Parallel Computing 49 (2015) 153–163

Contents lists available at ScienceDirect

Parallel Computing

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

On the parallel iterative solution of linear systems arising in the FEAST algorithm for computing inner eigenvalues☆

Martin Galgon[a,*], Lukas Krämer[a], Jonas Thies[b], Achim Basermann[b], Bruno Lang[a]

[a] Bergische Universität Wuppertal, Fachbereich C – Mathematik und Naturwissenschaften, 42097 Wuppertal, Germany
[b] German Aerospace Center (DLR), Simulation and Software Technology, Linder Höhe, 51147 Cologne, Germany

ARTICLE INFO

Article history:
Available online 25 June 2015

Keywords:
Parallel inner eigenvalue computation
FEAST
Sparse linear systems
CARP-CG
Multi-coloring
Graphene modeling,

ABSTRACT

Methods for the solution of sparse eigenvalue problems that are based on spectral projectors and contour integration have recently attracted more and more attention. Such methods require the solution of many shifted sparse linear systems of full size. In most of the literature concerning these eigenvalue solvers, only few words are said on the solution of the linear systems, but they turn out to be very hard to solve by iterative linear solvers in practice. In this work we identify a row projection method for the solution of the inner linear systems encountered in the FEAST algorithm and introduce a novel hybrid parallel and fully iterative implementation of the eigenvalue solver. Our approach ultimately aims at achieving extreme parallelism by exploiting the algorithm's potential on several levels. We present numerical examples where graphene modeling is one of the target applications. In this application, several hundred or even thousands of eigenvalues from the interior of the spectrum are required, which is a big challenge for state-of-the-art numerical methods.

© 2015 Elsevier B.V. All rights reserved.

1. Introduction

The solution of eigenvalue problems

$$AX = BX\Lambda \tag{1}$$

with Hermitian matrices $A, B \in \mathbb{C}^{N\times N}$ and positive definite B is a common task in numerical linear algebra. It arises in many application areas such as electronic structure calculations [1] or modeling of graphene nanoribbons [2], see also below. Typically, only a subset of the eigensystem is required, i. e., $m < N$ eigenvalues with corresponding eigenvectors have to be computed. Several methods are available to solve this kind of problem, for instance, of Lanczos [3] or Jacobi–Davidson type [4].

A new class of algorithms for the solution of large, sparse eigenproblems is part of current research. It is based on contour integration of the so called resolvent function $z \mapsto (zB - A)^{-1}B$ of the matrix pair (A, B). Techniques of this class for solving certain eigenvalue problems have been pioneered by Sakurai and co-workers [5–7]. A related approach is the so called FEAST algorithm, first introduced in 2009 [8]. Since then, efforts in analyzing the convergence of the method [9–11] have been made. In addition, a lot of work has been invested to improve the performance of FEAST and related methods, e.g., [10,12]. These methods

☆ This work was supported by the German Research Foundation (DFG) through the Priority Programme 1648 "Software for Exascale Computing" (SPPEXA) under project ESSEX.

* Corresponding author. Tel.: +49 2024392913.
E-mail addresses: galgon@math.uni-wuppertal.de (M. Galgon), lkraemer@math.uni-wuppertal.de (L. Krämer), jonas.thies@dlr.de (J. Thies), achim.basermann@dlr.de (A. Basermann), lang@math.uni-wuppertal.de (B. Lang).

http://dx.doi.org/10.1016/j.parco.2015.06.005
0167-8191/© 2015 Elsevier B.V. All rights reserved.

promise high potential for parallelism by subdividing tasks on multiple levels. Coarse grained parallelism can already be achieved by subdividing the region in which eigenvalues are sought [8,9]. Furthermore, they are able to compute hundreds or thousands of eigenpairs at once, where typical Krylov or Jacobi–Davidson solvers fail in this respect because they require orthogonalization of any new basis vector with respect to the previously converged ones and the current basis. A fixed search space size keeps the amount of required work and memory predictable.

The FEAST method essentially consists of two main components: the (numerical) integration of the resolvent function along a contour in the complex plane, coupled with a Rayleigh–Ritz procedure. The numerical integration step involves the solution (for V) of several size-N linear systems of the form

$$(z\mathsf{B} - \mathsf{A})\mathsf{V} = \mathsf{BY} \tag{2}$$

for several complex shifts z and an $N \times m$ matrix Y. The solution of very large and sparse linear systems is typically performed iteratively, e. g., with Krylov-based methods [13]. For linear systems (2) as found in the FEAST algorithm, standard methods such as GMRES [14] and other Krylov-based solvers show poor performance [9], requiring hundreds of iterations even for small system sizes. In the original publication on FEAST [8, Example III], GMRES was used to solve the inner linear systems to modest accuracy. In many publications on FEAST and similar methods, direct methods are used for the solution of the inner linear systems, or no comment is made on which linear solver was used [6,11,12,15].

The advantage of iterative methods, apart from memory efficiency, is that the accuracy from the solution of the linear systems translates to the accuracy of the eigenpairs obtained by FEAST [9]. Therefore, the goal of this paper is to identify an iterative method for solving the linear systems in FEAST and to present a numerical study that shows the effectiveness, but also the limits, of such an approach. We demonstrate that a parallel row-projection method gives promising results in terms of robustness and scalability. Our implementation is a combination of a scheme called CARP-CG [16] with a multi-coloring technique for better memory efficiency on hybrid shared/distributed memory machines. A limitation that we find is that the number of required iterations for the linear solver to converge increases with the proximity of the integration points to the spectrum, but this issue is left to future work. As a benchmark application we will use eigenvalue problems from graphene modeling and a 3D model problem for Anderson localization.

Graphene is a manifestation of atomic carbon, and eigenvalue problems arise from the modeling of graphene tubes and ribbons in the tight-binding approach [17]. In order to better understand its properties, simulations involving large sparse matrices are necessary, and a comparatively large number of inner eigenpairs is required for instance to compute an electric current through a graphene sample. This application was previously discussed in the context of the ESSEX project [2,18]. In our experiments, graphene samples of $W \times L$ atoms are described mathematically, resulting in Hermitian matrices of size $WL \times WL$. The main diagonal of these matrices contains random entries from an interval $[-\gamma, \gamma]$, $\gamma > 0$, the spectrum then is contained in $[-3-\gamma, 3+\gamma]$. In [18], the distribution of eigenvalues for a typical problem from graphene modeling can be found. The eigenpairs of interest are those with eigenvalue close to the center of the spectrum [2,18]. The basic configuration of graphene is a two-dimensional hexagonal lattice of carbon atoms, and we consider interactions with nearest and next-nearest neighbors. In practice however, three-dimensional models, e. g., of samples consisting of multiple layers of graphene, pose a greater challenge to the linear solver. For this reason, we also briefly study a three-dimensional model problem for the Anderson localization, which was also studied in [19].

The paper is structured as follows. In Section 2, the basic FEAST method and our parallel implementation are described. In Section 3 we describe the properties of the arising linear systems in more detail and review the so called CGMN algorithm. Section 4 covers the parallel implementation of CGMN for clusters of shared memory nodes, and leads to the novel multi-colored CARP-CG algorithm. Section 5 contains numerical experiments to firstly investigate the robustness and performance of the scheme in the context of FEAST and secondly demonstrate its scalability. Section 6 summarizes the paper and gives an outlook on future research.

2. The FEAST algorithm

2.1. Short description

The basic FEAST algorithm [8] can be formulated as in Algorithm 2.1. For simplicity and due to the target application we will henceforth focus on the standard eigenvalue problem, i. e., $\mathsf{B} = \mathsf{I}$. The symbol $\mathcal{C}$ in step 1 of Algorithm 2.1 denotes a closed contour in the complex plane, surrounding the interval $I_\lambda = \left[\underline{\lambda}, \overline{\lambda}\right]$ (for an example, see Fig. 1). We assume that the interior of $\mathcal{C}$ does not contain any other eigenvalues than those residing in I_λ. Approximating the matrix U in step 1 of Algorithm 2.1 leads to numerical integration of the function $z \mapsto (z\mathsf{I} - \mathsf{A})^{-1}\mathsf{Y}$, which can be formulated as

$$\mathsf{U} \approx \widetilde{\mathsf{U}} = \frac{1}{2\pi \mathbf{i}} \sum_{j=1}^{p} \omega_j \varphi'(t_j) \left(\varphi(t_j)\mathsf{I} - \mathsf{A}\right)^{-1} \mathsf{Y}. \tag{3}$$

Here, $\varphi : [0, 2\pi] \to \mathbb{C}$ is a parametrization of $\mathcal{C}$, the numbers ω_j, $j = 1, \ldots, p$, are integration weights and $t_j \in [0, 2\pi]$, $j = 1, \ldots, p$, are integration points.

Steps 2–4 of Algorithm 2.1 are a plain Rayleigh–Ritz process with the subspace spanned by $\widetilde{\mathsf{U}}$ [9]. In the computation of (3) the solution of p block linear systems

$$(z_j\mathsf{I} - \mathsf{A}) \cdot \mathsf{V} = \mathsf{Y} \tag{4}$$

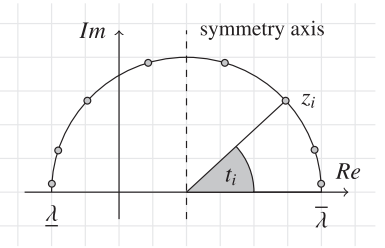

Fig. 1. Generic contour example over an interval $I_\lambda = \left[\underline{\lambda}, \overline{\lambda}\right]$. The eight shifts are not necessarily symmetric w.r.t. the imaginary axis. Real eigenvalue problems allow for exploitation of complex conjugated symmetry though; only one half (here the upper half in the imaginary positive half-plane) of the closed contour (here a circle) needs to be considered.

Algorithm 2.1 Skeleton of the FEAST algorithm (as in [9,10]).

Input: An interval $I_\lambda := \left[\underline{\lambda}, \overline{\lambda}\right]$ and an estimate $\widetilde{m}$ of the number of eigenvalues in I_λ.
Output: $\hat{m} \leq \widetilde{m}$ eigenpairs with eigenvalue in I_λ.

1: Choose $\mathsf{Y} \in \mathbb{C}^{N \times \widetilde{m}}$ of rank $\widetilde{m}$ and compute an approximation $\widetilde{\mathsf{U}}$ to

$$\mathsf{U} := \frac{1}{2\pi \mathbf{i}} \int_{\mathcal{C}} (z\mathsf{I} - \mathsf{A})^{-1} \mathsf{Y} dz.$$

2: Form the Rayleigh quotients $\mathsf{A}_{\widetilde{\mathsf{U}}} := \widetilde{\mathsf{U}}^H \mathsf{A} \widetilde{\mathsf{U}}$, $\mathsf{B}_{\widetilde{\mathsf{U}}} := \widetilde{\mathsf{U}}^H \widetilde{\mathsf{U}}$.
3: Solve the size-$\widetilde{m}$ generalized eigenproblem $\mathsf{A}_{\widetilde{\mathsf{U}}} \widetilde{\mathsf{W}} = \mathsf{B}_{\widetilde{\mathsf{U}}} \widetilde{\mathsf{W}} \widetilde{\Lambda}$.
4: Compute the approximate Ritz pairs $\left(\widetilde{\Lambda}, \widetilde{\mathsf{X}} := \widetilde{\mathsf{U}} \cdot \widetilde{\mathsf{W}}\right)$.
5: If convergence is not reached then go to Step 1 with $\mathsf{Y} := \widetilde{\mathsf{X}}$.

with complex numbers $z_j = \varphi(t_j)$ is necessary. This is in general by far the most time consuming part of the overall computation, hence particular attention has to be paid to this part of the algorithm. Problems in practice often yield very large and sparse matrices A. In this case, a method for the solution of (4) that is based on factorization of the system matrix $z_j\mathsf{I} - \mathsf{A}$ is not applicable; focus has to be shifted to iterative solvers. A variety of different iterative methods for the solution of such systems is known, for instance those based on Krylov subspaces. For an overview, see, e. g., [13]. We will discuss this topic in some detail in Section 3.1.

2.2. Parallel implementation

The FEAST algorithm exhibits potential for parallelism on multiple levels. Being a method to compute eigenpairs inside a given interval, the most obvious approach consists of subdividing the initial search interval I_λ into smaller sub-intervals that may be tackled independently.

While this seems to provide immediate parallelism without any synchronization between separate sub-intervals, eigenvectors of individually computed eigenpairs show poor orthogonality when their eigenvalues are in close proximity [20]. For an illustration of this effect, see [9]. Therefore, and due to the requirements of the target application—not much more than a few hundred eigenpairs around a certain point inside the spectrum are required—the multiple-intervals approach has not been pursued for the experiments in this paper.

Further opportunities for parallelization result from the size-N complex shifted linear systems arising during numerical integration of the resolvent. Using, e. g., a Gauß–Legendre scheme [21] as approximation, the solution of a set of p block linear systems $(z_j\mathsf{I} - \mathsf{A})\mathsf{V}_j = \mathsf{Y}$, $j = 1, \ldots, p$, is required, each with a block of multiple right-hand sides Y. Three levels of parallelization may be exploited here: I) The p systems may be assigned to p processor groups, solving the systems independently. II) The right-hand sides of each block system may be distributed over the corresponding processor group, again solving the same system with different right-hand sides independently.

At a third level, each BLAS-like operation performed during the linear solves as well as other operations from the FEAST algorithm may be performed in parallel, with matrix and vectors distributed over subgroups of processors.

In our current FEAST implementation, the third level is handled by the underlying PHIST[1] library developed in the context of the ESSEX project [18]. It provides an interface layer to hybrid-parallel building block functionality implemented in libraries such as GHOST[2] [18], as well as implementations of some iterative solvers for linear systems and extremal eigenvalue problems.

[1] https://bitbucket.org/essex/phist .
[2] https://bitbucket.org/essex/ghost .

As for the parallelization of the linear systems, right-hand sides are distributed over processor groups of variable size, in a manager/worker-type hierarchy (level II parallelism). A group of processors takes the position of the manager, performing the FEAST algorithm (Algorithm 2.1) while dealing out tasks of solving the linear systems (4) as required. All remaining groups take a worker role, each group working on their respective set of vectors once requested. Operations within these groups are parallelized as described in the preceding paragraph.

As the number of available nodes on the testing architectures (see Section 5.2) was moderate, we chose to process the differently shifted systems sequentially (i. e., no level I parallelism is used), thus bypassing a fan-in summation of the results and issues with load balancing because the convergence rate of iterative solvers depends very much on the shift.

3. An iterative solver for the shifted linear systems

While the FEAST algorithm comes with a promise of excellent parallelization potential and memory-efficient computation of many inner eigenvalues, its practical applicability is limited so far by the linear systems (2) that have to be solved by (sparse or banded) direct methods involving some triangular factorization of a sparse matrix for each shift. The factorization step is very expensive and memory consumption is high. For matrices from 3D elliptic PDEs it typically has a computational complexity of $\mathcal{O}(N^2)$, where N is the dimension of the matrix. For 2D problems a more favorable $\mathcal{O}(N^{2/3})$ [22] can be achieved. Nonetheless, we do not want to limit our eigensolver to applications such as graphene, which happens to possess this structure. The iterative method we propose achieves linear complexity in our experiments when the shift is fixed, and $\mathcal{O}(N^{2/3})$ for graphene matrices in the context of FEAST as the interval size has to be decreased with approximately $\sqrt{N}$ to keep the number of eigenvalues inside constant. It is also far more memory efficient and scalable than direct sparse solvers, which better suites our aim of exploiting the massive parallelism of today's supercomputers.

Throughout this section, we consider a linear system $\mathsf{M}\mathsf{x} = \mathsf{b}$, $\mathsf{M} \in \mathbb{C}^{N\times N}$, with a single right hand side b. The most popular alternative to direct methods are Krylov subspace methods which approximate the solution x in a low-dimensional Krylov subspace, $\mathcal{K}_k(\mathsf{M}, \mathsf{b}, \mathsf{x}_0) = \operatorname{span}\{\mathsf{x}_0, \mathsf{M}\mathsf{x}_0, \mathsf{M}^2\mathsf{x}_0, \ldots, \mathsf{M}^k\mathsf{x}_0\}$, $k \ll N$. The convergence behavior of such a method (and thus the number of iterations k required to achieve the desired accuracy) depends on the spectral properties of M. For Hermitian matrices, convergence will be reached fast if the spectral condition number $\kappa = \sigma_{\max}/\sigma_{\min}$ (i. e., the ratio of the largest and smallest singular value) is close to one. However, even with a large condition number, convergence speeds may still be quite reasonable if the distribution of the eigenvalues is favorable [23]. The eigenvalues of $z\mathsf{I} - \mathsf{A}$ will typically lie on both sides of the imaginary axis, which prevents methods as GMRES or BiCGStab from converging fast [14]. The system matrix is non-Hermitian and indefinite even if A is (real) symmetric and positive definite, hence the conjugate gradient (CG) method [13] for Hermitian positive definite (hpd) matrices is not directly applicable, and neither is MINRES, which requires a Hermitian indefinite matrix (and an hpd preconditioner). A method that can exploit the symmetry of the matrix (even though it is indefinite and non-Hermitian) is the simplified QMR method. One could also use the "double shift" strategy and solve the hpd system $(\bar{z}\mathsf{I} - \mathsf{A})(z\mathsf{I} - \mathsf{A})\mathsf{x} = (\bar{z}\mathsf{I} - \mathsf{A})\mathsf{b}$ (the normal equations) instead. The system matrix then is hpd as long as z is not an eigenvalue of A. This strategy squares the condition number of the original problem, but may still give quite good results in practice. The CGMN method used in this paper can be interpreted as a preconditioned "double shift" method.

3.1. Properties of complex-shifted linear systems

For the sake of this discussion, we consider the shifted matrix $\mathsf{A}_z := z\mathsf{I} - \mathsf{A}$, which is an instance of the system matrix in (2). All considerations made in the following can easily be transferred to the generalized eigenvalue problem with $\mathsf{B} \neq \mathsf{I}$ Hermitian and positive definite via a simple transformation [20]. If $z \in \mathbb{C}$ and A is real symmetric, then $\mathsf{A}_z = (z\mathsf{I} - \mathsf{A}) = \mathsf{A}_z{}^T \in \mathbb{C}^{N\times N}$ is complex symmetric and indefinite in general. If $\mathsf{A} \in \mathbb{C}^{N\times N}$ is Hermitian, then A_z is neither Hermitian nor complex symmetric. The eigenvalues of A_z are related to those of A by $\lambda_z{}^{(j)} = (z - \lambda^{(j)})$, and the eigenvectors are those of A. Hence, A_z is a normal matrix with the same eigenvalue distribution as A, but the condition number may be significantly worse if z is close to an eigenvalue of A. Assuming that we want to compute only a small section in the interior of the spectrum of A, and that the eigenvalue distribution is fairly dense in this region, we can roughly assume for the condition number of the system matrix A_z to be proportional to the inverse of the imaginary part of the shift. The shifts z are dictated by the FEAST algorithm. If we are only interested in a small portion of the spectrum, the interval I_λ and hence the contour $\mathcal{C}$ from step 1 of Algorithm 2.1 is small. This results in a small imaginary part of some of the shifts z and hence in a large condition number κ_z. In future work, we plan to investigate techniques that can bring the shifts further away from the real axis without sacrificing accuracy.

Despite the close relation between A and A_z, it is a challenging task to develop a preconditioner that can improve the convergence behavior of an iterative method for A_z. Consider, for instance, the multigrid method [24] as a preconditioner. What makes this approach successful is that matrices $\mathsf{A}^{(ell)}$, stemming from the discretization of elliptic partial differential equations (PDEs), typically have few distinct smooth eigenmodes at the lower end of the spectrum, which can be 'removed' by a coarse grid correction. By contrast, the matrix $\mathsf{A}_z{}^{(ell)}$, with z in the interior of the spectrum $\sigma\left(\mathsf{A}^{(ell)}\right)$ of $\mathsf{A}^{(ell)}$, typically has non-smooth eigenvectors associated with its smallest eigenvalues. A typical application yielding such matrices is the Helmholtz equation, and attempts have been made to adjust the standard multigrid method for this problem [25]. It is not clear whether these techniques can be applied to interior eigenvalue problems in general or graphene simulation in particular. Similarly, some authors have adapted FETI-type methods [26,27]. A commonly used technique for the Helmholtz equation is multigrid on the shifted Laplacian (typically with a complex shift "with opposite sign", see e. g. [28]).

Another class of preconditioners is based on the idea of incomplete LU (or LDL^T) factorizations, which are popular because of their algebraic nature and ease of use. However, they are difficult to parallelize and lack robustness for indefinite matrices unless great care is taken. For an overview and references, see [29]. The only robust algorithm we are aware of is ILUPACK [30], which was applied to the inner eigenvalue problem of the Anderson model in [19]. For our purposes ILUPACK does not provide sufficient potential for parallelism and the factorization consumes large amounts of memory and compute time. Furthermore, ILUPACK has difficulties factoring matrices with a very weak diagonal, i. e., $|\mathsf{M}_{ii}| \ll \sum_{j\neq i} |\mathsf{M}_{ij}|$ (as is the case in our graphene application) because it requires at least some diagonally dominant rows on each level in order to effectively reduce the size of the following Schur complement. The difficulties with weak diagonals seem to be inherent to ILU-type preconditioners.

3.2. The CGMN algorithm

An alternative to the presently popular preconditioning techniques mentioned above is the use of row projection (RP) methods [31]. One of the most well-known algorithms of this class is the Kaczmarz [32] method, denoted by KACZ in the following. Nowadays KACZ is rarely used as stand-alone solver for linear systems because it is typically inferior to Krylov methods with preconditioning for systems stemming from the discretization of elliptic PDEs. However, derivatives of RP methods may be attractive for challenging linear systems because they are purely algebraic and extremely robust. In 1979, Björck and Elfving [33] published several algorithms based on conjugate gradient acceleration of row projection methods. We are particularly interested in their CGMN method, see also [34]. It is based on the following ideas:

- The KACZ sweep (see Algorithm 3.1 below) is equivalent to SOR on the minimum norm problem (MNP),
$$\mathsf{M}\mathsf{M}^T\mathsf{y} = \mathsf{b}, \mathsf{x} = \mathsf{M}^T\mathsf{y}. \tag{5}$$
- Letting the forward KACZ sweep be followed by a backward sweep yields SSOR on the MNP,
$$\mathsf{x}^{(k+1)} = \mathsf{Q}_{SSOR}\mathsf{x}^{(k)} + \mathsf{R}_{SSOR}\,\mathsf{b}, \text{ with}$$
$$\mathsf{Q}_{SSOR} = Q_1 Q_2 \dots Q_n Q_{n-1} \dots Q_1, Q_i = \mathsf{I} - \frac{\omega}{||m_{i,:}^H||^2} m_{i,:}^H m_{i,:}, \text{ and}$$
$$\mathsf{R}_{SSOR} = \omega\delta_{ij} \sum_k m_{k,i}.$$
The Q_i are projections onto the ith row $m_{i,:}$ of M. For more details, see [33, Sections 4 and 5].
- CG for the related linear system $(\mathsf{I} - \mathsf{Q}_{SSOR})\mathsf{x} = \mathsf{R}_{SSOR}\,\mathsf{b}$ converges even though the system matrix is only symmetric positive semi-definite.

CGMN is thus an efficient implementation of the conjugate gradient method for (5) (for an $N \times K$, $K \leq N$ matrix M), preconditioned by symmetric SOR. Alternatively, it can be seen as the Kaczmarz method accelerated by CG. This solver has several features that make it attractive for our highly indefinite linear systems:

- **Robustness.** The method was developed for singular and non-square matrices.
- **Short recurrence**. The CG algorithm only requires scalar products and vector updates so that the main parallelization challenge is the Kaczmarz kernel (see Section 4).
- **Scaling.** Squaring the system matrix increases the relatively small diagonal elements occurring in our target application, graphene, while the inherent row scaling of the Kaczmarz method alleviates the squared condition number [35].

For the readers convenience, the complete CGMN algorithm is shown in Algorithm 3.2. The operator to which the CG method is applied is $(\mathsf{I} - \text{DKSWP}(\mathsf{M}, \mathsf{b}, \mathsf{x}, \omega))$, where $\text{DKSWP}(\mathsf{M}, \mathsf{b}, \mathsf{x}, \omega)$ denotes a "double Kaczmarz sweep", that is, a forward KACZ sweep (see Algorithm 3.1) followed by a backward sweep (with the i-loop reversed), in order to make the operator symmetric. In Algorithm 3.1, ω is a relaxation parameter, which we set to 1 in all our experiments (no significant speed-up was achieved by using $\omega \neq 1$ for our test cases). In contrast to the standard (sparse) matrix-vector operation $\mathsf{y} \leftarrow \mathsf{M}\mathsf{x}$, this operation updates x in-place and writes to x by indirect access (symbolized by j_i here). These properties prevent a straight-forward parallelization of the method.

Algorithm 3.1 $\mathsf{x} \leftarrow$ **KACZ** $(\mathsf{M}, \mathsf{b}, \mathsf{x}, \omega)$: **Forward Kaczmarz sweep.**
$m_{i,:}$ denotes the i-th row of M and j_i denotes the column indices of the nonzero entries in $m_{i,:}$.

1: **for** $i = 1, \dots, N$ **do**
2: $\quad s = (m_{i,j_i} x_{j_i} - b_i)/||m_{i,j_i}^H||_2^2$
3: $\quad x_{j_i} = x_{j_i} - \omega s m_{i,j_i}^H$
4: **end for**

In [36] it is shown that a solver based on CGMN can be very efficient when solving the Helmholtz equation at high frequency, but the method has not been applied to inner eigenvalue computations so far.

Algorithm 3.2 Simple implementation of the CGMN algorithm [33].

Input: A, x_0, z, b, ω, tol, maxIter

Output: A vector x fulfilling $\|(zI - A)x - b\|_2 / \|(zI - A)x_0 - b\|_2 <$ tol, or a warning message if the tolerance was not achieved within maxIter steps of CGMN.

1: $\text{nrm}_r = \text{nrm}_0 = \|(zI - A)x_0 - b\|_2$
2: $x = x_0$
3: $r = \text{DKSWP}(zI - A, b, x, \omega) - x$
4: $p = r$
5: $r2_{new} = r^H r$
6: **for** $k = 1$, maxIter **do**
7: $q = p - \text{DKSWP}(zI - A, 0, p, \omega)$
8: $\alpha = r2_{new}/(p^H q)$
9: $x = x + \alpha p$
10: **if** $\text{mod}(k - 1, 10) == 0$ **then**
11: $\text{nrm}_r = \|(zI - A)x - b\|_2$
12: **if** $\text{nrm}_r <$ tol $\cdot\, \text{nrm}_0$ **then**
13: **break**
14: **end if**
15: **end if**
16: $r = r - \alpha q$
17: $r2_{old} = r2_{new}$
18: $r2_{new} = r^H r$
19: $\beta = r2_{new}/r2_{old}$
20: $p = r + \beta p$
21: **end for**
22: **if** $\text{nrm}_r \geq$ tol $\cdot\, \text{nrm}_0$ **then**
23: **warning** "convergence was not achieved within maxIter steps"
24: **end if**
25: **return**

4. Parallel CGMN

Row projection methods were introduced as sequential algorithms to begin with, but many authors have worked on generalizing them to more parallel schemes. Most methods to date are based on the block projection (BP) algorithm of Elfving [37]. It requires choosing a partitioning of the rows of M into n block rows $M_{i,:}$,

$$M^H = \left(M^H_{1,:}, M^H_{2,:}, \ldots, M^H_{n,:}\right). \tag{6}$$

The solution vector is then successively projected onto the null space of each block row, which (in general) requires the solution of linear systems with matrices $M_{i,:}M^H_{i,:}$. For instances of this algorithm, see, e. g., [38,39]. A special situation is the case where each $M^H_{i,:}$ is an orthogonal matrix. The linear systems then reduce to diagonal scalings. Hence the partitioning should be chosen such that it produces (approximately) independent columns in each block $M^H_{i,:}$. We will further discuss this in Section 4.1.

Gordon and Gordon [16,40] propose a different approach. The system is also partitioned into block rows, but not to generate parallelism in each block. Instead, as is done in domain decomposition methods [26], the aim of the partitioning is to minimize the "edge cut", i. e., the number of non-zero entries in the off-diagonal blocks. The KACZ sweeps required for CGMN are then performed independently for each block row, which leads to inconsistent entries in the x-vector on the interface between two or more partitions. In [40] it is shown that averaging these "duplicate" vector elements between the partitions is equivalent to KACZ in some superspace of $\mathbb{R}^{N\times N}$ and therefore recovers the convergence properties of the sequential algorithm. In [16] the CGMN-like method based on this idea is presented, which is called CARP-CG.

4.1. Multi-coloring

Let G_M be the undirected graph which has a node for every row of M, and an edge between nodes i and j if $m_{i,j} \neq 0$. A distance-1 coloring is a decomposition of the nodes of G_M into n sets S_k such that any nodes $i \neq j$ in the same set are not connected by an edge. A distance-2 coloring of G_M is a distance-1 coloring of G_{M^2}, which means that any path between two nodes i, j in the same set consists of at least three edges and two nodes not in this set. For matrices with a symmetric sparsity pattern, the problem of finding an optimal row partitioning such that $M_{i,:}M^H_{i,:}$ is a diagonal matrix for each block i is equivalent to finding a distance-2 coloring of the adjacency graph G_M of M.

If a distance-2 coloring of M is available, the KACZ algorithm (Algorithm 3.1) can be parallelized by treating the rows that correspond to the same color in parallel. No race condition occurs in line 3 of Algorithm 3.1. The number of synchronization

points is equal to the number of colors, because the colors must be treated sequentially. The problem of finding an optimal coloring of a graph with the least possible number of colors is NP-complete, but algorithms and software exist that compute a reasonably good distance-2 coloring for a matrix with symmetric sparsity pattern in shared [41,42] and distributed [43] memory environments.

4.2. Hybrid-parallel approach

The approach based on graph coloring discussed in Section 4.1 has two problems. First, finding a "good" coloring in the sense that the number of colors is small may not be feasible for extremely large matrices on massively parallel and heterogeneous machines. Second, depending on the problem at hand, the number of required colors may be quite large and therefore applying the KACZ sweeps involves many global synchronization points. For instance, a graphene matrix with only nearest-neighbor interactions can be (distance-2) colored using not more than four colors, but as soon as interactions with the next-to-nearest neighboring atoms are taken into account during modeling, about 20 colors are required. For 3D problems this number increases even further. The 7-point stencil used in the Anderson model problem requires 15 colors, even with only nearest neighbor interactions taken into account. We therefore chose to parallelize by coloring only inside each (shared memory) node of the compute cluster. Between the nodes, the averaging method of CARP is used.

As an alternative, CARP can be used even within the single nodes. CARP requires synchronization only between the KACZ sweeps. However, its shared memory implementation requires the duplication of certain nodes in the interior of the MPI partition. Such an approach was pursued in [44] for GPUs, but with moderate success. We will show in Section 5 that the increase in "averaged" vector entries leads to a significant memory overhead in weak and strong scaling experiments with CARP, so that fewer nodes can be used compared to the hybrid approach. We expect that our approach will prove feasible also when parts of the computation are performed on the GPU.

5. Numerical experiments

We consider the following model problem. A square graphene sheet consisting of $W \times W$ carbon atoms with periodic boundary conditions results in a matrix of size $N = W^2$. The disorder parameter is set to $\gamma = 0.2$, and interactions with the nearest and next-to-nearest neighbors are taken into account. We will present two experiments. First, we assess the number of iterations required by CARP-CG when seeking a given number of eigenpairs for an increasing problem size. In the second set of experiments, we assess the performance and scalability of variants of the CARP-CG method, noting that its time consumption is proportional to the number of iterations. We also show the variation in iteration numbers between the CARP-CG variants and use the results of the two sets of experiments to extrapolate the performance of FEAST/CARP-CG for larger problems. The implementation of FEAST used throughout the experiments is our own code, as described in Section 2.2.

5.1. Experiment A: convergence behavior

We compute all eigenpairs in an interval around 0, for increasing values of W. In order to keep the number of eigenvalues in the interval approximately constant, the interval size is chosen to be proportional to $1/W$. For all experiments, we use Gauß–Legendre integration with eight integration points on a semi-circle (exploiting complex conjugated symmetry). The stopping criterion for both, the eigenvalue residuals and the inner linear solver, is set to tol $= 10^{-12}$. The CGMN method serves as iterative solver with multi-coloring for node-level parallelism (single node only). The starting vectors are always chosen as $x_0 = 0$ and the convergence criterion is the residual relative w.r.t. the right-hand side, $\|A_z x - b\|_2 / \|b\|_2 <$ tol.

It is obvious that in this setting, the integration points (and thus the shifts z in the linear systems) move closer to the spectrum as W grows. The resulting increase in numbers of CGMN iterations is shown in Table 1. Note that the shifts are distributed symmetrically with respect to the axis bisecting the interval in its center (see Fig. 1), such that shifts 1, 2, 3, 4 give similar iteration counts as shifts 8, 7, 6, 5. The numbers shown are a pessimistic average over all FEAST iterations and right-hand side vectors because the method iterates on blocks of 16 vectors at a time and moves on to the next block only if all systems in a block are converged. The iteration count of CGMN is roughly proportional to the inverse of the interval length, which is consistent with our expectations from Section 3.1. The fairly dense spectrum near the interval boundaries does not allow to effectively reduce the

Table 1
Interval boundaries: $[-\beta, \beta]$, number of eigenvalues found: n_{ev}, the last four columns show the average number of CGMN iterations required for the first four shifts.

W	β	n_{ev}	it_{FEAST}	$\overline{\text{it}}_{\text{MC-CGMN}}$			
				z_1	z_2	z_3	z_4
100	1/4	124	7	699	166	86	61
200	1/8	139	5	1273	333	165	120
400	1/16	132	6	2766	664	324	234

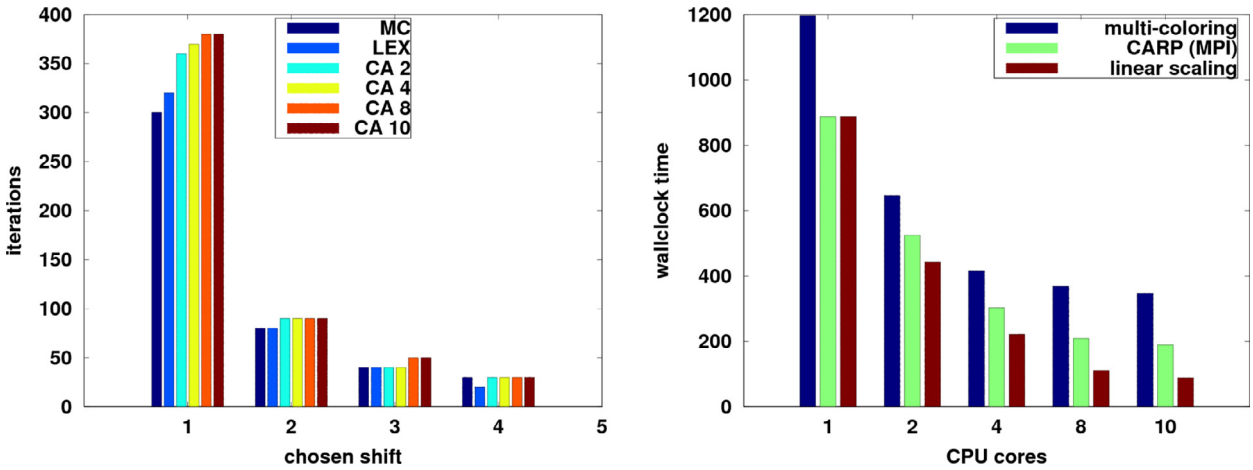

Fig. 2. Multi-coloring vs. CARP inside a 10 core shared memory socket. Left: the various bars marked 'CA-X' show the dependence of the number of iterations on the number of cores when CARP is used; LEX: using lexicographic ordering (serial Kaczmarz); MC: using multi-coloring (parallel Kaczmarz). The right-hand plot shows the run time for solving all the 64 linear systems with the two variants, CARP-CG and hybrid MC-CGMN.

iteration count by preconditioning, so the condition number increases linearly as the imaginary part of the shift moves closer to the spectrum. The outermost integration points (shift z_1 in Table 1) lead to the worst conditioned systems by far, and the number of iterations clearly reflects the circular shape of the integration curve on which the shifts are located.

Our results show that the proposed row projection method is able to converge within a reasonable number of iterations, whereas the standard techniques mentioned in Section 3.1 (e. g., GMRES(m) or BiCGStab with or without ILU preconditioning) fail in this regard. For instance, GMRES takes many iterations and requires an expensive preconditioner, see, e. g., [45].

5.2. Experiments B: weak and strong scaling of CARP-CG

Our second experiment gives a first indication of the performance characteristics of the two parallelization techniques for the CGMN method discussed in Section 4. We start with an experiment similar to Experiment A, measuring the required iterations for different shifts. The test problem is a graphene sheet of now fixed size 1024×1024 atoms. This results in a matrix of size $1024^2 \times 1024^2$, and for each of the eight shifts a system with eight right-hand sides is solved. For this experiment, we employed a single socket of the Emmy HPC cluster.

Access to the Emmy HPC cluster, named after Amalie Emmy Noether, was kindly provided by RRZE, the IT service provider of Friedrich-Alexander-Universität Erlangen-Nürnberg. The cluster features 560 compute nodes, each of which is equipped with two Intel(R) Xeon E5-2660V2 chips and 64 GB of shared memory. Each chip provides 10 SMT-enabled physical cores running at 2.2 GHz. The nodes are connected by an Infiniband network with 40 GBit/s bandwidth per link and direction. For experiments in this paper, up to 256 nodes (5120 physical cores) were used.

The results are shown in Fig. 2. It is clear from the plots that the variant that relies on multi-coloring (MC) takes between about 25% (one core) and 50% more time than the variant based on component averaging. This can be explained by an increasing number of scattered memory accesses (by construction there will be less memory re-use of the vector elements when accessing the matrix rows color-by-color). The disadvantage of the pure CARP-CG variant is a relatively large memory overhead, which we will see in the following experiments. We note that our multi-coloring based implementation is not optimal and it is possible to reduce the overhead of multi-coloring by code optimizations, but this is not the topic of this paper. To achieve the good performance of the MPI parallelization (by component averaging), we used persistent MPI communication. Without this, the scaling in a single socket and across multiple nodes was significantly worse than for the variant with coloring inside the socket.

Next, we investigate the scaling behavior beyond a single node. In Fig. 3 we show the weak (left) and strong (right) scaling behavior. The middle figure shows the speed-up achieved by solving several systems at once in a weak scaling experiment. A single linear system with a complex shift is solved for one or multiple right-hand sides and a convergence tolerance of 10^{-12}. The shifts used are not 'realistic' in the sense that the interval would contain far too many eigenvalues for the larger problems in an actual FEAST run, but for the performance experiments conducted here, this does not matter as the runtime of CGMN is proportional to the number of iterations.

We use one MPI process per socket and OpenMP with multi-coloring for the intra-socket parallelization. Using OpenMP within the full nodes leads to problems with the placement of data in the ccNUMA memory hierarchy that cannot easily be resolved for the KACZ kernel. This variant is compared with an MPI-only CARP-CG using lexicographic ordering on each subdomain (MPI partition) and 8 MPI processes per socket. To keep the comparison fair, only 8 of the 10 available cores are also used in the OpenMP variant. Using all 10 cores does not improve the overall runtime significantly due to the memory-bounded KACZ kernel,

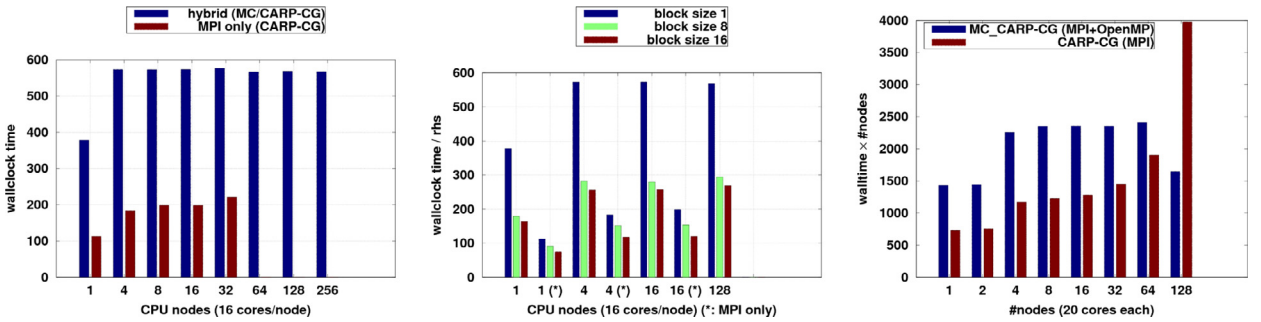

Fig. 3. Left: weak scaling for the CARP-CG and MC_CARP-CG variants (ca. 20M unknowns per node). Center: speed-up through blocking. Right: strong scaling, 20M unknowns in total. The bars are scaled such that a constant bar height indicates linear speed-up w.r.t. the number of nodes used. The groups of bars marked by an asterisk (∗) denote the MPI-only CARP-CG variant.

and leads to even more memory overhead for the MPI-only variant. The problem size of $3200 \times 6400 \approx 2 \times 10^7$ on each Emmy node already exceeds what can be done by a direct solver such as PARDISO [46] in the Intel(R) Math Kernel Library MKL. The total problem size of more than five billion degrees of freedom could not be handled by any parallel direct solver today, especially since the subspace in the FEAST algorithm has to be stored as well. The problem size we achieve is realistic for presently manufacturable graphene structures [47].

In the left and middle plot of Fig. 3, the problem size grows linearly with the number of nodes, with ∼ 20 million matrix rows per node (the graphene sheet is extended by doubling the number of atoms along the shortest edge when doubling the number of nodes used). In the right plot, the total problem size is fixed to ∼ 20 million in total, so the number of rows per process diminishes with increasing node count. A very simple z-partitioning is used, achieved by numbering the grid points in a lexicographic order and assigning consecutive blocks of rows to MPI processes. Obviously this is a poor choice, but even with this ordering we get near linear scaling between the nodes. The lower socket-level performance of our coloring-based implementation can be seen on the larger scale, however, as it requires significantly fewer MPI processes (and thus internal buffer space), its lower memory requirements allow scaling the application to higher node counts.

In the middle plot we solve for $n_{vec} = 1, 8, 16$ vectors simultaneously. Scalar products are bundled, even though a separate Krylov subspace is used for each right-hand side. The value shown is the time required per right-hand side of the linear system, multiplied by the number of nodes to make the results easier to compare. Here we see that the use of block operations alleviates the poor ratio of memory transfers to flops in our current implementation of multi-colored KACZ, whereas it increases the memory overhead of pure MPI CARP-CG linearly with the block size, restricting this approach to even fewer nodes (16 instead of 32 for a single system). For the two-dimensional test matrices from graphene simulation, we observe excellent parallel performance as the amount of data communicated is roughly proportional to $\sqrt{N}$ (the performance is bounded by the main memory of each node, not network bandwidth or latency). The time required for finding the distance-2 coloring on each MPI process is negligible.

5.2.1. A 3D problem

In order to show how the performance results carry over to three-dimensional problems, we use another class of test matrices in a final experiment. The test problem is a model problem for Anderson localization and was studied using ILUPACK in [19]. The matrix shows the sparsity pattern of a standard 7-point Laplace operator on a structured grid with periodic boundary conditions. The diagonal contains random numbers between $-L/2$ and $+L/2$, where we use $L = 16.5$ here. The shift is chosen as $z \approx -0.25 + 7.8 \times 10^{-3}\mathbf{i}$, and again, we solve for eight right-hand sides. For this model problem, the interval has to be contracted proportionally to the problem size N in order to keep the number of eigenvalues inside constant, which makes the number of iterations grow more sharply in an actual FEAST run with the present solver. The number of iterations is found to be roughly constant (around 900 for a residual tolerance of 10^{-10}) as the number of nodes is increased while keeping the number of unknowns per node fixed to $256^3 \approx 16$M. We stick to the simple z-partitioning because it is not clear whether graph partitioning tools such as ParMETIS [48] will be feasible for extremely large problems in the future. Obviously, a simpler idea such as octree partitioning could be used in this particular case, and would certainly improve the inter-node performance we achieve here.

Fig. 4 shows the weak scaling behavior. We expect the strong scaling to be slightly worse because of the decreasing ratio of computation to communication. However, as we are primarily interested in solving very large problems and our FEAST implementation offers another level of parallelism (the manager/worker model, see above), we did not perform the strong scaling experiment here. Even more than in 2D, 3D problems quickly hit the memory barrier when addressed by direct methods or ILU-type preconditioners such as ILUPACK, and the problem size of approximately 2.1 billion (128 nodes $\times$ 256^3 unknowns/node) shown here is well beyond what could be achieved by such an approach today. In the 3D case the performance shifts from memory-bounded towards communication-bounded such that our coloring approach is more competitive compared to a pure MPI implementation; but as mentioned before, smarter partitioning algorithms could be used to improve this.

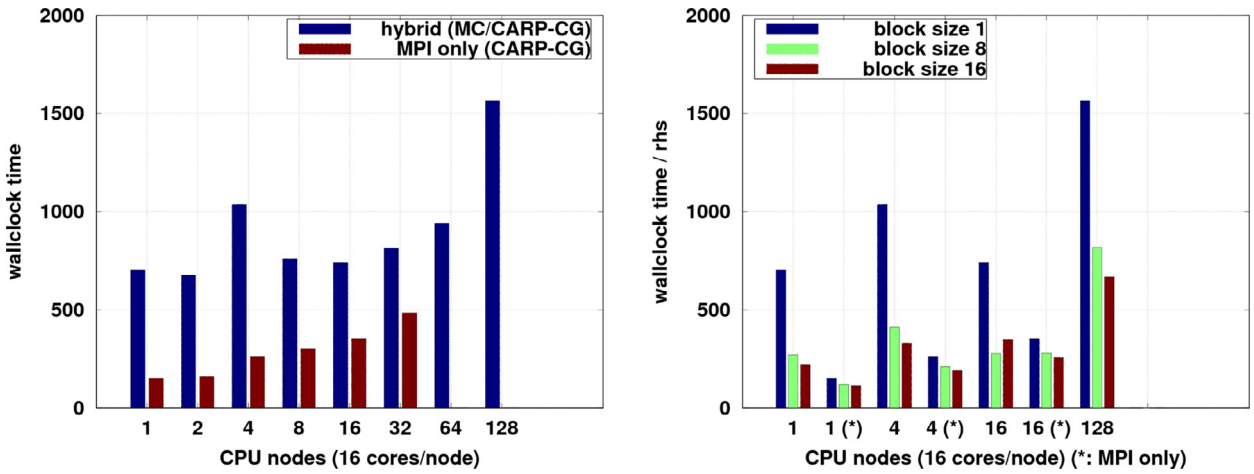

Fig. 4. Weak scaling experiment for the 3D Anderson model problem, $256^3 \approx 16$ M unknowns per node and one/multiple right-hand side(s) (left/right).

6. Conclusion

We have developed a fully iterative implementation of the FEAST algorithm to compute all eigenvalues and corresponding eigenvectors of a large sparse symmetric or Hermitian matrix in a given interval from the interior of the spectrum. We have summarized the numerical properties of the shifted linear systems in question that made a direct solution of these systems infeasible and proposed to use a row projection method called CGMN in the context of FEAST. In order to parallelize this scheme for use on modern hybrid-parallel HPC clusters, the existing CARP-CG variant of CGMN was extended to use multi-coloring for leveraging intra-node parallelism and reducing the memory overhead of CARP-CG when implemented using MPI only.

In numerical experiments with FEAST, CGMN proves to be very robust when applied to the considered class of matrices from graphene simulation and Anderson localization. In all cases it eventually achieves convergence to the desired tolerance of 10^{-12}. The drawback of the considered algorithm is that the number of iterations increases with growing matrix size if the number of desired eigenvalues is kept constant. An obvious remedy is to keep the interval size constant as the problem size grows. This leads to a roughly linearly increasing number of eigenvalues in the interval and therefore a likewise increasing size of the search space. Our FEAST implementation allows for compensation of this effect by using more worker processes.

Future work will focus on further reduction of the number of iterations needed by the linear solver. This can be achieved in part by well-known techniques such as using a block variant of CG instead of single CG iterations for the various right-hand sides. Furthermore, it may be possible to use a relaxed convergence tolerance in early FEAST iterations, when the Ritz values are still far from convergence. These ideas will not, however, solve the problem of the linearly increasing conditioning of the systems as the problem size grows and the spectrum becomes increasingly dense. We intend to investigate the use of multilevel acceleration. It is also intended to investigate techniques that can bring the shifts further away from the real axis without sacrificing accuracy.

Another important investigation will be the application of the method to eigenvalue problems stemming from other research areas.

Low-level optimization of the KACZ kernel on a variety of hardware, including multi-core CPUs, GPGPUs and accelerators such as the Intel(R) Xeon Phi, will also drastically reduce the overall run-time, as we have shown that the overall performance for graphene sheets is dictated by the node-level performance.

Acknowledgments

The authors thank Andreas Pieper and Andreas Alvermann (Ernst-Moritz-Arndt-Universität Greifswald) for useful discussions and comments on our code. The authors would also like to thank the unknown referees whose comments helped to make the exposition clearer.

References

[1] T. Auckenthaler, V. Blum, H.-J. Bungartz, T. Huckle, R. Johanni, L. Krämer, B. Lang, H. Lederer, P.R. Willems, Parallel solution of partial symmetric eigenvalue problems from electronic structure calculations, Parallel Comput. 37 (12) (2011) 783–794.
[2] M. Galgon, L. Krämer, B. Lang, A. Alvermann, H. Fehske, A. Pieper, Improving robustness of the FEAST algorithm and solving eigenvalue problems from graphene nanoribbons, PAMM 14 (1) (2014) 821–822.
[3] Y. Saad, Numerical Methods for Large Eigenvalue Problems, second edition, SIAM, Philadelphia, PA, 2011.
[4] G.L.G. Sleijpen, H.A. van der Vorst, A Jacobi–Davidson iteration method for linear eigenvalue problems., SIAM J. Matrix Anal. Appl. 17 (2) (1996) 401–425.
[5] T. Ikegami, T. Sakurai, U. Nagashima, A filter diagonalization for generalized eigenvalue problems based on the Sakurai-Sugiura projection method, J. Comput. Appl. Math. 233 (8) (2010) 1927–1936.
[6] T. Sakurai, H. Sugiura, A projection method for generalized eigenvalue problems using numerical integration, J. Comput. Appl. Math. 159 (2003) 119–128.

[7] J. Asakura, T. Sakura, H. Tadano, T. Ikegami, K. Kimura, A numerical method for nonlinear eigenvalue problems using contour integrals, JSIAM Lett. 1 (2009) 52–55.
[8] E. Polizzi, Density-matrix-based algorithm for solving eigenvalue problems, Phys. Rev. B 79 (2009) 115112.
[9] L. Krämer, E. Di Napoli, M. Galgon, B. Lang, P. Bientinesi, Dissecting the FEAST algorithm for generalized eigenproblems, J. Comput. Appl. Math. 244 (2013) 1–9.
[10] L. Krämer, Integration based solvers for standard and generalized Hermitian eigenvalue problems, Bergische Universität Wuppertal, 2014 Ph.D. thesis.Published electronically, http://nbn-resolving.de/urn/resolver.pl?urn=urn:nbn:de:hbz:468-20140701-112141-6
[11] P.T.P. Tang, E. Polizzi, FEAST as a subspace iteration eigensolver accelerated by approximate spectral projection, SIAM J. Matrix Anal. Appl. 35 (2) (2014) 354–390.
[12] S. Güttel, E. Polizzi, P. Tang, G. Viaud, Zolotarev quadrature rules and load balancing for the FEAST eigensolver, 2014. The University of Manchester, MIMS EPrint 2014.39, http://www.manchester.ac.uk/mims/eprints.
[13] Y. Saad, Iterative Methods for Sparse Linear Systems, 2nd edition, SIAM, Philadelphia, PA, 2003.
[14] L.N. Trefethen, D. BauIII, Numerical Linear Algebra, SIAM, Philadelphia, PA, 1997.
[15] H.M. Aktulga, L. Lin, C. Haine, E.G. Ng, C. Yang, Parallel eigenvalue calculation based on multiple shift-invert Lanczos and contour integral based spectral projection method, Parallel Comput. 40 (7) (2014) 195–212.
[16] D. Gordon, R. Gordon, CARP-CG: a robust and efficient parallel solver for linear systems, applied to strongly convection dominated PDEs, Parallel Comput. 36 (9) (2010) 495–515.
[17] A.H. Castro Neto, F. Guinea, N.M.R. Peres, K.S. Novoselov, A. Geim, The electronic properties of graphene, Rev. Mod. Phys. 81 (2009) 109–162.
[18] A. Alvermann, A. Basermann, H. Fehske, M. Galgon, G. Hager, M. Kreutzer, L. Krämer, B. Lang, A. Pieper, M. Röhrig-Zöllner, F. Shahzad, J. Thies, G. Wellein, ESSEX: Equipping Sparse Solvers for Exascale, 2014. Preprint BUW-IMACM 14/31, http://www.imacm.uni-wuppertal.de/fileadmin/imacm/preprints/2014/imacm_14_31.pdf.
[19] O. Schenk, M. Bollhöfer, R.A. Römer, On large scale diagonalization techniques for the Anderson model of localization, SIAM J. Sci. Comput. 28 (3) (2006).293–283
[20] B.N. Parlett, The Symmetric Eigenvalue Problem, vol. 20 of Classics in Applied Mathematics, Classics edition, SIAM, Philadelphia, PA, 1998.
[21] P.J. Davis, P. Rabinowitz, Methods of Numerical Integration, second edition, Academic Press, Orlando, FL, 1984.
[22] T.A. Davis, Direct Methods for Sparse Linear Systems (Fundamentals of Algorithms 2), SIAM, Philadelphia, PA, USA, 2006.
[23] J. Liesen, P. Tichý, Convergence analysis of Krylov subspace methods, GAMM-Mitt. 27 (2) (2004) 153–173.
[24] W. Hackbusch, Multi-Grid Methods and Applications, Springer Series in Computational Mathematics, Springer, Berlin, Heidelberg, 2003.
[25] A. Brandt, I. Livshits, Wave-ray multigrid method for standing wave equations, Electron. Trans. Numer. Anal. 6 (1997) 162–181.
[26] A. Toselli, O. Widlund, Domain decomposition methods: algorithms and theory, vol. 34 of Series in Computational Mathematics, Springer, 2005.
[27] C. Farhat, J. Li, An iterative domain decomposition method for the solution of a class of indefinite problems in computational structural dynamics, Appl. Numer. Math. 54 (2) (2005) 150–166.
[28] Y.A. Erlangga, C. Vuik, C.W. Oosterlee, Comparison of multigrid and incomplete LU shifted-Laplace preconditioners for the inhomogeneous Helmholtz equation, Appl. Numer. Math. 56 (5) (2006) 648–666.
[29] M. Benzi, Preconditioning techniques for large linear systems: a survey, J. Comp. Phys. 182 (2) (2002) 418–477.
[30] M. Bollhöfer, J.I. Aliaga, A.F. Martín, E.S. Quintana-Ortí, ILUPACK, in: D.A. Padua (Ed.), Encyclopedia of Parallel Computing, Springer, 2011, pp. 917–926.
[31] R. Bramley, Row Projection Methods for Linear Systems, Center for Supercomputing Research and Development Urbana, Ill: CSRD report, University of Illinois at Urbana-Champaign, 1989.
[32] S. Kaczmarz, Angenäherte Auflösung von Systemen linearer Gleichungen, Bulletin International de l'Académie Polonaise des Sciences et des Lettres. Classe des Sciences Mathématiques et Naturelles. Série A, Sciences Mathématiques 1937 (1937) 355–357.
[33] Å. Björck, T. Elfving, Accelerated projection methods for computing pseudoinverse solutions of systems of linear equations, BIT 19 (2) (1979) 145–163.
[34] D. Gordon, R. Gordon, CGMN revisited: robust and efficient solution of stiff linear systems derived from elliptic partial differential equations, ACM Trans. Math. Softw. 35 (3) (2008) 18:1–18:27.
[35] D. Gordon, R. Gordon, Row scaling as a preconditioner for some nonsymmetric linear systems with discontinuous coefficients, J. Comput. Appl. Math. 234 (12) (2010) 3480–3495.
[36] D. Gordon, R. Gordon, Robust and highly scalable parallel solution of the Helmholtz equation with large wave numbers, J. Comput. Appl. Math. 237 (1) (2013) 182–196.
[37] T. Elfving, Block-iterative methods for consistent and inconsistent linear equations, Numer. Math. 35 (1) (1980) 1–12.
[38] R. Bramley, A. Sameh, Row projection methods for large nonsymmetric linear systems, SIAM J. Sci. Stat. Comput. 13 (1992) 168–193.
[39] M. Arioli, I.S. Duff, D. Ruiz, M. Sadkane, Block lanczos techniques for accelerating the block cimmino method, SIAM J. Sci. Comput. 16 (6) (1995) 1478–1511.
[40] D. Gordon, R. Gordon, Component-averaged row projections: a robust, block-parallel scheme for sparse linear systems, SIAM J. Sci. Comput. 27 (3) (2005) 1092–1117.
[41] Ü.V. Çatalyürek, J. Feo, A.H. Gebremedhin, M. Halappanavar, A. Pothen, Graph coloring algorithms for multi-core and massively multithreaded architectures, Parallel Comput. 38 (10–11) (2012) 576–594.
[42] A.H. Gebremedhin, D.C. Nguyen, M.M.A. Patwary, A. Pothen, Colpack: software for graph coloring and related problems in scientific computing, ACM Trans. Math. Softw. 40 (1) (2013) 1.
[43] D. Bozdağ, U. Catalyurek, A. Gebremedhin, F. Manne, E. Boman, F. Özgüner, A parallel distance-2 graph coloring algorithm for distributed memory computers, in: L.T. Yang, O.F. Rana, B. Di Martino, J. Dongarra (Eds.), High Performance Computing and Communications, vol. 3726 of Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2005, pp. 796–806.
[44] J.M. Elble, N.V. Sahinidis, P. Vouzis, GPU computing with Kaczmarz's and other iterative algorithms for linear systems, Parallel Comput. 36 (5–6) (2010) 215–231.
[45] M. Galgon, L. Krämer, B. Lang, Adaptive choice of projectors in projection based eigensolvers, 2015, submitted for publication, available from http://www.imacm.uni-wuppertal.de/fileadmin/imacm/preprints/2015/imacm_15_07.pdf.
[46] O. Schenk, K. Gärtner, Solving unsymmetric sparse systems of linear equations with PARDISO, Future Gener. Comput. Syst. 20 (3) (2004) 475–487.
[47] E. Wolf, Applications of Graphene: An Overview, SpringerBriefs in Materials, Springer International Publishing, 2014.
[48] G. Karypis, K. Schloegel, ParMETIS. Parallel Graph Partitioning and Sparse Matrix Ordering Library, 4.0, University of Minnesota, Department of Computer Science and Engineering, Minneapolis, 2013.