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*

Abstract

Methods for the solution of 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 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 which exploits parallelism on several levels. We present numerical examples where graphene modeling is one of the target applications.

Keywords: Parallel eigenvalue computation, FEAST, linear systems, CARP-CG, row projection methods, multi-coloring
2000 MSC: 65F10, 65F15, 65F50, 68W10

1. Introduction

The solution of eigenvalue problems

$$\mathsf{A}\mathsf{X} = \mathsf{B}\mathsf{X}\Lambda \tag{1}$$

with Hermitian matrices $\mathsf{A}, \mathsf{B} \in \mathbb{C}^{N\times N}$ 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 $z \mapsto z(\mathsf{B} - \mathsf{A})^{-1}\mathsf{B}$ of the matrix pair (A, B). Techniques of this class for solving certain eigenvalue problems have been pioneered by Sakurai and co-workers [5, 6, 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, 10, 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].

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

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

[☆]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

Email addresses: galgon@math.uni-wuppertal.de (Martin Galgon), lkraemer@math.uni-wuppertal.de (Lukas Krämer), jonas.thies@dlr.de (Jonas Thies), achim.basermann@dlr.de (Achim Basermann), lang@math.uni-wuppertal.de (Bruno Lang) for several complex shifts z and an $N \times m$ matrix Y. The solution of very large 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 [13] 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, see, e. g., [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 solution of 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 performance on hybrid shared/distributed memory machines. A limitation 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 the 3D Anderson model.

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. 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 introduce 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, in the following we will focus on the standard eigenvalue problem, i. e., $\mathsf{B} = \mathsf{I}$. The symbol C in (3) denotes a closed contour in the complex plane, surrounding the interval $I_\lambda = [\underline{\lambda}, \overline{\lambda}]$. We assume that the interior of C does not contain any other eigenvalue than those residing in I_λ. Approximating the matrix U in (3) leads to numerical integration of the function $z \mapsto (z\mathsf{I} - \mathsf{A})^{-1}\mathsf{Y}$, which can be formulated as

$$\mathsf{U} \approx \widetilde{\mathsf{U}} = \frac{1}{2\pi\mathbf{i}} \sum_{j=1}^{p} \omega_j \varphi'(t_j)(\varphi(t_j)\mathsf{I} - \mathsf{A})^{-1}\mathsf{Y}. \tag{4}$$

Here, $\varphi : [0, 2\pi] \to \mathbb{C}$ is a parametrization of 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 (4) the solution of p block linear systems

$$(z_j\mathsf{I} - \mathsf{A}) \cdot \mathsf{V} = \mathsf{Y} \tag{5}$$

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

```
Input: An interval $I_\lambda := [\underline{\lambda}, \overline{\lambda}\,]$ and an estimate $\widetilde{m}$ of the number of eigenvalues in $I_\lambda$.
Output: $\hat{m} \le \widetilde{m}$ eigenpairs with eigenvalue in $I_\lambda$.
1: Choose $\mathsf{Y} \in \mathbb{C}^{N\times\widetilde{m}}$ of rank $\widetilde{m}$ and compute an approximation $\widetilde{\mathsf{U}}$ to
       $$\mathsf{U} := \frac{1}{2\pi\mathbf{i}} \int_C (z\mathsf{I} - \mathsf{A})^{-1}\,\mathsf{Y}\,dz. \qquad (3)$$
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 $(\widetilde{\Lambda}, \widetilde{\mathsf{X}} := \widetilde{\mathsf{U}} \cdot \widetilde{\mathsf{W}})$.
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 from practice often yield very large and sparse matrices A. In this case, a method for the solution of (5) that is based on factorization of the system matrix $z_j\mathsf{I} - \mathsf{A}$ is not applicable; one has to resort to iterative solvers for (5). 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 promise 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]. Due to this consideration and the requirements of the described target applications—typically not more than a few hundred eigenpairs around a certain point inside the spectrum of the matrix are required—the multiple-intervals approach has not been pursued for use in 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 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 [18], as well as implementations of some iterative solvers for linear systems and extremal eigenvalue problems.

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 (5) 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 Sec. 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.

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 (for 3D problems it typically has a computational complexity of $O(N^2)$ where N is the dimension of the matrix) and memory consuming. Furthermore, the parallelization of direct methods is more difficult than it is for iterative methods.

Throughout this section, we consider a linear system $\mathsf{Mx} = \mathsf{b}$, $\mathsf{M} \in \mathbb{C}^{N\times N}$, with a single right hand side b.

The most popular alternative to direct methods are Krylov subspace methods which approximates the solution x in a low-dimensional Krylov subspace, $\mathcal{K}_k(\mathsf{M}, \mathsf{b}, \mathsf{x}_0) = \operatorname{span}\left\{\mathsf{x}_0, \mathsf{M}\mathsf{x}_0, \mathsf{M}^2\mathsf{x}_0, \ldots, \mathsf{M}^k\mathsf{x}_0\right\}$, $k \ll N$. 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 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 may still be quite reasonable if the distribution of the eigenvalues is favorable [22]. However, the eigenvalues of $z\mathsf{I} - \mathsf{A}$ will typically lie on both sides of the imaginary axis, which prevents methods as GMRES from converging fast [14]. The system matrix is not Hermitian even if A is, hence methods as MINRES and CG [13], tailored to Hermitian systems, are not applicable.

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 area, we can roughly estimate that the condition number of the system matrix A_z will be proportional to the inverse of the imaginary part of the shift. The shifts z are dictated by the FEAST algorithm. If one is only interested in a small portion of the spectrum, the interval I_λ and hence the contour C from (3) are small. This results in a small imaginary part of some of the shifts z and hence in a large condition number κ_z.

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 [23] 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 $\sigma(\mathsf{A}^{(ell)})$ 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 [24]. 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 [25, 26].

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 [27]. The only robust algorithm we are aware of is ILUPACK [28], which was applied to the inner eigenvalue problem of the Anderson model in [19]. For our purposes ILUPACK does not provide sufficient parallelism and the factorization consumes large amounts of memory and computing 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 [29]. One of the most well-known algorithms of this class is the Kaczmarz [30] 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 [31] published several algorithms based on conjugate gradient acceleration of row projection methods. We are particularly interested in their CGMN method, see also [32]. It can be interpreted as a conjugate gradient (CG, [14]) method on the minimum norm problem $\mathsf{M}\mathsf{M}^H\mathsf{y} = \mathsf{b}$ (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.** The inherent row scaling of the Kaczmarz method alleviates the relatively small diagonal elements occurring in our target application, graphene [33].

The operator to which the CG method is applied is $(\mathsf{I} - DKSWP(\mathsf{M}, \mathsf{b}, \mathsf{x}, \omega))$, where $DKSWP(\mathsf{M}, \mathsf{b}, \mathsf{x}, \omega)$ 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. Here 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).

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

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.

In [34] 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.

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 [35]. It requires choosing a partitioning of the rows of M into n block rows $\mathsf{M}_{i,:}$:

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

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 $\mathsf{M}_{i,:}\mathsf{M}_{i,:}^H$. For instances of this algorithm, see, e. g., [36, 37]. A special situation is the case where each $\mathsf{M}_{i,:}^H$ 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 $\mathsf{M}_{i,:}^H$. We will further discuss this in Section 4.1.

Gordon and Gordon [16, 38] 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 classical domain decomposition methods [25], 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 [38] 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 $\mathsf{M}_{i,:}\mathsf{M}_{i,:}^H$ 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 the algorithm. 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 [39, 40] and distributed [41] 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 duplicating certain nodes in the interior of the MPI partition. Such an approach was pursued in [42] for GPUs, but with moderate success. We will show that the increase in 'averaged' vector entries leads to poor strong scaling of CARP compared to multi-coloring in Section 5. 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 CARP-CG required iterations 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.

5.1. *Experiment A: convergence behavior*

We use our FEAST solver to 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 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 0 and the convergence criterion is implemented as $\|\mathsf{A}_z\mathsf{x} - \mathsf{b}\|_2/\|\mathsf{b}\|_2 <$ 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, 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 proportional to the inverse

				$\overline{\text{it}}_{\text{MC-CGMN}}$			
W	β	n_{ev}	it_{FEAST}	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

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

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 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 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.

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

Our second experiment aims at motivating the choices made for parallelizing the CGMN method in Section 4. The test problem is again a graphene sheet matrix of now fixed size 1024×1024 atoms. 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 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, partitions of at most 64 nodes (1280 physical cores) were available.

It is clear from Figure 1 that the MPI overhead and additional memory transfers due to the buffers for exchanging internal partition boundaries in a pure MPI/CARP implementation prevent parallel speed-up inside a shared memory node with such an approach. The results clearly justify our choice to use multi-coloring here, instead.

In Figure 2 we show the weak (left) and strong (right) scaling behavior beyond a single node. We use one MPI process per socket and OpenMP with multi-coloring for the intra-socket parallelization.

As can be seen in the left plot, the problem size grows with the number of nodes, with 4096^2 matrix rows per node. In the right plot, the total problem size is fixed to $N = 4096^2$, so the number of rows per process diminishes with increasing node count. We focus on a single shift and show the behavior for an increasing block size instead: the KACZ kernel is applied to $n_{vec} = 8, 16, 32, 64$ vectors simultaneously. Scalar products are bundled, even though a

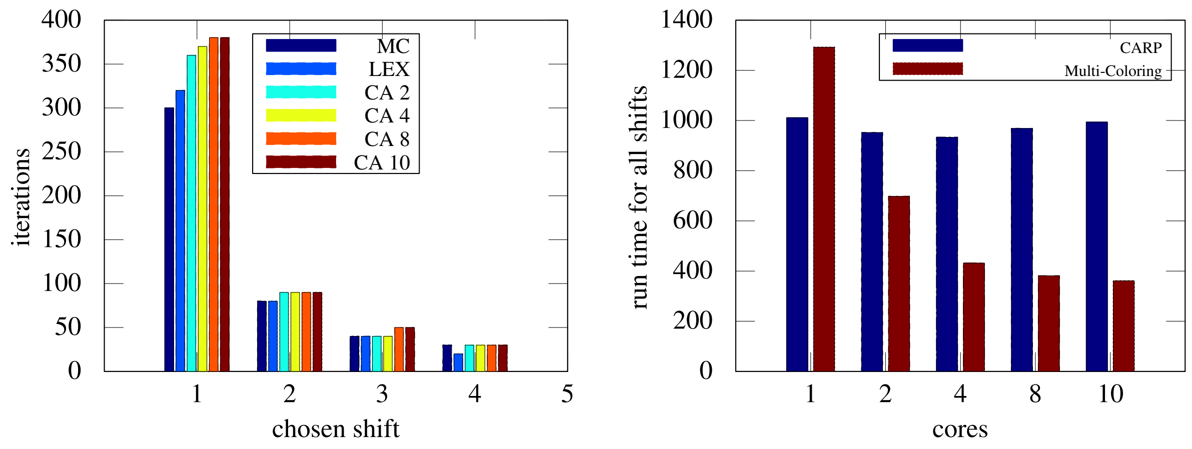

Figure 1: Multi-coloring vs. CARP inside a 10 core shared memory socket. The various bars marked 'CA-X' show the dependence of the number of iterations on the number of cores when CARP is used. LEX: using lexicographic ordering (serial Kaczmarz). MC: using multi-coloring (parallel Kaczmarz).

separate Krylov subspace is used for each right-hand side. The time 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.

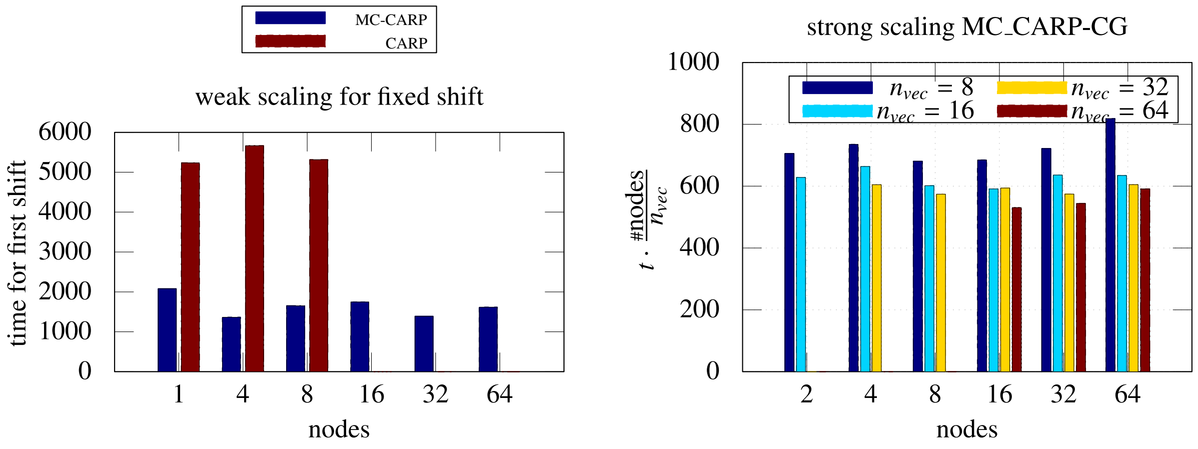

Figure 2: MC-CARP-CG, weak scaling (left, ca. 16M dof/node), strong scaling and block operations (right, ca. 16M dof in total).

For the two-dimensional test matrices from graphene simulation, we observe excellent parallel performance as the amount of data communicated is proportional to $\sqrt{N}$. The time required for finding the distance-2 coloring on each MPI process is negligible. Shown is the time for eight shifts and eight right-hand sides per shift with 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. In the right plot, it can be seen that the method performs very well for multiple right-hand sides simultaneously, even though we do not yet use a block CG variant. Operating on multiple vectors at once drastically improves the performance.

In order to show that the good performance results carry over to three-dimensional problems as well, 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 $z \approx -0.25 + 7.8 \cdot 10^{-3}\mathbf{i}$, 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

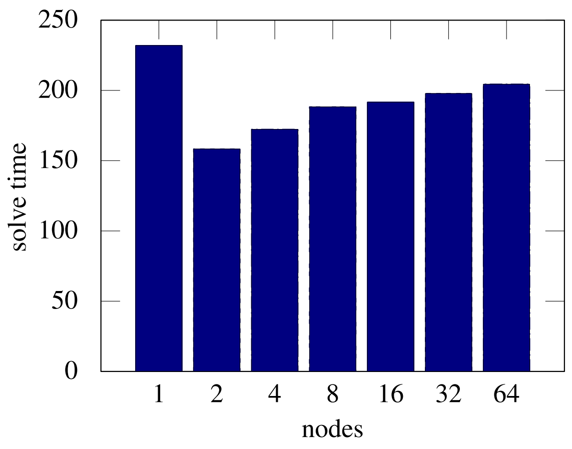

Figure 3: Weak scaling experiment for the 3D Anderson model problem, 100^3 unknowns per node and eight vectors iterated simultaneously.

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 one million. ParMETIS [43] was used for partitioning the matrix to reduce communication. The time spent for repartitioning the matrix and finding the graph coloring were negligible compared to the time required for solving the system. Figure 3 shows the weak scaling behavior. We expect that the strong scaling will 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), we did not perform the strong scaling experiment here.

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. To avoid direct solution of the linear systems, we have summarized the numerical properties of the shifted linear systems in question 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 improved intra-node performance and overall scalability.

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. Another important investigation will be to apply the method to eigenvalue problems from other application areas.

Acknowledgments

The authors thank Andreas Pieper and Andreas Alvermann (Ernst-Moritz-Arndt-Universität Greifswald) for useful discussions and comments on our code.

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, preprint, submitted for publication (2014).

[3] Y. Saad, Numerical Methods for Large Eigenvalue Problems, 2nd 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 Letters 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, Ph.D. thesis, Bergische Universität Wuppertal, published electronically, `http://nbn-resolving.de/urn/resolver.pl?urn=urn:nbn:de:hbz:468-20140701-112141-6` (2014).

[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, The University of Manchester, MIMS EPrint 2014.39, `http://www.manchester.ac.uk/mims/eprints` (2014).

[13] Y. Saad, Iterative Methods for Sparse Linear Systems, 2nd Edition, SIAM, Philadelphia, PA, 2003.

[14] L. N. Trefethen, D. Bau, III, 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. G. andN. 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, preprint BUW-IMACM 14/31, `http://www.imacm.uni-wuppertal.de` (2014).

[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, Classics Edition, Vol. 20 of Classics in Applied Mathematics, SIAM, Philadelphia, PA, 1998.

[21] P. J. Davis, P. Rabinowitz, Methods of numerical integration, 2nd Edition, Academic Press, Orlando, FL, 1984.

[22] J. Liesen, P. Tichý, Convergence analysis of Krylov subspace methods, GAMM-Mitt. 27 (2) (2004) 153–173.

[23] W. Hackbusch, Multi-Grid Methods and Applications, Springer Series in Computational Mathematics, Springer, Berlin, Heidelberg, 2003.

[24] A. Brandt, I. Livshits, Wave-ray multigrid method for standing wave equations, Electron. Trans. Numer. Anal 6 (1997) 162–181.

[25] A. Toselli, O. Widlund, Domain Decomposition Methods: Algorithms and Theory, Vol. 34 of Series in Computational Mathematics, Springer, 2005.

[26] 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.

[27] M. Benzi, Preconditioning techniques for large linear systems: A survey, J. Comp. Phys. 182 (2) (2002) 418 – 477.

[28] 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.

[29] 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.

[30] 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.

[31] Å. Björck, T. Elfving, Accelerated projection methods for computing pseudoinverse solutions of systems of linear equations, BIT 19 (2) (1979) 145–163.

[32] 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.

[33] 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.

[34] 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.

[35] T. Elfving, Block-iterative methods for consistent and inconsistent linear equations, Numer. Math. 35 (1) (1980) 1–12.

[36] R. Bramley, A. Sameh, Row projection methods for large nonsymmetric linear systems, SIAM J. Sci. and Stat. Comput. 13 (1992) 168–193.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] G. Karypis, K. Schloegel, ParMETIS. Parallel Graph Partitioning and Sparse Matrix Ordering Library, University of Minnesota, Department of Computer Science and Engineering, Minneapolis, 4th Edition (2013).
URL `http://glaros.dtc.umn.edu/gkhome/metis/parmetis`