform higher level of factorizations. As the fill-in level in an ILU decomposition increases, the quality of the ILU preconditioner improves. This also changes the sparsity of the preconditioner matrix, preliminary tests using MATLAB implementations have revealed that for good and faster convergence a higher order factorization that generates a preconditioner matrix almost 30–35 times denser than the original TB Hamiltonian matrix is needed. Thus, more accurate ILU preconditioners require more memory, to such an extent that eventually the running time of the algorithm increases even though the total number of iterations in the linear solver decreases. Also, the parallelization of ILU involves a lot of data transfers between the nodes since almost the entire TB Hamiltonian matrix is needed on each node and it takes a noticeable amount of compute time as a fresh ILU factorization is needed to be computed for each contour point as the matrix keeps on changing. Therefore, we do not present a FEAST implementation that utilizes an incomplete factorization based method to generate a preconditioner matrix. To obtain a higher speedup and low memory foot print, parallel preconditioners that are better suited for GPU parallelism must be developed and implemented. This is part of our future research plans.

3 Numerical results and discussion

All benchmarks are performed by analyzing the algorithms to find the lowest 8 conduction energy eigenstates of atomistic quantum dots similar to the one shown in Fig. 1. In our previous work [20], we concentrated particularly on the sparse matrix-vector multiplication timings and discussed the performance of GPUs to find one energy eigenstate using the Lanczos method. Here, we compare different methods and especially focus on their ability to compute multiple eigenpairs. For this reason, we report the wall time of different algorithms to compute 8 energy eigenstates.

The GPU implementation of the algorithms and linear solvers are done utilizing the TB Hamiltonian splitting approach, the mixed complex-real arithmetic matrix-vector multiplication CUDA kernel and all of the parallel GPU implementation techniques and optimizations discussed in Ref. [20]. However, in the case of the FEAST method, the matrix keep on changing with different contour points as $zI - H$ (or $z^*I - H$), therefore, it is not optimal to use the splitting approach since tests have shown that a significant amount of time is spent building the splitted matrix and dropping the zeros.

The Lanczos algorithm has been fully ported to the GPU and vectorized to scale with MPI parallelization on multi-GPU workstations. Similarly, also the Jacobi–Davidson algorithm has been implemented on GPUs, along with GMRES method which is utilized as a linear solver for the Jacobi–Davidson correction equation. In order to spare GPU memory, the subspace vectors have been saved on the host memory. This strategy enables us to treat larger systems at the expense of more device–host communication. A comparison between Jacobi–Davidson algorithm with and without the subspace in the device memory is shown below. Concerning FEAST, we have ported only the linear solver (CGS) to the GPU, given that this is the most time consuming part of the algorithm. In this respect, Lanczos and Jacobi–Davidson can be considered as pure GPU implementations and FEAST as a hybrid CPU/GPU, even though 98 % of the total time is spent on the GPU solving the block liner system.

In order to give a broad perspective, we also compare the GPU performance with a HPC cluster. The relevant details of the test hardware are given below:

– Test System 1 (Multi-GPU workstation): Intel Xeon Processor E5-2620 (6 cores, 2 GHz, Cache 15 MB), 64 GB DDR3 SDRAM (Speed 1333 Mhz) and 2 Nvidia Tesla K40 (Chip Kepler GK110B GPU, Processor Clock 745 MHz, CUDA cores 2880, Memory Clock 3.0 GHz, Memory Size 12 GB, Peak performance 1.43 Tflops) + 2 Nvidia Tesla K20 (Chip Kepler GK110 GPU, Processor Clock 706 MHz, CUDA cores 2496, Memory Clock 2.6 GHz, Memory Size 5 GB, Peak performance 1.17 Tflops) connected on the same PCI-E with an operating system based on Linux kernel 3.0.85.
– Test System 2 (HPC cluster): 2208 compute nodes, each node has 2 Intel Xeon X5570 (4 cores, 2.93 GHz, Cache 8 MB), 24 GB DDR3 SDRAM (Speed 1066 MHz). Nodes are connected through an Infiniband QDR network with non-blocking Fat Tree topology with a total Peak performance of 207 Tflops and having an operating system based on Linux kernel 2.6.32.

3.1 Single and multi-GPU CUDA implementation

As seen from left panel of Fig. 2, on a single GPU Jacobi–Davidson with subspace in host memory performs almost 2× times faster as Lanczos and 13× faster as FEAST. However, when we move from one GPU to a multi-GPU scenario as shown in the right panel, Jacobi–Davidson with a subspace in host memory performs only 1.4× times faster than Lanczos when we search for the first few eigenstates. The decrease in speedup compared to a single GPU implementa-

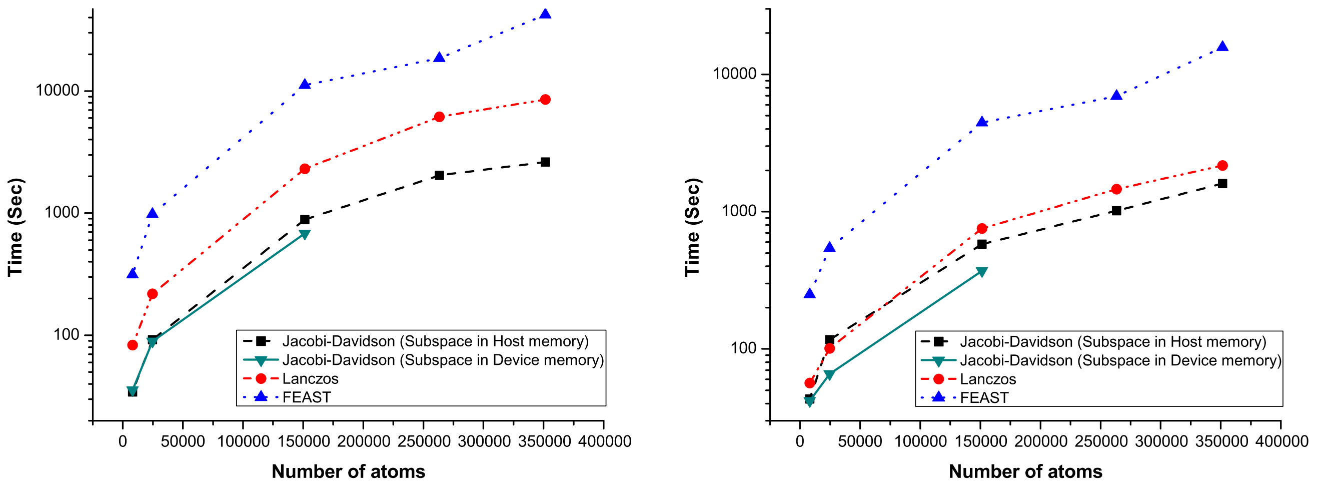

Fig. 2 Time comparison between methods on (*left*) 1 GPU and (*right*) 4 GPUs for the calculation of 8 energy eigenstates

tion is attributed to the fact that the sparse, mix complex-real matrix-vector operations become less significant as seen in Table 1. Also, since the subspace is saved on the host memory it imposes more inter host-GPU data movement than Lanczos as seen from Fig. 4, which is also the ultimate speed limiting factor for any parallel implementation. To attain ideal scaling, there should not be any data dependency or synchronizations between GPUs. Also, there should be enough data to utilize all the GPU cores efficiently. As noticeable from Figs. 2 and 5 with regards to Jacobi–Davidson implementation with subspace stored in device memory, we could only fit up to 151,472 atom quantum dots on GPUs having a memory limit of 5 GB therefore, as already stated, it is crucial to employ the implementation that spares memory by moving the subspace to the host memory. The rest of the discussion in the following subsections corresponds to the Jacobi–Davidson method with subspace stored in the host memory.

Figure 3 shows the scaling of each method over multiple GPUs. We observe that the Lanczos and the FEAST method exhibit a strong scaling for large quantum dots. The ample data movement in the Jacobi–Davidson implementation due to the subspace being stored in host memory impedes its scaling performance.

The profiling results from a data movement perspective for 151,472 atom quantum dot are shown in Fig. 4. These profiling tests have revealed that given the sequential nature of the iterative algorithms and the pure GPU implementation with minimal data transfer, we were unable to obtain any significant memory copy or compute overlap. Only in the case of Jacobi–Davidson method, we obtained 3 % of compute/memory copy overlap since the subspace vectors are stored on the host memory. We expect this number to increase as the size of the quantum dot increases.

We can also notice from Fig. 4 that Lanczos is a compute intensive algorithm as almost 99 % of time is used for computation with minimal data transfer which happens only during the launch as the matrix is loaded onto the GPU memory. Whereas, in the case of Jacobi–Davidson method the host-to-device and device-to-host data transfer account for 15–20 % of the total effective time since the subspace is stored on the host memory. The CGS method used to solve the block linear system within FEAST, imposes an ample amount of device-to-device data transfer accounting to 10–25 % of the total computation time. This can be a potential bottle neck if data transfer has to be performed via the host memory. We attain a peak bandwidth of 7.45 Gb/s between the host and the device.

Tables 1, 2, and 3 shows the profiling results for compute operations of the algorithms for 151,472 atom quantum dot. In all of the three methods, the sparse matrix-vector multiplication is the most important computation task. However, when we go from a single GPU to a multi-GPU implementation for the Jacobi–Davidson method, the dense subspace-vector multiplication gains significance over the sparse Hamiltonian matrix-vector multiplication. We notice in Table 1 that the GPU occupancy for this operation is very low; hence, it would be best to off load this operation onto the CPU. Increasing the warp efficiency will maximize GPU compute resource utilization. The warp efficiency is defined as the ratio of the average active threads per warp to the maximum number of threads per warp supported on a multiprocessor. A low value indicates that there are divergent branches.

As the size of the nanostructure is increased, usually more energy states are needed and these states happen to be closely spaced. This poses a challenge for realistic nanostructure simulations since the eigenvalue happen to be less distinct. Our investigation has shown that Jacobi–Davidson happens to be the most robust method in terms of convergence. Even for closely placed energy states, the algorithm performs fairly well compared to the other methods, typ-

Table 1 Profiler output for 151,472 atom quantum dot, listing the most significant compute operations within Jacobi–Davidson method with subspace stored in host memory

	Computation profile for single GPU			Computation profile for multi-GPU			Shared memory	Registers
	Compute time (%)	GPU occupancy	Warp efficiency (%)	Compute time (%)	GPU occupancy	Warp efficiency (%)		
Mix complex-real, SpM×V product $Mul(H_{real}, q_{complex})$	45.30	0.991	90.89	32.20	0.972	94.05	4096	28
Vector operations $y = y + \alpha x$	15.50	0.976	100.00	12.20	0.942	100.00	0	20
Dense M×V operation	14.70	0.197	89.35	37.40	0.201	89.33	10,240	60
Dot product	13.80	0.497	100.00	8.30	0.482	100.00	1024	28
Shift matrix	3.00	0.998	69.94	1.70	0.997	73.14	0	8

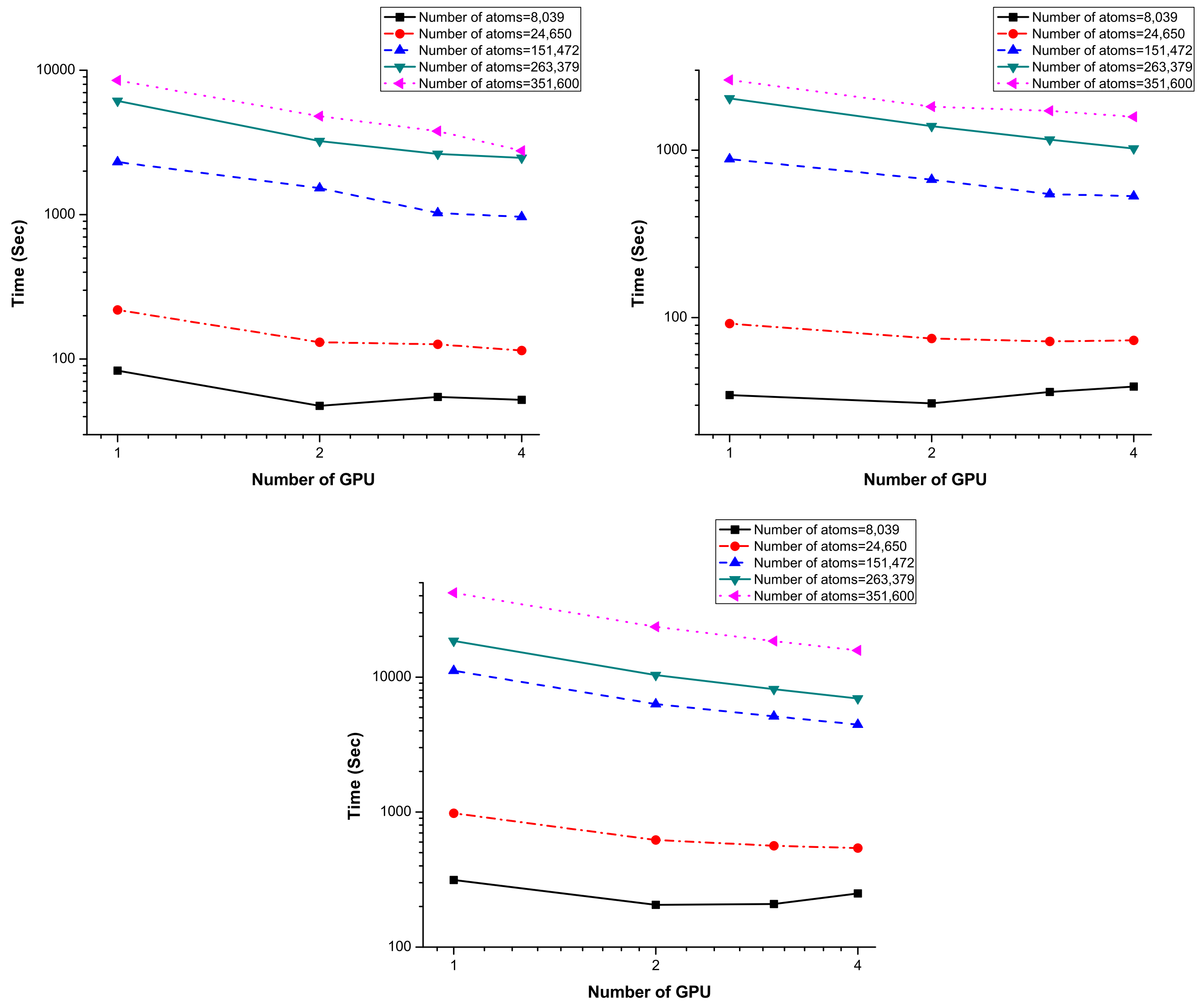

Fig. 3 Scaling of (*left*) Lanczos, (*right*) Jacobi–Davidson (subspace in host memory) and (*center*) FEAST method on 1–4 GPUs respectively

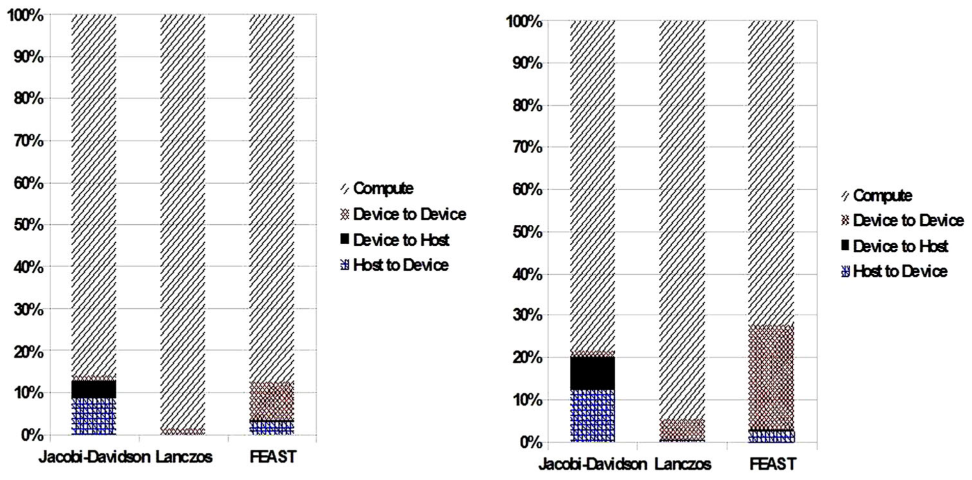

Fig. 4 Percentage of time taken for memory and compute operations on (*left*) 1 GPU and (*right*) 4 GPUs respectively

Table 2 Profiler output for 151,472 atom quantum dot, listing the most significant compute operations within Lanczos method

	Computation profile for single GPU			Computation profile for multi-GPU			Shared memory	Registers
	Compute time (%)	GPU occupancy	Warp efficiency (%)	Compute time (%)	GPU occupancy	Warp efficiency (%)		
Mix complex-real, SpM×V product $Mul(H_{real}, q_{complex})$	84.20	0.941	91.14	82.80	0.924	94.13	4096	29
Mix complex-real, SpM×V product $Mul(H_{imag}, q_{complex})$	3.20	0.876	42.02	3.50	0.829	51.87	0	32
Vector operations $y = y + \alpha x$	7.80	0.781	100.00	8.30	0.748	100.00	0	14

Table 3 Profiler output for 151,472 atom quantum dot, listing the most significant compute operations within the CGS method (linear solver for FEAST)

	Computation profile for single GPU			Computation profile for multi-GPU			Shared memory	Registers
	Compute time (%)	GPU occupancy	Warp efficiency (%)	Compute time (%)	GPU occupancy	Warp efficiency (%)		
Complex SpM×V product $Mul(H_{complex}, q_{complex})$	85.50	0.993	89.83	83.80	0.976	93.07	4096	31
Vector operations $y = y + \alpha x$	11.70	0.973	100.00	13.60	0.923	100.00	0	16–21
Dot product	2.70	0.497	100.00	2.40	0.491	100.00	1024	28

ically 300–600 iteration are sufficient to find the first few energy states. Experience shows that in Jacobi–Davidson for fast convergence the minimum dimension of the subspace can safely be restricted to 4 more than the number of wanted energy states and the maximum dimension needs to be at least 10 more than the number of wanted energy states, i.e. in our case $minimum = 8 + 4$, $maximum = 8 + 10$ since we look for 8 energy states. In the case of the Lanczos method, the convergence is drastically lowered for a dense eigenvalue spectrum. The convergence rate falls as the size of the quantum dot is increased. Usually for big systems around 10,000–20,000 Lanczos iteration are needed to find each energy state. Similarly, in the case of FEAST more contour points and a bigger search space is needed to improve convergence, which also translates into more work and more memory utilization for each FEAST iteration. This is evident from Fig. 2 where for the largest nanostructure the contour points had to be increased from 3 to 5 and the search space from 12 to 14. Typically, 10–25 number of FEAST iteration are sufficient for a good accuracy. Comparing the accuracy of the methods with the direct diagonalization carried out on a small nanostructure, we found that FEAST delivered results

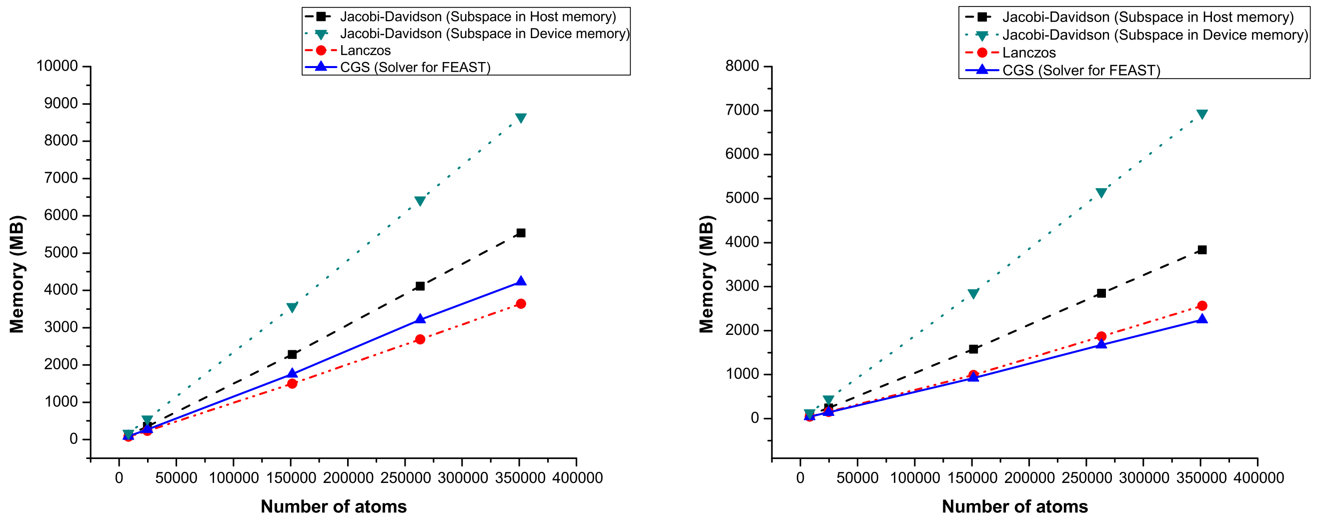

Fig. 5 Memory consumption between methods on (*left*) 1 GPU and (*right*) 4 GPUs respectively

to an absolute accuracy of 10^{-11}. While, Lanczos and Jacobi–Davidson methods delivered to an absolute accuracy of 10^{-6}. Stopping convergence criteria in all the three methods were set to 10^{-5} eV.

Regarding memory occupancy as shown in Fig. 5, a single GPU Lanczos implementation occupies the least amount of memory since we do not store any subspace vectors. Whereas, the slightly higher memory occupancy of CGS used in the FEAST method can be attributed to the original complex TB Hamiltonian matrix since we didn't use the splitting technique. For the Jacobi–Davidson method a subspace of 8 + 10 is needed for basis vectors and another additional space of 8 + 10 vectors is needed for the projection of the Hamiltonian matrix onto this subspace, if the subspace is stored on the GPU the feasible simulation size of the quantum dot is reduced by half. In a multi-GPU system, the TB Hamiltonian is divided equally among GPUs. As the Hamiltonian size is reduced on each node, the subspace and temporary vectors required in the implementation scheme tend to gain importance and takes over the Hamiltonian as the chief memory consumption entity.

One advantage of the Lanczos method over other methods is that since each eigenstate is calculated one at a time, it is possible to calculate the degenerate energy state with just one matrix-vector multiplication, once found this eigenpair is projected out and other unique energy states are calculated. However, Jacobi–Davidson is also found to be robust in this case since it happens to find the degenerate state within a few iterations, in most cases using the harmonic extraction.

3.2 MPI/OpenMP benchmarking on a HPC cluster

A HPC cluster is normally the machine of choice of computational scientists for large scale calculations. Our multi-GPU implementations of the algorithms are developed for a distributive computing environment using MPI and thereby, facilitates the performance benchmarking of the methods on the HPC cluster with a few modifications. As previously described, in Test System 2 each node has a dual quad-core CPU with 23 GB of main memory. We have utilized a hybrid MPI/OpenMP (multi-process/multi-thread) implementation for each of the methods. The benchmark calculations have been performed for 4, 8, 16 and 32 MPI processes with a constant of 8 OpenMP threads on each nodes corresponding to 32, 64, 128 and 256 CPU cores in usage (Fig. 6).

To summarize our findings for small systems Jacobi–Davidson performs on an average 10.2× times faster than Lanczos which further increases to 17.2× with the increase in the system size given the slow convergence nature of Lanczos for closely spaced energy states in large quantum dot. Whereas, in the case of FEAST method, it executes on average 1.6× times slower than Lanczos for small system which increases to 9.3× for large systems since more contour points are needed for convergence. In all three methods, one thing that is common is the trend of speedups when we double the number of nodes employed which is 1.5× for 4–8 nodes, 1.3× for 8–16 and 1.15× for 16–32 nodes. The decrease in speedup with increase in nodes is mainly due to process synchronization and limitations in inter-node communications. Atomistic simulation software like NEMO-3D and OMEN that uses a similar TB parameterization, matrix distribution scheme and a Lanczos method have reported close to ideal scaling on Ranger and Cray XT4 supercomputers using only MPI technology to parallelize [34]. We also can expect a similar scaling on these machines, as the speedups strongly depend on the underlying machine and network architecture.

Memory analysis shows that there is no significant difference in memory consumption when the Hamiltonian is split on 4 nodes or 32 nodes. This is because the size of the

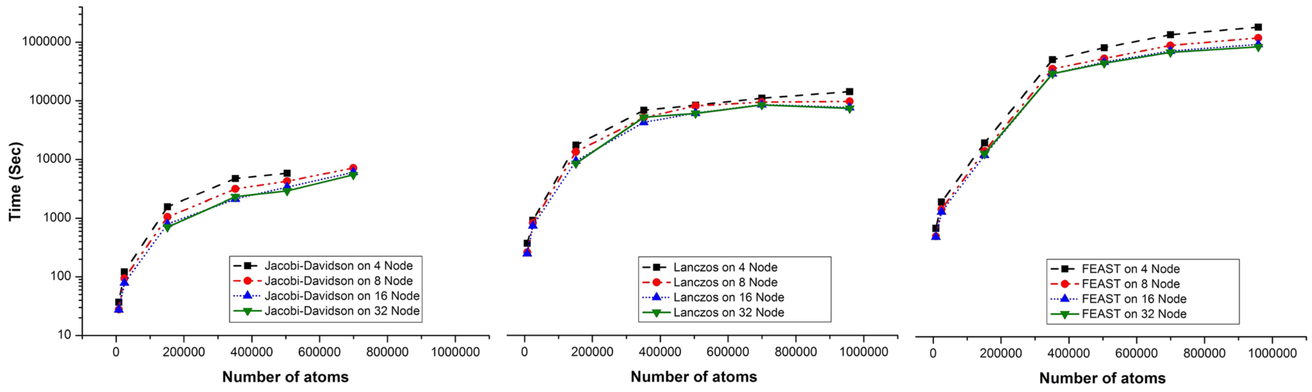

Fig. 6 Time comparison between Jacobi–Davidson, Lanczos and FEAST method on 4, 8, 16 and 32 nodes of the HPC cluster for the calculation of 8 energy eigenstates

subspace and temporary vectors overplay the importance of the TB Hamiltonian matrix which has been highly memory optimized using the single precision storage and splitting technique. Out of the three methods considered, Lanczos is most memory efficient given that we do not store any subspace because of our choice of more flops over bytes [20]. It is followed by the FEAST method using CGS as linear solver, which requires 3.2× times more memory than Lanczos mainly because we need a search space bigger than the number of eigenpairs in the given interval. The Jacobi–Davidson method is found to be the most memory expensive given its requirement to save an adequate subspace and find a solution to the complex algebra correction equation. Jacobi–Davidson requires 5× times more memory than Lanczos and hence, we can fit only up to 699,399 atom quantum dots on the test hardware.

3.3 Performance comparison between GPU and HPC cluster

To examine the advantage of GPUs over an expensive HPC cluster for tight-binding calculations, we will compare the performance of 1 and 4 Tesla Kepler GPUs with 256 CPU cores and also inspect the gain of multi-GPUs over a single GPU. So comparing the performance of the different method against different hardware, we can infer that for Lanczos and FEAST method we get a 3.0× and 2.6× scaling in speedup when we go from 1 GPU to 4 GPUs for large quantum dots. Whereas, in the case of Jacobi–Davidson the speedup is limited to a factor of 1.6× demonstrating that the transfer of the subspace from the host to the device and vice versa is the limiting factor as already stated.

When we equate the performance of 256 CPU cores on the HPC cluster with a single Tesla Kepler GPU, the Jacobi–Davidson method on the HPC cluster is found to outperform the GPU by a factor of 1.2×. On the contrary, the GPU implementation of Lanczos and FEAST methods on 1 GPU beats the performance of 256 CPU cores by a factor of 5.8× and 4.1×, respectively. Comparing the multi-GPU implementation on 4 GPUs against 256 CPU cores of the HPC cluster for Jacobi–Davidson, Lanczos and FEAST method the multi-GPU system outperform the HPC cluster by a factor of 1.5×, 13.7× and 10.8×, respectively.

4 Conclusion

A comprehensive study of Jacobi–Davidson, Lanczos and FEAST methods for energy eigenstate calculation in nanostructures have been conducted. By creating, testing and profiling, a GPU based performance enhanced implementation of the methods we have examined their feasibility and advantage as an eigensolvers specifically for tight-binding calculations. Our investigation has shown that Jacobi–Davidson is the most robust method in terms of convergence and is fast in terms of execution time. However, it has high memory consumption and is therefore less suited for calculating the energy eigenstates of large nanostructures. This shortcoming can be overcome by moving the subspace vectors to the host memory as shown thus enabling us to calculate the energy states of larger systems. Nevertheless, this type of GPU implementation of the Jacobi–Davidson does not scale well as compared to Lanczos and FEAST.

Lanczos on the contrary is most memory efficient method, but the poor convergence for higher energy eigenstates in large nanostructures is a primary bottleneck making it not the first method of choice. However, on a multi-GPU system, it shows a superior scaling trend. FEAST method performs the worst since we do not use any preconditioner matrix while solving the block linear system because the construction of the typical preconditioner based on incomplete factorization is expensive in terms of both memory and time and is not ideal for a GPU based implementation.

To conclude Jacobi–Davidson is the best method given its good convergence even without a preconditoner matrix and should be considered as the method of choice on computing systems where memory is not a constraint. On GPUs, it can be employed to calculate the energy eigenstates of few hundred thousand atom nanostructures. Lanczos, on the other hand, is the method of choice when memory usage is the limiting factor, even though Lanczos is slow in convergence it can be easily scaled using a multi-GPU implementation to perform in par with Jacobi–Davidson.

Acknowledgments The authors acknowledge the SOPHIA European Research Infrastructure for access to the JUROPA HPC cluster at the Jülich Supercomputing center.

References

1. Di Carlo, A.: Tight-binding methods for transport and optical properties in realistic nanostructures. Physica B **314**(2002), 211–219 (2002)
2. Pecchia, A., Di Carlo, Aldo: Atomistic theory of transport in organic and inorganic nanostructures. Rep. Prog. Phys. **67**(8), 1497–1561 (2004)
3. Di Carlo, A.: Microscopic theory of nanostructured semiconductor devices: beyond the envelope-function approximation. Semicond. Sci. Technol. **18**, R1–R31 (2003)
4. Penazzi, G., Pecchia, A., Sacconi, F., Di Carlo, A.: Calculation of optical properties of a quantum dot embedded in a GaN/AlGaN nanocolumn. Superlattices Microstruct. **47**(1), 123–128 (2010)
5. Colombot, L., Sawyer, W., Marict, D.: A parallel implementation of tight-binding molecular dynamics based on reordering of atoms and the Lanczos Eigen-Solver. MRS Proc. **408**, 107 (1995). doi:10.1557/PROC-408-107
6. Bergamaschi, L., Pini, G., Sartoretto, F.: Computational experience with sequential and parallel, preconditioned Jacobi-Davidson for large, sparse symmetric matrices. J. Comput. Phys. **188**(1), 318–331 (2003). doi:10.1016/S0021-9991(03)00190-6
7. Camara, M., Mauger, A., Devos, I.: Electronic structure of the layer compounds GaSe and InSe in a tight-binding approach. Phys. Rev. B **65**, 125206 (2002)
8. Levin, A.R., Zhang, D., Polizzi, E.: FEAST fundamental framework for electronic structure calculations: reformulation and solution of the muffin-tin problem. Comput. Phys. Commun. **183**(11), 2370–2375 (2012). doi:10.1016/j.cpc.2012.06.004
9. Laux, S.E.: Solving complex band structure problems with the FEAST eigenvalue algorithm. Phys. Rev. B **86**, 075103 (2012)
10. Lanczos, C.: An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. J. Res. Nat'l Bur. Std. **45**, 225–282 (1950)
11. Stewart, G.W.: Eigensystems. In: Matrix Algorithms. Chap. 5, vol. II, pp. 306–367, SIAM, Philadelphia (2001). ISBN 0-470-21820-7
12. Cullum, J.K., Willoughby, R.A.: Lanczos Algorithms for Large Symmetric Eigenvalue Computations, vol. 1. SIAM, Philadelphia (2002)
13. Parlett, B.N., Scott, D.S.: The Lanczos algorithm with selective orthogonalization. Math. Comput. **33**(145), 217–238 (1979)
14. San-Cheng, C.: Lanczos algorithm with selective reorthogonalization for eigenvalue extraction in structural dynamic and stability analysis. Comput. Struct. **23**(2), 121–128 (1986). doi:10.1016/0045-7949(86)90206-3
15. Sleijpen, G.L.G., van der Vorst, H.A.: A Jacobi-Davidson iteration method for linear eigenvalue problems. SIAM J. Matrix Anal. Appl. **17**, 401–425 (1996)
16. Hochstenbach, M.E., Notay, Y.: The Jacobi-Davidson method. GAMM Mitt. **29**(2), 368–382 (2006). ISSN:0936–7195
17. Arbenz, P., Hochstenbach, M.E.: A Jacobi-Davidson method for solving complex symmetric eigenvalue problems SIAM. J. Sci. Comput. **25**, 1655–1673 (2004). doi:10.1137/S1064827502410992
18. Fletcher, R.: Conjugate gradient methods for indefinite systems. In: Lecture Notes in Mathematics, 2nd revised edn, pp. 73–89. Springer, Berlin (1976)
19. Polizzi, E.: Density-matrix-based algorithms for solving eigenvalue problems. Phys. Rev. B. **79**, 115112 (2009)
20. Rodrigues, W., Pecchia, M., Lopez, M.A., der Maur, A., Di Carlo, A.: Accelerating atomistic calculations of quantum energy eigenstates on graphic cards. Comput. Phys. Commun. **185**, 2510–2518 (2014). doi:10.1016/j.cpc.2014.05.028
21. Jancu, J.M., Bassani, F., Sala, F.D., Scholz, R.: Transferable tight-binding parametrization for the group-III nitrides. Appl. Phys. Lett. **81**, 4838 (2002)
22. Kesheng, W., Simon, H.: Thick-restart Lanczos method for large symmetric eigenvalue problems. SIAM J. Matrix Anal. Appl. **22**(2), 602–616 (2000)
23. Grosso, G., Martinelli, L., Pastori Parravicini, G.: Lanczos-type algorithm for excited states of very-large-scale quantum systems. Phys. Rev. B **51**, 13033–13038 (1995)
24. Pescetelli, S., Di Carlo, A., Lugli, P.: Conduction band mixing in T- and V-shaped quantum wires. Phys. Rev. B **56**, 1668 (1997)
25. Barret, R., Berry, M., Chan, T.F., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., van der Vorst, H.: Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods. SIAM, Philadelphia (1994)
26. Sleijpen, G.L.G., Booten, J.G.L., Fokkema, D.R., van der Vorst, H.A.: Jacobi-Davidson type methods for generalized eigenproblems and polynomial eigenproblems. BIT **36**, 595–633 (1996)
27. Hochstenbach, M.E., Sleijpen, G.L.G.: Harmonic and refined Rayleigh-Ritz for the polynomial eigenvalue problem. Numer. Linear Algebra Appl. **15**(1), 35–54 (2008)
28. Saad, Y.: Iterative Methods for Sparse Linear Systems, 2nd edn. Society for Industrial and Applied Mathematics, Philadelphia (2003). ISBN 978-0-89871-534-7
29. Saad, Y., Schultz, M.H.: GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. SIAM J. Sci. Stat. Comput. **7**, 856–869 (1986). doi:10.1137/0907058
30. Galgon, M., Kramer, L., Lang, B.: The FEAST algorithm for large eigenvalue problems. PAMM. Proc. Appl. Math. Mech. **11**, 747–748 (2011). doi:10.1002/pamm.201110363
31. Polizzi, E.: A high-performance numerical library for solving eigenvalue problems: FEAST solver user's guide (2012). arxiv.org/abs/1203.4031
32. Fokkema, D.R., Sleijpen, G.L.G., Van der Vorst, H.A.: Generalized conjugate gradient squared. J. Comput. Appl. Math. **71**, 125–146 (1996)
33. Benzi, M.: Preconditioning techniques for large linear systems: a survey. J. Comput. Phys. **182**, 418–477 (2002)
34. Bae, H., Clark, S., Haley, B., Ryu, H., Klimeck, G., Lee, S., Luisier, M., Saied, F.: A nano-electronics simulator for petascale computing: from NEMO to OMEN, TeraGrid 2008, Jun 9–13, Las Vegas (2008)