DOMAIN DECOMPOSITION APPROACHES FOR ACCELERATING CONTOUR INTEGRATION EIGENVALUE SOLVERS FOR SYMMETRIC EIGENVALUE PROBLEMS

VASSILIS KALANTZIS*, JAMES KESTYN†, ERIC POLIZZI‡, AND YOUSEF SAAD*

Abstract. This paper discusses techniques for computing a few selected eigenvalue-eigenvector pairs of large and sparse symmetric matrices. A recently developed powerful class of techniques to solve this type of problems is based on integrating the matrix resolvent operator along a complex contour that encloses the interval containing the eigenvalues of interest. This paper considers such contour integration techniques from a domain decomposition viewpoint, and proposes two schemes. The first scheme can be seen as an extension of domain decomposition linear system solvers in the framework of contour integration methods for eigenvalue problems, such as FEAST. The second scheme focuses on integrating the resolvent operator primarily along the interface region defined by adjacent subdomains. A parallel implementation of the proposed schemes is described and results on distributed computing environments reported. These results show that domain decomposition approaches can lead to reduced runtimes and improved scalability.

Key words. Domain decomposition, symmetric eigenvalue problem, Cauchy integral formula, parallel computing, FEAST.

AMS subject classifications. 15A18, 65F10, 65F15, 65F50

1. Introduction.

This paper addresses the problem of computing all eigenvalues located in an interval $[\alpha, \beta]$ and their associated eigenvectors of a sparse real symmetric matrix A of size $n \times n$. A common approach for solving this type of problems is via a Rayleigh-Ritz (projection) process on a well-selected low-dimensional subspace $\mathcal{U}$. In an ideal situation, $\mathcal{U}$ spans an invariant subspace associated with the sought eigenvalues.

Much interest has been generated in recent years by techniques in which the subspace $\mathcal{U}$ is extracted via an approximation of the spectral projector obtained by numerically integrating the resolvent $(\zeta I - A)^{-1}$, $\zeta \in \mathbb{C}$ on a closed complex contour Γ that encloses the desired eigenvalues. The core of the method is then to compute the action of the matrix contour integral $\int_\Gamma (\zeta I - A)^{-1} d\zeta$ on a set of vectors. From a numerical viewpoint, contour integration eigenvalue solvers (eigensolvers) can be viewed as rational filtering techniques that convert the solution of the original eigenvalue problem into the solution of a series of complex symmetric linear systems with multiple right-hand sides. Popular algorithms of the class of contour integration-based eigensolvers are the FEAST method of Polizzi [32, 45] and the SS method of Sakurai and Sugiura [40, 41].

We focus on distributed computing environments, possibly with a large number of processors, and study contour integration eigensolvers from a domain decomposition (DD) viewpoint [43, 46]. In a domain decomposition approach the (discretized) computational domain is partitioned into a number of subdomains, each assigned to a different processor group. Domain decomposition techniques for the solution of eigenvalue problems have been studied in the past, see for example [4, 9, 10, 20, 27, 28, 31], but to our knowledge these methods have not yet been considered within a contour integration framework.

*Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455, USA (kalantzi@cs.umn.edu, saad@cs.umn.edu). This work supported jointly by NSF under award CCF-1505970 (theoretical aspects) and by the Scientific Discovery through Advanced Computing (SciDAC) program funded by U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research and Basic Energy Sciences under award number DE-SC0008877 (Implementations, application to DFT).

†Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA 01003, USA (kestyn@ecs.umass.edu, polizzi@ecs.umass.edu). This work supported by NSF under award CCF-1510010.

Contour integration eigensolvers have been mostly associated with the use of sparse direct solvers to solve the linear systems which arise from the numerical approximation of the contour integral. This approach, however, is not always feasible due to the possible large amount of fill-in in the triangular factors, e.g., when factorizing matrices that originate from discretizations of 3D computational domains. One of the goals of this paper is to fill part of the gap that exists between contour integration approaches and the use of hybrid iterative solvers within their context. In particular, we would like to accelerate iterative solutions of the linear systems encountered in the FEAST algorithm, by utilizing domain decomposition preconditioners to solve the complex linear systems with multiple right-hand sides.

Even when direct solvers are the most practical way to solve the linear systems encountered in a FEAST approach, domain decomposition based approaches can lead to faster and more scalable computations. This will be illustrated by comparing the FEAST algorithm implemented with a domain decomposition-based direct solver, with an implementation of FEAST that uses a standard parallel sparse direct solver.

Domain decomposition type methods naturally lend themselves for parallelization in multi-core and/or many-core environments. The paper discusses practical aspects of an implementation of the proposed numerical schemes in distributed computing environments.

Finally, we discuss a modified contour integration scheme which approximates only certain parts of the contour integral of the matrix resolvent. This leads to a numerical scheme that can be computationally more efficient than following the standard approach of numerically integrating the entire resolvent.

The organization of this paper is as follows. In Section 2 we describe the main idea behind contour integration eigensolvers, and FEAST algorithm in particular. In Section 3 we describe the domain decomposition framework. In Section 4 we present two computational schemes within the context of domain decomposition. Section 5 focuses on the solution of the linear systems during the numerical integration phase, and discusses the use of domain decomposition-based preconditioners, as well as their implementation in distributed computing environments. In Section 6 we present computational experiments. Finally, in Section 7, we state a few concluding remarks.

2. Contour integration-based eigenvalue solvers. For simplicity, throughout this paper we will assume that the sought eigenpairs of A lie inside the interval $[-1, 1]$. The above assumption is not restrictive since the eigenvalues of A located inside any real interval $[\alpha, \beta]$ can be mapped to the interval $[-1, 1]$ by the following linear transformation:

$$A := (A - cI)/e, \;\; c = \frac{\alpha + \beta}{2}, \;\; e = \frac{\beta - \alpha}{2}. \tag{2.1}$$

Now let A have r eigenvalues, denoted by λ_1, ..., λ_r, located in the interval $[-1, 1]$, and let $X = [x^{(1)}, \ldots, x^{(r)}]$ be the $n \times r$ orthonormal matrix formed by the associated eigenvectors. Then, the spectral projector $\mathcal{P} = x^{(1)}(x^{(1)})^T + \ldots + x^{(r)}(x^{(r)})^T = XX^T$ can be expressed via the Cauchy integral [36]:

$$\mathcal{P} = \frac{1}{2i\pi} \int_\Gamma (\zeta I - A)^{-1} d\zeta, \tag{2.2}$$

where Γ is a smooth, counter-clockwise oriented curve that encloses only the sought eigenvalues $\lambda_1, \ldots, \lambda_r$. The invariant subspace associated with the eigenvectors $x^{(1)}$, ..., $x^{(r)}$ can be then captured by multiplying $\mathcal{P}$ by some full-rank matrix $V \in \mathbb{R}^{n \times \hat{r}}$, where it is assumed that $V^T X$ has rank r. The span of $\mathcal{P}V$ can then exploited by a Rayleigh-Ritz projection procedure to extract eigenpairs $(\lambda_1, x^{(1)}), \ldots, (\lambda_r, x^{(r)})$.

In practice, the spectral projector in (2.2) is approximated by numerical quadrature, e.g., an N_c-point Gaussian quadrature rule. A basis of the approximate invariant subspace then takes the form:

$$\mathcal{P}V \approx \tilde{\mathcal{P}}V = \sum_{j=1}^{N_c} \omega_j (\zeta_j I - A)^{-1} V, \tag{2.3}$$

where $\{\zeta_j, \omega_j\}_{1 \le j \le N_c}$ are the N_c quadrature node-weight pairs of the quadrature rule. The numerical integration scheme in (2.3) approximates the exact spectral projector $\mathcal{P}$ in (2.2) by the approximate projector $\tilde{\mathcal{P}} = \rho(A)$, where

$$\rho(\zeta) = \sum_{j=1}^{N_c} \frac{\omega_j}{\zeta_j - \zeta}. \tag{2.4}$$

The rational function $\rho(\zeta)$ can be interpreted as a "spectral" filter function which maps the eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ of A to the eigenvalues $\rho(\lambda_1)$, $\rho(\lambda_2)$, ..., $\rho(\lambda_n)$ of $\rho(A)$. In an ideal situation $\rho(\zeta) \approx 1$, for $\zeta \in [-1, 1]$ and $\rho(\zeta) \approx 0$, for $\zeta \notin [-1, 1]$.

If the accuracy of the approximate eigenpairs computed by a Rayleigh-Ritz projection on the subspace created by (2.3) is not satisfactory, the procedure can be repeated using the most recent approximate eigenvectors as the new matrix V to multiply $\tilde{\mathcal{P}}$. If direct solvers are used to solve the complex linear systems in (2.3), this approach essentially amounts to subspace iteration with matrix A replaced by $\rho(A)$. This approach has been implemented in the FEAST package introduced by Polizzi [22, 32, 45]. The subspace iteration viewpoint relaxes the requirement for a highly accurate approximation of the exact spectral projector $\mathcal{P}$. It is also possible to consider contour integrals of other rational functions, e.g., the scalar function $u^*(\zeta I - A)^{-1} v$, with $u,\ v,\ \in \mathbb{C}^n$, as proposed by Sakurai and Sugiura (SS) [40, 41]. The poles of this scalar function are the eigenvalues of A.

This paper focuses on the FEAST framework, i.e., on an approach based on a filtered subspace iteration.

3. The domain decomposition framework. In a domain decomposition framework, we typically begin by subdividing the computational domain into P partitions (subdomains) using a graph partitioner [21, 29]. The partitioning can be either recursive, using nested dissection [16], or P-way, where the domain is partitioned in P subdomains directly. Throughout this paper, we assume a non-overlapping P-way partitioning of the domain, where each vertex is a pair equation-unknown (equation number i and unknown number i) and the partitioner subdivides the vertex set into P partitions, i.e., P non-intersecting subsets whose union is equal to the original vertex set (the aforementioned is also known either as vertex-based or edge-separator partitioning).

Once the graph is partitioned, we can identify three different types of unknowns: (1) interior unknowns that are coupled only with local equations; (2) local interface unknowns that are coupled with both non-local (external) and local equations; and (3) external interface unknowns that belong to other subdomains and are coupled with local interface variables. Within the i^{th} subdomain $i = 1, \ldots, P$, a local reordering is applied in which interior points are listed before the interface ones. With this, the local eigenvector x_i can be split into two parts: the subvector u_i of internal components followed by the subvector y_i of local interface components. Let now subdomain i have d_i interior variables and s_i interface variables, i.e., the length of vectors u_i and y_i is d_i and s_i respectively. We denote by $B_i \in \mathbb{R}^{d_i \times d_i}$ the matrix that represents the couplings between the interior variables, i.e., between variables in u_i. Similarly, we denote by $\hat{E}_i \in \mathbb{R}^{d_i \times s_i}$ the matrix that maps interface variables of subdomain i to

interior variables of subdomain i, and by $C_i \in \mathbb{R}^{s_i \times s_i}$ the matrix that represents the couplings between the interface variables of subdomain i, i.e., between variables in y_i. Finally, we let $E_{ij} \in \mathbb{R}^{s_i \times s_j}$ be the matrix representing the couplings between external interface unknowns from subdomain j and local interface variables of subdomain i.

When block partitioned according to the above splitting, the equation $(A - \lambda I)x = 0$ can be written locally as

$$\underbrace{\begin{pmatrix} B_i - \lambda I & \hat{E}_i \\ \hat{E}_i^T & C_i - \lambda I \end{pmatrix}}_{A_i} \underbrace{\begin{pmatrix} u_i \\ y_i \end{pmatrix}}_{x_i} + \begin{pmatrix} 0 \\ \sum_{j \in N_i} E_{ij} y_j \end{pmatrix} = 0. \tag{3.1}$$

Matrix A_i and vector x_i are local in the i^{th} subdomain, and N_i is the set of indices of the subdomains that are neighbors to the i^{th} subdomain. The term $E_{ij}y_j$ is a part of the product which reflects the contribution to the local equation from the neighboring subdomain j. Note that E_{ij} ($i \neq j$) is non-zero only if subdomains i and j are nearest neighbors. The second block of rows in (3.1) contains the couplings between the local variables of the i^{th} subdomain and external interface variables from neighboring subdomains, and gives the following equation:

$$\hat{E}_i^T u_i + (C_i - \lambda I) y_i + \sum_{j \in N_i} E_{ij} y_j = 0.$$

The action of the operation on the left-hand side of the above equation on the vector of all interface variables, i.e., the vector $y^T = [y_1^T, y_2^T, \cdots, y_P^T]$, can be accomplished into the following matrix $C \in \mathbb{R}^{s \times s}$:

$$C = \begin{pmatrix} C_1 & E_{12} & \dots & E_{1P} \\ E_{21} & C_2 & \dots & E_{2P} \\ \vdots & \vdots & \ddots & \vdots \\ E_{P1} & E_{P2} & \dots & C_P \end{pmatrix}. \tag{3.2}$$

where $E_{ij} = 0$ if $j \notin N_i$, and $s = s_1 + s_2 + \cdots + s_p$.

Thus, if we stack all interior variables $u_1, u_2, \cdots, u_P$ into a vector u, in this order, and we reorder the equations so that the u_i's are listed first followed by the y_i's, we obtain a reordered global eigenvalue problem that has the following form:

$$\underbrace{\begin{pmatrix} B_1 & & & & E_1 \\ & B_2 & & & E_2 \\ & & \ddots & & \vdots \\ & & & B_P & E_P \\ E_1^T & E_2^T & \dots & E_P^T & C \end{pmatrix}}_{PAP^T} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_P \\ y \end{pmatrix} = \lambda \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_P \\ y \end{pmatrix}, \tag{3.3}$$

The matrix E_i, $i = 1, \ldots, p$, $E_i \in \mathbb{R}^{d_i \times s}$, in (3.3) is an expanded version of the corresponding matrix $\hat{E}_i$ defined earlier and used in (3.1). More specifically, we have $E_i = [0_{d_i,\ell_i}, \hat{E}_i, 0_{d_i,\nu_i}]$, where $\ell_i = \sum_{j=1}^{j<i} s_j$, $\nu_i = \sum_{j>i}^{j=P} s_j$, and $0_{\chi,\psi}$ denotes the zero matrix of size $\chi \times \psi$. Note in particular that we have $E_i y = \hat{E}_i y_i$.

Figure 3.1 shows an example of a Laplacian operator discretized in two dimensions (2D) using finite differences. The left subfigure shows the sparsity pattern after local reordering,

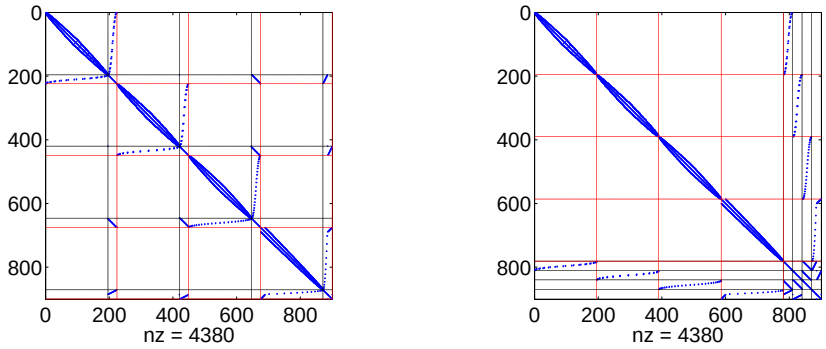

FIG. 3.1. *An example of a 2D Laplacian matrix partitioned in $P = 4$ subdomains and reordered according to (3.1) (left) and (3.3) (right).*

while the right subfigure shows the sparsity pattern if we assume that interior variables across all subdomains are ordered before the interface ones. The coefficient matrix of the system (3.3) can be also written in a more compact form as

$$A = \begin{pmatrix} B & E \\ E^T & C \end{pmatrix}, \tag{3.4}$$

where we kept the original symbol A for the permuted matrix as well. For the rest of this paper we will assume that the matrix A is represented as in (3.4).

4. Contour integration in the domain decomposition framework. In this section we present two numerical schemes which utilize contour integration approaches from a domain decomposition perspective. Both schemes start with the expression of the resolvent operator $(\zeta I - A)^{-1}$ under the domain decomposition framework.

4.1. Full integration of the matrix resolvent. For $\zeta \in \mathbb{C}$ consider the complex shifted matrix $A - \zeta I$ written through its block LU factorization [14]:

$$A - \zeta I = \begin{pmatrix} I & 0 \\ E^T(B - \zeta I)^{-1} & I \end{pmatrix} \begin{pmatrix} B - \zeta I & E \\ 0 & S(\zeta) \end{pmatrix}, \tag{4.1}$$

where

$$S(\zeta) = C - \zeta I - E^T(B - \zeta I)^{-1}E \tag{4.2}$$

is a matrix-valued rational function known as the Schur complement matrix. Then, the negated resolvent operator $-(\zeta I - A)^{-1} = (A - \zeta I)^{-1}$ can be expressed through the identity

$$(A - \zeta I)^{-1} = \begin{pmatrix} (B - \zeta I)^{-1} & -(B - \zeta I)^{-1}ES(\zeta)^{-1} \\ 0 & S(\zeta)^{-1} \end{pmatrix} \begin{pmatrix} I & 0 \\ -E^T(B - \zeta I)^{-1} & I \end{pmatrix}, \tag{4.3}$$

where both $(B - \zeta I)$ and $S(\zeta)$ are assumed to be non-singular (this assumption is trivially satisfied for any complex ζ with non-zero imaginary part).

Multiplying the two block triangular matrices in (4.3), and defining $F(\zeta) = (B - \zeta I)^{-1}E$, we get:

$$(A - \zeta I)^{-1} = \begin{pmatrix} (B - \zeta I)^{-1} + F(\zeta)S(\zeta)^{-1}F(\zeta)^T & -F(\zeta)S(\zeta)^{-1} \\ -S(\zeta)^{-1}F(\zeta)^T & S(\zeta)^{-1} \end{pmatrix}. \tag{4.4}$$

Let now Γ be a counter-clockwise oriented smooth Jordan curve, e.g., a circle, that encloses only the eigenvalues of A inside $[-1,1]$, and let $\mathcal{P}$ denote the associated spectral projector defined in (2.2). Then, the spectral projector $\mathcal{P}$ can be written in a 2×2 block form, by integrating each block of $(A-\zeta I)^{-1}$ separately:

$$\mathcal{P} = \frac{-1}{2i\pi}\int_\Gamma (A-\zeta I)^{-1}d\zeta \equiv \begin{pmatrix} \mathcal{H} & -\mathcal{W} \\ -\mathcal{W}^\mathcal{T} & \mathcal{G} \end{pmatrix} \tag{4.5}$$

with

$$\begin{cases} \mathcal{H} = \frac{-1}{2i\pi}\int_\Gamma [(B-\zeta I)^{-1} + F(\zeta)S(\zeta)^{-1}F(\zeta)^T]d\zeta \\ \mathcal{G} = \frac{-1}{2i\pi}\int_\Gamma S(\zeta)^{-1}d\zeta \\ \mathcal{W} = \frac{-1}{2i\pi}\int_\Gamma F(\zeta)S(\zeta)^{-1}d\zeta. \end{cases} \tag{4.6}$$

In order to extract an eigenspace from the expression of $\mathcal{P}$ in (4.5), we consider the product $\mathcal{P}V$, where V has $\hat{r}$ columns, written as

$$\mathcal{P}\begin{pmatrix} V_u \\ V_s \end{pmatrix} = \begin{pmatrix} \mathcal{H}V_u - \mathcal{W}V_s \\ -\mathcal{W}^T V_u + \mathcal{G}V_s \end{pmatrix} \equiv \begin{pmatrix} Z_u \\ Z_s \end{pmatrix}, \tag{4.7}$$

where $V = [V_u^T, V_s^T]^T$, and $V_u \in \mathbb{R}^{d\times\hat{r}},\ V_s \in \mathbb{R}^{s\times\hat{r}}$ are the parts of V that correspond to the local and interface variables, respectively. We finally get

$$\begin{cases} Z_u = \frac{-1}{2i\pi}\int_\Gamma (B-\zeta I)^{-1}V_u d\zeta - \frac{-1}{2i\pi}\int_\Gamma F(\zeta)S(\zeta)^{-1}[V_s - F(\zeta)^T V_u]d\zeta \\ Z_s = \frac{-1}{2i\pi}\int_\Gamma S(\zeta)^{-1}[V_s - F(\zeta)^T V_u]d\zeta. \end{cases} \tag{4.8}$$

Assuming that $V \in \mathbb{R}^{n\times\hat{r}}$ is chosen such that X^TV has rank r, (4.8) captures the exact invariant subspace of A associated with the eigenvalues located inside $[-1,1]$. Then, $Z \equiv \mathcal{P}V$ is used in a Rayleigh-Ritz projection to recover the actual eigenpairs of A. Because the above discussed scheme considers all blocks of $\mathcal{P}$, we will refer to it as "Domain Decomposition Full Projector" (DD-FP). In the following, we summarize the practical details of the DD-FP scheme.

4.1.1. Practical aspects of the DD-FP scheme. In practice, the integrals in (4.8) will have to be approximated numerically. Once a quadrature rule is selected, with quadrature nodes and weights $\{\zeta_j, \omega_j\},\ j = 1,\dots,N_c$, the integrals are approximated by the following summations (the scaling $-1/2i\pi$ is omitted):

$$\tilde{Z}_u = \sum_{j=1}^{N_c}\omega_j(B-\zeta_j I)^{-1}V_u - \sum_{j=1}^{N_c}\omega_j F(\zeta_j)S(\zeta_j)^{-1}[V_s - F(\zeta_j)^T V_u], \tag{4.9}$$

$$\tilde{Z}_s = \sum_{j=1}^{N_c}\omega_j S(\zeta_j)^{-1}[V_s - F(\zeta_j)^T V_u]. \tag{4.10}$$

Different quadrature rules can be used to perform the numerical integration, e.g., the Gauss-Legendre [32] or the trapezoidal [40] rules. Using a rule in which the quadrature nodes appear

in conjugate pairs, i.e., $\zeta_j = \bar{\zeta}_{j+N_c/2},\ j = 1, \ldots, N_c/2$, reduces the cost of the numerical approximation of the contour integral by a factor of two, since

$$B - \zeta_j I = \overline{B - \zeta_{j+N_c/2} I},\ \ S(\zeta_j) = \overline{S(\zeta_{j+N_c/2})},\quad j = 1, \ldots, N_c/2,$$

and thus numerical integration must be carried out only on one of the two semi-circles. For the rest of this paper, N_c will denote the number of quadrature nodes used to integrate along the positive semi-circle only. Viewing contour integration as a form of rational filtering, other integration rules become possible, e.g., Zolotarev rational filters [18], or least-squares filters [47], however, we do not explore these options in this paper.

For each quadrature node $\zeta_j,\ j = 1, \ldots, N_c$, and each column in V, we must solve two linear systems with $B - \zeta_j I$ and one linear system with $S(\zeta_j)$. The calculation takes four steps that accumulate the sums (4.9)-(4.10) into $\tilde{Z}_u,\ \tilde{Z}_s$, and is shown in Algorithm 4.1:

ALGORITHM 4.1. *DD-FP*

0. *Start with random* $V \in \mathbb{R}^{n \times \hat{r}}$ *and* $\tilde{Z}_s = \tilde{Z}_u = 0$
1. *Do until convergence*
2. *For* $j = 1, \ldots, N_c$*:*
3. $W_u := (B - \zeta_j I)^{-1} V_u$
4. $W_s := V_s - E^T W_u$
5. $W_s := S(\zeta_j)^{-1} W_s,$ $\tilde{Z}_s := \tilde{Z}_s + \Re e(\omega_j W_s)$
6. $W_u := W_u - (B - \zeta_j)^{-1} E W_s,$ $\tilde{Z}_u := \tilde{Z}_u + \Re e(\omega_j W_u)$
7. *End*
8. *Rayleigh-Ritz: solve the eigenvalue problem* $\tilde{Z}^T A \tilde{Z} Q = \tilde{Z}^T \tilde{Z} Q \Theta$
-. *If not satisfied, repeat with* $V_u = \tilde{Z}_u Q,\ V_s = \tilde{Z}_s Q$
9. *EndDo*

The factorization of each $B - \zeta_j I,\ j = 1, \ldots, N_c$ is decoupled into factorizations of the matrices $B_i - \zeta_j I,\ i = 1, \ldots, P$, each one being local to the i^{th} subdomain. Moreover, since A is symmetric, only the real parts of $\tilde{Z}_s$ and $\tilde{Z}_u$ have to be retained. The DD-FP scheme can be cast as an iterative scheme in which approximate eigenvectors are improved in a straightforward manner by using their most recent approximation as the new set V to multiply the approximate spectral projector $\mathcal{P}$. Step 8 of Algorithm 4.1 extracts the approximate eigenpairs and also checks whether all eigenpairs inside $[-1, 1]$ are approximated up to a sufficient accuracy (this part is omitted from the description of the algorithm).

If a direct solver is utilized to solve the linear systems with $S(\zeta_j),\ j = 1, \ldots, N_c$ then the DD-FP scheme is practically a straightforward application of the domain decomposition viewpoint applied to the computation of an approximation of $\mathcal{P}V$, and can be seen as equivalent to the FEAST algorithm tied with a domain decomposition solver to compute the products $(A - \zeta_j I)^{-1} V,\ j = 1, \ldots, N_c$. However, a factorization of $S(\zeta)$ is not always feasible (see Section 5). In such scenarios, the DD-FP scheme can leverage hybrid iterative solvers which might be more practical.

4.2. Partial integration of the matrix resolvent. In this section we describe an alternative scheme, also based on domain decomposition, which attempts to extract approximate eigenpairs at a lower cost than the DD-FP scheme developed in the previous section.

Let the spectral projector $\mathcal{P}$ defined in (4.5), be expressed in the form $\mathcal{P} = XX^T,\ X \in \mathbb{R}^{n \times r}$, where X is written as $X = [X_u^T,\ X_s^T]^T$ with $X_u \in \mathbb{R}^{d \times r},\ X_s \in \mathbb{R}^{s \times r}$. Then, $\mathcal{P}$ can be also expressed in a block-partitioned form:

$$X \equiv \begin{pmatrix} X_u \\ X_s \end{pmatrix}, P = XX^T \rightarrow \mathcal{P} = [\mathcal{P}_1, \mathcal{P}_2] = \begin{pmatrix} X_u X_u^T & X_u X_s^T \\ X_s X_u^T & X_s X_s^T \end{pmatrix}. \tag{4.11}$$

Under the mild assumption that $r \leq s$, i.e., the number of interface variables s is greater than the number of eigenvalues r of A lying inside $[-1, 1]$, the range of $\mathcal{P}$ can be captured by the range of $\mathcal{P}_2 = XX_s^T = [X_u^T,\ X_s^T]^T X_s^T$, since $\mathcal{P}_2$ is also of rank r and spans the same space as $\mathcal{P}$. By equating (4.11) with (4.5), it is clear that $X_s X_s^T \equiv \mathcal{G}$ and $X_u X_s^T \equiv -\mathcal{W}$, and thus, in contrast with the DD-FP scheme we only need to compute the contour integrals $-\mathcal{W}$ and $\mathcal{G}$, and ignore the block $\mathcal{H}$. As discussed in Section 4.3, and confirmed via experiments in Section 6, avoiding the computation of $\mathcal{H}$ can lead to considerable savings in some cases. Because this scheme approximates the spectral projector $\mathcal{P}$ only partially, we will refer to it as "Domain Decomposition Partial Projector" (DD-PP).

Further insight in the above scheme can be given if we consider the representation of $x(\lambda)$, the eigenvector of A corresponding to eigenvalue λ, in a domain decomposition framework. It can be easily shown that for any eigenvalue λ of A that does not belong to $\Lambda(B)$ (the spectrum of B), $S(\lambda)$ is singular and vice versa, i.e.,

$$\lambda \notin \Lambda(B), \quad \lambda \in \Lambda(A) \Leftrightarrow \det\left[S(\lambda)\right] = 0. \tag{4.12}$$

Let $y(\lambda) \in \mathbb{R}^s$ be the eigenvector associated with the zero eigenvalue of $S(\lambda)$. The eigenvector $x(\lambda)$ associated with eigenvalue λ can then be written as:

$$x(\lambda) = \begin{pmatrix} -F(\lambda)y(\lambda) \\ y(\lambda) \end{pmatrix}. \tag{4.13}$$

Comparing equations (4.11) and (4.13) shows that $\mathrm{span}(X_s) \equiv \mathrm{span}(y(\lambda_1), \ldots, y(\lambda_r))$ and $\mathrm{span}(X_u) \equiv \mathrm{span}(-F(\lambda_1)y(\lambda_1), \ldots, -F(\lambda_r)y(\lambda_r))$.

4.2.1. The DD-PP scheme. The range of $\mathcal{G}$ and $-\mathcal{W}$ can be approximated as:

$$\mathcal{G}R = \frac{-1}{2i\pi}\int_\Gamma S(\zeta)^{-1}Rd\zeta, \quad -\mathcal{W}R = \frac{1}{2i\pi}\int_\Gamma (B-\zeta I)^{-1}ES(\zeta)^{-1}Rd\zeta. \tag{4.14}$$

for any $R \in \mathbb{R}^{s\times\hat{r}},\ (\hat{r} \geq r)$ whose columns have a non-trivial projection along the direction of each eigenvector $y(\lambda_i),\ i = 1, \ldots, r$ of the nonlinear eigenvalue problem $S(\lambda)y(\lambda) = 0$.

In practice, both $\mathcal{G}$ and $-\mathcal{W}$ will be approximated numerically as in (4.15). Approximating the two integrals by a quadrature rule results in

$$\tilde{\mathcal{G}}R = \frac{-1}{2i\pi}\sum_{j=1}^{N_c}\omega_j S(\zeta_j)^{-1}R, \quad -\tilde{\mathcal{W}}R = \frac{1}{2i\pi}\sum_{j=1}^{N_c}\omega_j (B-\zeta_j I)^{-1}ES(\zeta_j)^{-1}R. \tag{4.15}$$

Combining the contribution of all quadrature nodes together, the final subspace accumulation proceeds as in Algorithm 4.2, which we abbreviate as DD-PP. Note that because we are actually interested in an approximation of an invariant subspace of X, the signs are not important (the signs in Steps 2 and 3 in Algorithm 4.2 could be reversed).

ALGORITHM 4.2. *DD-PP*

0. *Start with a random* $R \in \mathbb{R}^{s\times\hat{r}}$ *and* $\tilde{Z}_s = \tilde{Z}_u = 0$
1. *For* $j = 1, \ldots, N_c$*:*
2. $\quad W_s := S(\zeta_j)^{-1}R,$ $\qquad \tilde{Z}_s := \tilde{Z}_s + \Re e(\omega_j W_s)$
3. $\quad W_u := -(B-\zeta_j)^{-1}EW_s,$ $\qquad \tilde{Z}_u := \tilde{Z}_u + \Re e(\omega_j W_u)$
4. *End*
5. *Perform a Rayleigh-Ritz projection and extract approximate eigenpairs*

TABLE 4.1
Number of linear system solutions with $B - \zeta I$ and $S(\zeta)$, and Matrix-Vector multiplications with E/E^T performed by the DD-FP and DD-PP schemes per quadrature node.

Scheme/Operation	$B - \zeta I$	$S(\zeta)$	E/E^T
DD-FP	$2 \times \hat{r}$	$\hat{r}$	$2 \times \hat{r}$
DD-PP	$\hat{r}$	$\hat{r}$	$\hat{r}$

Because there is no straightforward way to improve the eigenvalue approximations produced by the DD-PP scheme, we have to either use a large number of quadrature nodes N_c to obtain a sufficiently accurate approximation for the spectral projector, or use a large value of $\hat{r}$.

Figure 4.1 shows the average residual norm of the approximate eigenpairs obtained by the DD-FP and DD-PP schemes for a small 2D discretized Laplacian of size $n = 51 \times 50$ in the interval $[\alpha = 1.6, \beta = 1.7]$ (more details on matrices of this form will be given in Section 6.2). Because of the iterative nature of the DD-FP scheme we can use a small number of

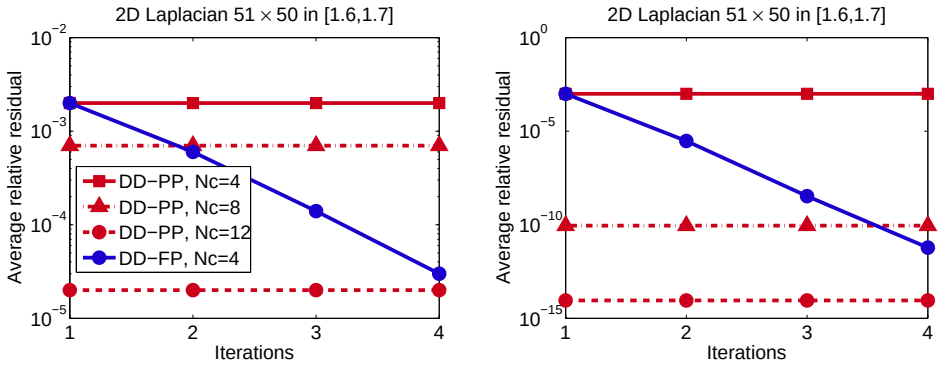

FIG. 4.1. *Average residual norm for a 51×50 2D Laplacian in the interval $[1.6, 1.7]$. Left: $\hat{r} = r$. Right: $\hat{r} = 2r$. The Gauss-Legendre quadrature rule was used [2].*

quadrature nodes and correct the approximate eigenpairs by repeating the numerical integration phase using the most recent approximate eigenvectors as the new set of right-hand sides. Indeed, after four iterations, the DD-FP scheme with $N_c = 4$ quadrature nodes achieves an accuracy similar to that of the DD-PP scheme utilizing $N_c = 12$ quadrature nodes.

4.3. Computational comparison of the DD-FP and DD-PP schemes. From a numerical viewpoint the DD-PP and DD-FP schemes can perform similarly but, from a computational viewpoint, there are some notable differences. Algorithm 4.2 has a lower computational complexity per quadrature node than Algorithm 4.1 since it avoids the first two steps of Algorithm 4.1. Table 4.1 shows the number of solves with matrices $B - \zeta I$ and $S(\zeta)$, as well as the number of Matrix-Vector operations with E/E^T introduced per quadrature node by each one of the two schemes. Furthermore, a straightforward calculation reveals that for each quadrature node, the DD-FP scheme also introduces $n \times \hat{r}$ more floating-point operations than the DD-PP scheme (the block matrix subtractions in Steps 3 and 5 in Algorithm 4.1). Accounting for all N_c quadrature nodes together, the DD-FP scheme introduces $N_c \times \hat{r} \times [cost\_solve(B - \zeta I) + cost\_MV(E) + n]$ additional floating-point operations compared to the DD-PP scheme. Here, $cost\_solve(B - \zeta I)$ and $cost\_MV(E)$ denote the costs to multiply $(B - \zeta I)^{-1}$ (by solving the linear system) and E/E^T by a single vector, respectively.

Given a distributed computing environment, in which each subdomain is assigned to a different processor group, the actual extra cost introduced by the i^{th} subdomain (processor) when using Algorithm 4.1 compared to Algorithm 4.2, amounts to $N_c \times \hat{r} \times [cost\_solve(B_i - \zeta I) + cost\_MV(E_i) + d_i]$. Thus, if d_i is large and/or many eigenvalues of A lie in $[-1, 1]$ (and thus $\hat{r}$ is large), the dense matrix operations in Steps 3 and 5 of Algorithm 4.1 become noticeable. The extra operations performed by Algorithm 4.1 are entirely local within each subdomain. On the other hand, as Algorithm 4.2 is a one-shot method, we need a rather accurate projector for the entire approximation to work well. This is not a constraint for Algorithm 4.1 since it is an iterative scheme.

5. Solving linear systems with the spectral Schur complement matrix. From a computational viewpoint, the major computational procedure in both Algorithm 4.1 and Algorithm 4.2, is the solution of linear systems with the Schur complement matrices $S(\zeta_j),\ j = 1, \ldots, N_c$, where each linear system has $\hat{r} \geq r$ right-hand sides.

Assuming that the computational domain is partitioned in P non-overlapping subdomains, with each subdomain assigned to a different processor, $S(\zeta_j)$ is distributed by rows among the different processors and has a natural block structure of the form

$$S(\zeta_j) = \begin{pmatrix} S_1(\zeta_j) & E_{12} & \ldots & E_{1P} \\ E_{21} & S_2(\zeta_j) & \ldots & E_{2P} \\ \vdots & \vdots & \ddots & \vdots \\ E_{P1} & E_{P2} & \ldots & S_P(\zeta_j) \end{pmatrix},\ j = 1, \ldots, N_c, \tag{5.1}$$

where

$$S_i(\zeta_j) = C_i - \zeta I - E_i^T (B_i - \zeta I)^{-1} E_i,\ i = 1, \ldots, P,$$

is the "local" Schur complement that corresponds to the i^{th} subdomain and is a dense matrix in general. The off-diagonal blocks $E_{ik},\ i, k = 1, \ldots, P,\ i \neq k$, account for the coupling among the different subdomains and are sparse matrices of size $s_i \times s_k$ (they are identical with those of the local system in (3.2)).

The standard approach to solve the distributed linear systems with the Schur complement in (5.1) would be to explicitly form $S(\zeta_j)$ and compute its LU factorization by a call to a parallel sparse direct solver, e.g., MUMPS [3] or SuperLU_DIST [25]. This approach, however, requires that the diagonal blocks $S_i(\zeta_j),\ i = 1, \ldots, P$ be formed explicitly. For problems issued from discretizations of 2D domains, forming and factorizing $S(\zeta_j)$ explicitly is an attractive option since the size of the Schur complement is small even for a large number of subdomains (the interface region between any two subdomains is a 1D object). Schur complements that originate from discretizations of 3D computational domains [34] typically require much more memory since in the 3D case the size of the Schur complement can become exceedingly large (the interface region now is a combination of planes).[1] An alternative discussed next is to solve the linear systems with $S(\zeta_j),\ j = 1, \ldots, N_c$ using a preconditioned iterative method (e.g., GMRES [37]). Iterative methods avoid forming $S(\zeta_j)$ explicitly and only require a routine that is able to accomplish the multiplication between $S(\zeta_j)$ and a vector (details will be given in Section 5.2).

[1] For example, the memory requirements to store $S(\zeta_j)$ when discretizing the Laplacian operator on the unit cube with $n^{1/3}$ discretization points along each direction scales as $O(n^{4/3})$ if nested dissection [16] is used to reorder $(A - \zeta_j I)$.

5.1. Preconditioning the Schur complement. Schur complement preconditioning relies on two procedures: a) approximation of $S(\zeta)$ by a matrix $\hat{S}(\zeta)$, and b) a mechanism to apply $\hat{S}(\zeta)^{-1}$ on a vector. One of the first specialized libraries to offer distributed Schur complement preconditioners is the pARMS library [26,38,39] which implements a multilevel partial elimination of the Schur complement. In this paper we consider sparsified approximations of $S(\zeta_j)$ which are based on sparsity and/or numerical constraints [11, 17, 33].

5.1.1. Building and applying the preconditioner. Since the i^{th} processor holds the i^{th} block of rows of $S(\zeta)$, a straightforward approach is to solve the linear systems in (5.1) by applying a block-Jacobi preconditioner, i.e., to utilize a preconditioner of the form:

$$S_{\mathtt{BJ}}(\zeta) \;=\; \begin{pmatrix} S_1(\zeta) & & & \\ & S_2(\zeta) & & \\ & & \ddots & \\ & & & S_P(\zeta) \end{pmatrix}. \tag{5.2}$$

The LU factorization of each on-diagonal block $S_i(\zeta),\ i = 1, \ldots, P$ can be obtained directly from the LU factorization of

$$A_i(\zeta) = \begin{pmatrix} B_i - \zeta I & \hat{E}_i \\ \hat{E}_i^T & C_i - \zeta I \end{pmatrix}, \tag{5.3}$$

which can be written as $A_i(\zeta) = L_{A_i} U_{A_i}$, with

$$L_{A_i} = \begin{pmatrix} L_{B_i} & 0 \\ \hat{E}_i^T U_{B_i}^{-1} & L_{S_i} \end{pmatrix}, \quad U_{A_i} = \begin{pmatrix} U_{B_i} & L_{B_i}^{-1}\hat{E}_i \\ 0 & U_{S_i} \end{pmatrix}, \tag{5.4}$$

and noticing that $S_i(\zeta) = L_{S_i} U_{S_i}$ [26, 35]. The block-Jacobi preconditioner is applied in a completely parallel fashion, with each processor performing a forward/backward triangular substitution with L_{S_i}/U_{S_i}.

Extending the block-Jacobi preconditioner we can take the coupling E_{ik}, $i, k = 1, \ldots, P,\ i \neq k$, among the different subdomains also into account, which then leads to a preconditioner $S_G(\zeta)$ that approximates $S(\zeta)$ more accurately. The matrix $S_G(\zeta)$ is distributed among the different groups of processors and thus communication among the processors is necessary when the preconditioner is applied. To form $S_G(\zeta)$ we use two levels of dropping based on numerical constraints. The first level of dropping concerns the LU factorization of $B - \zeta I$ which is performed inexactly, by dropping all entries in the LU factorization whose real or imaginary part is below a threshold value `drop-B`. Then, the i^{th} subdomain forms its local Schur complement

$$\hat{S}_i(\zeta) = C_i - \zeta I - (\hat{U}_i^{-T} E_i)^T (\hat{L}_i^{-1} E_i), \tag{5.5}$$

while dropping any entry whose real or imaginary part is below a threshold value `drop-S`. Matrices $\hat{L}_i$ and $\hat{U}_i$ denote the LU factors of the incomplete factorization of each $B_i - \zeta I,\ i = 1, \ldots, P$. Overall, the preconditioner takes the form:

$$S_G(\zeta) \;=\; \begin{pmatrix} \hat{S}_1(\zeta) & E_{12} & \ldots & E_{1P} \\ E_{21} & \hat{S}_2(\zeta) & \ldots & E_{2P} \\ \vdots & \vdots & \ddots & \vdots \\ E_{P1} & E_{P2} & \ldots & \hat{S}_P(\zeta) \end{pmatrix}, \tag{5.6}$$

where matrices E_{ik} are identical with those in (5.1). The construction of preconditioner $S_G(\zeta)$ is summarized in Algorithm 5.1.

ALGORITHM 5.1. *Schur complement preconditioner* $S_G(\zeta)$

0. Given $\zeta \in \mathbb{C}$, `drop-B, drop-S`
1. For $i = 1, \ldots, P$*:*
2. Obtain a factorization $[\hat{L}_i, \hat{U}_i] = B_i - \zeta I$ *with drop tolerance* `drop-B`
3. Form $\hat{S}_i(\zeta) = C_i - \zeta I - (\hat{U}_i^{-T} E_i)^T (\hat{L}_i^{-1} E_i)$ *and*
-. drop any entry smaller than `drop-S`
4. End
5. Factorize $S_G(\zeta)$ *by a distributed sparse solver.*

A few details regarding Algorithm 5.1. When forming $S_G(\zeta)$, we form $\hat{S}_i(\zeta)$ a few columns at a time and immediately sparsify (for each incomplete factorization of $B_i - \zeta I$ we must solve a linear system with s_i sparse right-hand sides). In this paper, by default we form $\hat{S}_i(\zeta)$ two hundred columns at a time, where all right-hand sides are solved simultaneously using the Pardiso software package [24, 30]. More details will be given in Section 6. Small values of `drop-B, drop-S` might generally lead to more robust preconditioners, but, on the other hand, will also introduce larger computational and memory overheads during the formation and factorization of $S_G(\zeta)$.

Before we conclude this section, we note that preconditioned iterative methods to solve the linear systems with matrices $S(\zeta_j),\ j = 1, \ldots, N_c$ can still be computationally expensive in certain cases. First, depending on the number of sought eigenpairs, we might have to solve systems with hundreds or thousands of right-hand sides for each $S(\zeta_j),\ j = 1, \ldots, N_c$. Second, using larger values for N_c will bring some of the quadrature nodes close to the real axis. As a result, the iterative solution scheme will typically be slower. We will quantify this behavior in Section 6.

5.2. Matrix-Vector products with $S(\zeta)$. To solve a linear system with matrix $S(\zeta)$ (here ζ can take any value), each iteration of a Krylov subspace preconditioned iterative solver requires at least one Matrix-Vector (MV) product with $S(\zeta)$ and linear system solution with the preconditioner matrix, as well as a few vector dot-products and vector updates.

The MV product between $S(\zeta)$ and a vector $v \in \mathbb{R}^s$ can be computed as:

$$S(\zeta)v = (C - \zeta I)v - E^T (B - \zeta I)^{-1} E v. \tag{5.7}$$

An important property from domain decomposition with edge-separators is that the second term on the right-hand side of (5.7) is entirely local, as can be easily seen from the structure of $S(\sigma)$ in (5.1). Matrices B_i and $E_i,\ i = 1, \ldots, P$ are entirely local in each processor and no communication is required when we perform operations with them. On the other hand, performing operations with C demands communication between processors which handle neighboring subdomains. In summary, the computations involved in (5.7) are:

1. Compute $E^T (B - \zeta I)^{-1} E v$ (local),
2. Distribute (exchange) the necessary parts of v and perform $(C - \zeta I)v$ (global),
3. Subtract the vector in 1) from the vector in 2) (local).

Communication in step 2) might overlap with computations in step 1). Using more subdomains (larger values for P) will reduce the computational cost per processor, but, on the other hand, will increase the communication cost. Each local solve with the block-diagonal matrix $B - \zeta I$ is carried out by using a sparse solver.

The linear system solution between the preconditioner and the current residual vector can be performed either by a direct or an iterative solver. In this paper we consider the

preconditioners, $S_{BJ}(\zeta)$ and $S_G(\zeta)$, discussed in Section 5.1.1, and we always apply them using a direct method.

6. Experiments. In this section we analyze the performance of the proposed domain decomposition schemes by reporting experiments performed in distributed computing environments. The numerical schemes discussed were implemented in C/C++ and built on top of the PETSc [6–8] and Intel Math Kernel scientific libraries [1]. For PETSc, we used a complex build.[2] The source files were compiled with the Intel MPI compiler `mpiicpc`, using the -O3 optimization level. The computational domain was partitioned in P non-overlapping subdomains with the help of the METIS graph partitioner [21]. Each subdomain was assigned to a distinct processor group and communication between different processor groups was achieved by means of the Message Passing Interface standard (MPI) [44]. Each subdomain was handled by a separate MPI process and the number of subdomains, P, will be also denoting the number of MPI processes. The LU factorizations and linear system solutions associated with matrices $B - \zeta_j I,\ j = 1, \ldots, N_c$, were performed using the shared-memory, multi-threaded version of the Pardiso library (version 5.0.0) [24,30]. Unless stated otherwise, the default number of threads per MPI process, as denoted by variable T, will be equal to one. Whenever we computed an incomplete factorization of matrices $B-\zeta_j I,\ j = 1, \ldots, N_c$, that was obtained by the UMFPACK [12] library,[3] and the resulting triangular factors were then passed to Pardiso. We followed this approach in order to take advantage of the multi-threading capability of Pardiso, as well as of the fact that Pardiso has the ability to solve linear systems with multiple right-hand sides simultaneously.

The quadrature nodes and weights $\zeta_j,\ \omega_j,\ j = 1, \ldots, N_c$ used for the numerical approximation of the contour integrals were computed by the Gauss-Legendre quadrature rule of order $2 \cdot N_c$ [2], retaining only the N_c quadrature nodes (and their associated weights) with positive imaginary part.

Since the multiple right-hand sides for each linear system solution are available at the same time, it is possible to utilize block Krylov subspace solvers [19], e.g., block GMRES [42]. While we explored this option, using our custom implementation of block GMRES, we do not report results using block Krylov subspace methods in this paper. Throughout the rest of this section, the multiple right-hand sides are solved one after the other.

6.1. Computational system. The experiments performed at the `Mesabi` Linux cluster at Minnesota Supercomputing Institute. `Mesabi` consists by 741 nodes of various configurations with a total of 17,784 compute cores provided by Intel Haswell E5-2680v3 processors. Each node features two sockets, each socket with twelve physical cores at 2.5 GHz. Each node is also equipped with 64 GB of system memory. In total, `Mesabi` features a peak performance of 711 Tflop/s and 67 TB of memory.

Each MPI process will be paired with a single socket of each `Mesabi` node.

6.2. The model problem. The model problem test matrices originate from discretizations of elliptic PDEs on 2D and 3D computational domains. More specifically, we are interested in solving the eigenvalue problem

$$-\Delta u = \lambda u \tag{6.1}$$

on a rectangular domain, with Dirichlet boundary conditions (Δ denotes the Laplacian differential operator). Using second order centered finite differences with $n_x,\ n_y$ and n_z discretization points along each corresponding dimension, we obtain matrix A, the discretized version of Δ, of size $n = n_x n_y n_z$.

[2] The complex version of PETSc was built using the option `-with-fortran-kernels=generic`

[3] Using the routines `umfpack_zi_XXX`

TABLE 6.1

Average time spent on a single quadrature node using the DD-PP and DD-FP schemes to approximate the eigenvalues $\lambda_{1001}, \ldots, \lambda_{1200}$ and associated eigenvectors for three discretized 2D Laplacians. The number of right-hand sides $\hat{r}$ as well as the number of subdomains P were varied.

	$P = 16$		$P = 32$		$P = 64$	
	DD-PP	DD-FP	DD-PP	DD-FP	DD-PP	DD-FP
$n = 500^2$						
$\hat{r} = r + 10$	12.6	14.7	8.13	9.20	6.62	7.25
$\hat{r} = 3r/2 + 10$	15.5	18.2	12.1	13.0	8.87	9.77
$\hat{r} = 2r + 10$	18.7	22.9	13.3	15.3	11.4	12.6
$n = 1000^2$						
$\hat{r} = r + 10$	75.2	85.7	43.5	48.7	26.7	30.9
$\hat{r} = 3r/2 + 10$	86.0	101.1	51.1	58.6	33.0	39.1
$\hat{r} = 2r + 10$	98.1	118.8	59.6	69.9	40.4	48.5
$n = 1500^2$						
$\hat{r} = r + 10$	267.1	290.2	116.9	121.1	66.4	74.8
$\hat{r} = 3r/2 + 10$	295.0	328.6	134.1	140.4	79.7	91.9
$\hat{r} = 2r + 10$	326.9	370.0	153.3	161.5	93.4	109.1

6.3. A comparison of the DD-FP and DD-PP schemes. We begin by reporting a comparison of the DD-FP and DD-PP schemes on a set of discretized 2D Laplacian matrices, where the Schur complement matrices $S(\zeta_j),\ j = 1, \ldots, N_c$ were formed and factorized explicitly (`drop-B=drop-S=1e-16`) by MUMPS. In order to perform a fair comparison between the two schemes, only one iteration of the DD-FP scheme was allowed.

We used three different discretizations of the 2D Laplacian operator on the unit plane to obtain three different matrices of size $n = 500^2,\ n = 1000^2$, and $n = 1500^2$. The interval of interest was arbitrarily set to $[\alpha, \beta] = [(\lambda_{1000} + \lambda_{1001})/2, (\lambda_{1200} + \lambda_{1201})/2]$, including $r = 200$ eigenvalues. We used $N_c = 4,\ N_c = 8$ and $N_c = 12$ quadrature nodes, while the number of right-hand sides, $\hat{r}$, was varied. Table 6.1 reports the average wall-clock time spent on a single quadrature node for the case $N_c = 8$. By the term total wall-clock timing we mean the total timings to perform all factorizations and linear system solutions, as well to perform the Rayleigh-Ritz projections and check convergence. Per quadrature node timings for the other choices of N_c were basically identical. The DD-PP scheme was always faster than the DD-FP scheme, especially as P and $\hat{r}$ obtained smaller and larger values, respectively. The above lies in agreement with the discussion in Section 4.3.

Figure 6.1 plots the maximum (dashed) and average (solid) residual norm of the approximate eigenpairs for all different combinations of N_c and $\hat{r}$ reported in Table 6.1. The relative residual errors were of the same order for both the DD-FP and DD-PP schemes and we report results only for the DD-PP scheme. The `x`-axis runs across the different number of right-hand sides $\hat{r}$ while curves with different markers represent the different number of quadrature nodes used and are indexed as "●": $N_c = 4$, "▲": $N_c = 8$, "■": $N_c = 12$. Naturally, as we increase N_c and/or $\hat{r}$, the accuracy of the approximate eigenpairs improves.

The last experiment of this section discusses the case where the DD-FP scheme is allowed to perform more than one outer iterations, until all eigenpairs of the $n = 1500^2$ Laplacian located inside the interval $[\alpha, \beta] = [(\lambda_{1000}+\lambda_{1001})/2, (\lambda_{1200}+\lambda_{1201})/2]$ are approximated to at least eight digits of accuracy each. We compared the DD-FP scheme against a PETSc-based

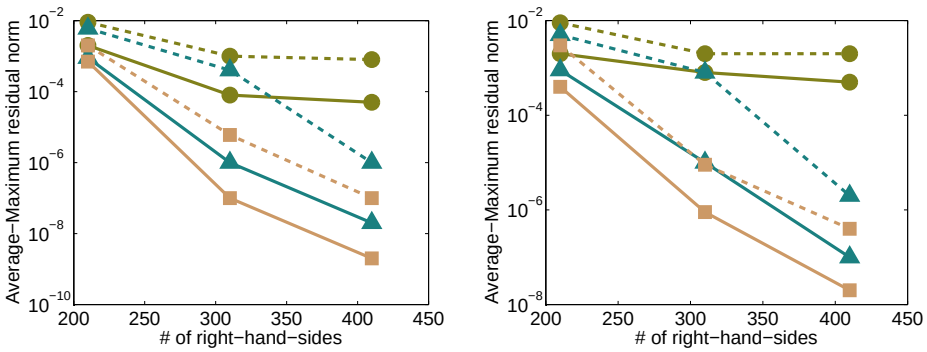

FIG. 6.1. *Maximum (dashed) and average (solid) residual norm of the eigenpairs inside the interval* $[\alpha, \beta] = [(\lambda_{1000} + \lambda_{1001})/2, (\lambda_{1200} + \lambda_{1201})/2]$. *Left:* $n = 500^2$. *Right:* $n = 1000^2$. *Legend: "●":* $N_c = 4$, *"▲":* $N_c = 8$, *"■":* $N_c = 12$.

TABLE 6.2

Time elapsed to compute eigenvalues $\lambda_{1001}, \ldots, \lambda_{1200}$ *and corresponding eigenvectors of the* $n = 1500^2$ *Laplacian by the CI-M and DD-FP schemes, using different choices of* N_c *and* $\hat{r}$. *"Its" denotes the number of outer iterations.*

		$P = 64$		$P = 128$		$P = 256$	
	Its	CI-M	DD-FP	CI-M	DD-FP	CI-M	DD-FP
$N_c = 2$							
$\hat{r} = 3r/2$	9	3,922.7	2,280.6	2,624.3	1,242.4	1,911.2	859.5
$\hat{r} = 2r$	5	2,863.2	1,764.5	1,877.7	998.5	1,255.5	615.3
$N_c = 4$							
$\hat{r} = 3r/2$	5	4,181.5	2,357.0	2,815.7	1,280.2	1,874.1	877.5
$\hat{r} = 2r$	4	4,330.3	2,571.4	2,869.5	1,462.9	2,023.2	1,036.2
$N_c = 6$							
$\hat{r} = 3r/2$	3	3,710.3	2,068.2	2,504.1	1,122.1	1,790.8	766.5
$\hat{r} = 2r$	3	4,774.8	2,798.5	3,177.7	1,595.2	2,743.6	1,125.1
$N_c = 8$							
$\hat{r} = 3r/2$	3	4,911.6	2,722.2	3,318.7	1,476.1	2,367.7	1,006.5
$\hat{r} = 2r$	2	4,204.7	2,445.2	2,802.1	1,395.4	1,806,6	982.1

implementation of the FEAST algorithm, referred to as Contour Integration-MUMPS (CI-M), which utilized MUMPS to factorize and solve the linear systems with matrices $A - \zeta_j I,\ j = 1, \ldots, N_c$ (not a domain decomposition approach).

Table 6.2 reports the total wall-clock time to compute eigenvalues $\lambda_{1001}, \ldots, \lambda_{1200}$ and associated eigenvectors for the $n = 1500^2$ 2D discretized Laplacian, by both the CI-M and DD-FP schemes, when different values of N_c and $\hat{r}$ are used. Variable "Its" denotes the number of outer iterations (same in both schemes). As expected, using higher values for N_c generally results to fewer outer iterations. However, this does not necessarily lead to lower runtimes, since increasing the value of N_c does not generally lead to a great reduction in the number of outer iterations for fixed values of $\hat{r}$. Similarly, increasing $\hat{r}$ after a certain value does not affect the number of outer iterations much, thus leading to increased runtimes. The performance gap between the DD-FP and CI-M schemes follows a slightly increasing trend

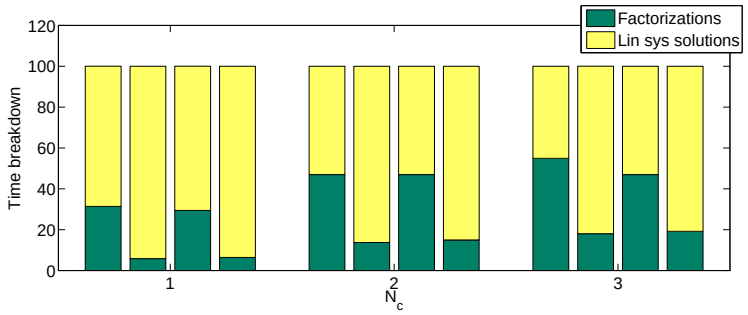

FIG. 6.2. *Time breakdown of the CI-M scheme (time spent on factorizations and linear system solutions) for* $N_c = 1,\ N_c = 2$ *and* $N_c = 3$*, using the optimal choice of* $\hat{r} := \hat{r}^*$ *for each case. Results are shown for* $P = 128$*. For each choice of* N_c*, we show the breakdown for intervals* $[\alpha, \beta] = [(\lambda_{100} + \lambda_{101})/2, (\lambda_{120} + \lambda_{121})/2]$ *(first-leftmost spike),* $[\alpha, \beta] = [(\lambda_{100}+\lambda_{101})/2, (\lambda_{200}+\lambda_{201})/2]$ *(second spike),* $[\alpha, \beta] = [(\lambda_{500}+\lambda_{501})/2, (\lambda_{520}+\lambda_{521})/2]$ *(third spike), and* $[\alpha, \beta] = [(\lambda_{500} + \lambda_{501})/2, (\lambda_{600} + \lambda_{601})/2]$ *(fourth-rightmost spike).*

as larger values of P are used, mainly because the linear system solution phase scales better for the DD-FP scheme than what for the CI-M scheme.

6.4. Contour integration using preconditioned iterative solvers. In this section we consider the solution of eigenvalue problems for which a direct formation and factorization of the Schur complement matrices $S(\zeta_j),\ j = 1, \ldots, N_c$ is rather expensive or impractical, e.g., discretizations of 3D or higher-dimensional computational domains. The alternative is to solve the linear systems with the Schur complement matrices $S(\zeta_j),\ j = 1, \ldots, N_c$ by a preconditioned iterative solver.

In contrast with Section 6.3, the solution of linear systems with the Schur complement matrices far dominates the computational procedure, and thus the DD-FP and DD-PP schemes in practice share the same computational profile. For this reason, we only compare the DD-FP and CI-M schemes.

For the rest of this section, the preconditioned iterative solver of choice will be the right preconditioned GMRES(250), i.e., we allowed 250 preconditioned iterations per each restart. A linear system will be considered solved after its initial residual norm gets reduced by at least ten orders of magnitude.

6.4.1. A 3D model problem. In this section we consider the solution of an eigenvalue problem where A originates from a discretization of the Laplacian operator on the unit cube, using 150 discretization points along each corresponding dimension, i.e., $n = 150 \times 150 \times 150$. We utilized the DD-FP and CI-M schemes to compute the lowest r=20 and r=100 eigenvalues (and associated eigenvectors) located inside the intervals: $[\alpha, \beta] = [(\lambda_{100} + \lambda_{101})/2, (\lambda_{200} + \lambda_{201})/2]$, and $[\alpha, \beta] = [(\lambda_{500} + \lambda_{501})/2, (\lambda_{600} + \lambda_{601})/2]$.

We first comment on the results obtained by the CI-M scheme, which, as noted earlier, essentially is the FEAST algorithm tied with MUMPS to factorize and solve linear systems with $A - \zeta_j,\ j = 1, \ldots, N_c$. The results are shown in tables located in the appendix section. Table .1 shows the total elapsed time to compute the eigenpairs associated with eigenvalues $\lambda_{101}, \ldots, \lambda_{120}$ and $\lambda_{501}, \ldots, \lambda_{520}$, as the number of quadrature nodes N_c and size of subspace $\hat{r}$ are varied. The optimal value of $\hat{r}$ in terms of the total number of right-hand sides solved is shown as $\hat{r}^*$. Table .2 conveys the same information for the eigenpairs associated with eigenvalues $\lambda_{101}, \ldots, \lambda_{200}$ and $\lambda_{501}, \ldots, \lambda_{600}$. Figure 6.2 shows the time breakdown of the CI-M scheme, focusing on the two main computational procedures, i.e., time spent on factorizations and linear system solutions, for all different choices of N_c and different

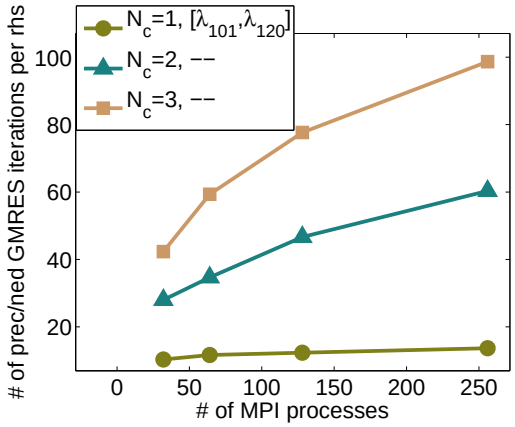

FIG. 6.3. *Total number of preconditioned GMRES iterations in order to solve a linear system with a single right-hand side for all* $N_c = 1,\ N_c = 2$ *and* $N_c = 3$ *quadrature nodes, when* $P = 32,\ P = 64,\ P = 128$ *and* $P = 256$ *subdomains are used.*

intervals $[\alpha, \beta]$ tested. Results are shown only for the optimal choice of $\hat{r},\ \hat{r}^*$, and for $P = 128$ MPI processes. For the subintervals that contain only $r = 20$ eigenvalues, factorizing $A - \zeta_j I,\ j = 1, \ldots, N_c$ introduces a large overhead, especially for larger values of N_c (the average factorization time per quadrature node was 679.02, 332.11, and 298.43 seconds, for $P = 64,\ 128$ and $P = 256$ MPI processes, respectively). On the other hand, when many eigenvalues lie within the interval of interest, we expect the time spent on solving linear systems to dominate (unless high values of N_c are chosen).

An alternative, which avoids the need to factorize matrices $A - \zeta_j,\ j = 1, \ldots, N_c$ (and thus their excessive requirements of memory and factorization time), is to exploit the DD-FP scheme tied with a preconditioned iterative solver to solve the linear systems associated with matrices $S(\zeta_j),\ j = 1, \ldots, N_c$. However, this approach requires some caution since *under the assumption that a factorization of the matrix is already at hand, the cost to solve a linear system by using a preconditioned iterative solver can be much higher than the cost to solve the same system by a direct solver*. We expect preconditioned iterative solvers to be a better alternative than the direct solvers (when a factorization is possible) when the number r of eigenvalues sought is not high and convergence of each linear system is rapid. As a brief numerical illustration, Figure 6.3 shows the total number of preconditioned GMRES iterations to compute $\sum_{j=1}^{N_c} S(\zeta_j)^{-1} v$ for a random $v \in \mathbb{C}^s$, if $P = 32,\ P = 64,\ P = 128$ and $P = 256$ subdomains are used (details on the preconditioner will be given later in this section). The interval of interest was set to $[\alpha, \beta] = [(\lambda_{100} + \lambda_{101})/2, (\lambda_{120} + \lambda_{121})/2]$. Note the superlinear increase in the number of preconditioned GMRES iterations as N_c increases, as well as the increase in the number of iterations as larger values of P are used. Iterative solvers are greatly affected by the location of the quadrature nodes $\zeta_j,\ j = 1, \ldots, N_c$, with ζ_j's which lie closer to the real axis leading to slower convergence. By construction, higher values of N_c will lead to some quadrature nodes being closer to the real axis, and thus using low values for N_c, e.g. $N_c = 1$, might in practice be a good alternative when direct solvers are impractical.

In contrast with the discretization of 2D domains, for 3D domains, the standard approach where each subdomain is handled by a single MPI process, each utilizing a single compute core, will not scale satisfactorily, since increasing the number of cores implies a larger number of subdomains, leading to an increase in the size of the Schur complement matrices. An

TABLE 6.3

Time spent on different phases when computing $\sum_{j=1}^{N_c} S(\zeta_j)^{-1}v$ *for a random* $v \in \mathbb{C}^s$. *We used* $N_c = 1$ *while the interval of interest was set to* $[\alpha, \beta] = [(\lambda_{100} + \lambda_{101})/2, (\lambda_{200} + \lambda_{201})/2]$. T: *number of threads per MPI process.*

	$P \times T = 32$	$P \times T = 64$		$P \times T = 128$		$P \times T = 256$	
	$T = 1$	$T = 1$	$T = 2$	$T = 1$	$T = 4$	$T = 1$	$T = 8$
MV with $S(\zeta_1)$	1.03	0.38	0.45	0.12	0.27	0.04	0.23
Factorization of $S_G(\zeta_1)$	3.20	5.01	3.20	7.23	3.20	9.06	3.20
Application of $S_G(\zeta_1)$	0.28	0.47	0.28	0.58	0.28	0.89	0.28

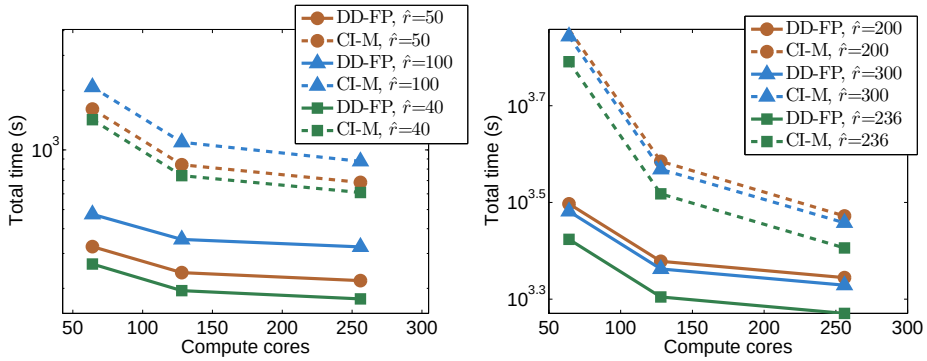

FIG. 6.4. *Total elapsed time to compute a few eigenpairs by the DD-FP and CI-M schemes using* $N_c = 1$ *and varying values of* $\hat{r}$. *Left: Computation of eigenvalues* $\lambda_{101}, \ldots, \lambda_{120}$ *and associated eigenvectors. Right: Computation of eigenvalues* $\lambda_{101}, \ldots, \lambda_{200}$ *and associated eigenvectors.*

alternative is to increase (or redistribute) the available compute cores so that each MPI process utilizes more than one compute cores. In the context of the DD-FP scheme, we could use lower values for P and increase the number of available compute threads T in Pardiso during the factorization and linear system solutions with matrices $B - \zeta_j I,\ j = 1, \ldots, N_c$. Table 6.3 shows a comparison of the pure MPI and hybrid (MPI+Threads) parallel paradigms when computing $\sum_{j=1}^{N_c} S(\zeta_j)^{-1}v,\ N_c = 1$, with $v \in \mathbb{C}^s$ a random vector, while the interval of interest was set to $[\alpha, \beta] = [(\lambda_{100} + \lambda_{101})/2, (\lambda_{200} + \lambda_{201})/2]$. We report the amount of time spent on performing the MV products with $S(\zeta_j)$, factorizing $S_G(\zeta_j)$, and applying $S_G(\zeta_j)$. For the preconditioner $S_G(\zeta_j)$ we used $\texttt{dropB} = \texttt{1e} - \texttt{4}, \texttt{dropS} = \texttt{1e} - \texttt{2}$. Assuming a fixed number of available compute cores, the standard approach of assigning one compute core per subdomain (pure MPI) is preferable when performing the MV products with $S(\zeta_j)$, since factorizing and solving linear systems with $B - \zeta_j$ in this case is faster, and also more scalable, than performing the same operation using fewer MPI processes, each with more than one compute threads. However, the main advantage of the hybrid parallel scheme (MPI+Threads) is that *the cost to apply the preconditioner does not increase as we increase the number of compute cores*. Table 6.3 shows that when 256 compute cores are available, using 32 MPI processes, each with eight threads, is about five times faster than using 256 MPI processes, each with a single thread.

In the following we focus only on the case $N_c = 1$, which proved to be the most competitive choice for the DD-FP scheme when a preconditioned iterative solver was used. The number of subdomains in the DD-FP scheme was fixed to $P = 32$.

Figure 6.4 compares the DD-FP and CI-M schemes when computing all eigenvalues and associated eigenvectors in the intervals $[\alpha, \beta] = [(\lambda_{100} + \lambda_{101})/2, (\lambda_{120} + \lambda_{121})/2]$ (left), and $[\alpha, \beta] = [(\lambda_{100} + \lambda_{101})/2, (\lambda_{200} + \lambda_{201})/2]$ (right). The results for the CI-M scheme were

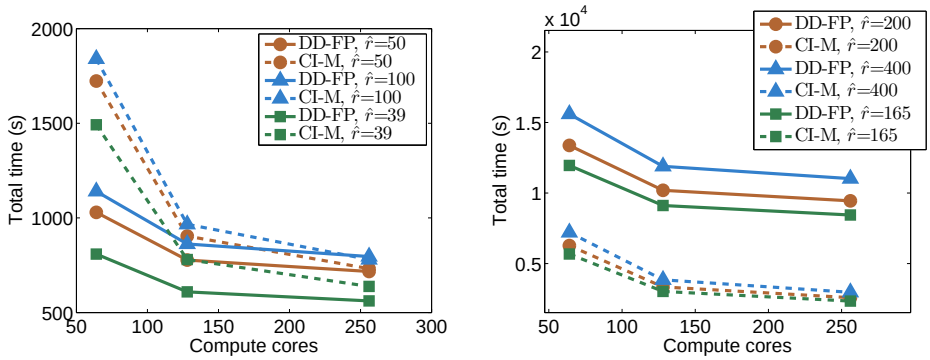

FIG. 6.5. *Total elapsed time to compute a few eigenpairs by the DD-FP and CI-M schemes using* $N_c = 1$ *and varying values of* $\hat{r}$*. Left: Computation of eigenvalues* $\lambda_{501}, \ldots, \lambda_{520}$ *and associated eigenvectors. Right: Computation of eigenvalues* $\lambda_{501}, \ldots, \lambda_{600}$ *and associated eigenvectors.*

extracted from Tables .1 and .2. For this experiment, the DD-FP scheme is faster than the CI-M scheme since it avoids the large overhead introduced by factorizing $(A - \zeta_1 I)$, while the preconditioned GMRES solver converges reasonably fast.

Figure 6.5 compares the DD-FP and CI-M schemes when computing all eigenvalues and associated eigenvectors in the intervals $[\alpha, \beta] = [(\lambda_{500} + \lambda_{501})/2, (\lambda_{520} + \lambda_{521})/2]$ (left), and $[\alpha, \beta] = [(\lambda_{500} + \lambda_{501})/2, (\lambda_{600} + \lambda_{601})/2]$ (right). This eigenvalue problem becomes tougher for the DD-FP scheme since the solution of each right-hand side takes much more time than in the previous case. For the case where $r = 20$ the DD-FP scheme is generally faster because it avoids the large overhead introduced by the expensive matrix factorizations by the CI-M scheme. However, for the case where $r = 100$, many right-hand sides must be solved and the CI-M scheme is much faster since the average time to solve each right-hand side is much lower than that in the DD-FP scheme.

Note that the results for the DD-FP scheme (especially its scalability) can improve significantly since the additional compute cores per MPI process used by Pardiso can be also utilized by MUMPS when applying the preconditioner, however, for reasons of fair comparison against the CI-M scheme, for MUMPS we use only MPI parallelism, and thus only a fraction of the available compute cores are active when we apply the preconditioner $S_G(\zeta_j)$.

6.4.2. The PARSEC matrix collection. Our third set of experiments originates from applications in Electronic Structure Calculations using the Density Functional Theory in real space. The matrices (Hamiltonians) were generated using the PARSEC software package [23], and can be found in the University of Florida Sparse Matrix Collection [13].[4] The matrices of this collection are real, symmetric, and have clustered eigenvalues. Table 6.4 lists the size n, the total number of non-zero entries nnz, as well as the interval of interest $[\alpha, \beta]$.

The number of nonzero entries of each Hamiltonian is quite large, a consequence of the high-order discretization and the addition of a (dense) 'non-local' term. Together with the 3D nature of the problem this leads to a large interface region, even if a good partitioning is employed. Before we continue, it is important to stress that contour integration approaches are not effective for this type of problems, and, in practice, polynomial filtering techniques have often been found to perform better [5, 15]. Thus, in this section, we do not compute any actual eigenpairs of the matrices in Table 6.4. We include this set of experiments with the purpose of demonstrating how preconditioned iterative solves can reduce the cost of contour

[4] https://www.cise.ufl.edu/research/sparse/matrices/

TABLE 6.4

Test matrices obtained by the PARSEC collection. We list the size n, the total number of non-zero entries nnz, as well as the interval of interest $[\alpha, \beta]$ and the number of eigenvalues r located inside $[\alpha, \beta]$.

Matrix	n	nnz	$[\alpha, \beta]$
$Ge_{99}H_{100}$	112,985	8,451,295	$[-0.65, -0.0096]$
$Si_{41}Ge_{41}H_{72}$	185,639	15,011,265	$[-0.64, -0.00282]$
$Si_{87}H_{76}$	240,369	10,661,631	$[-0.6600, -0.030]$

TABLE 6.5

Elapsed time for performing the N_c factorizations of the form $A - \zeta_j I,\ j = 1, \ldots, N_c$ (CI-M) versus elapsed time to compute the factorization of each matrix $A_i(\zeta_j),\ i = 1, \ldots, P$ (DD-FP). The same experiment was repeated for different values of P.

	$P = 4$		$P = 8$		$P = 16$		$P = 32$	
	CI-M	DD-FP	CI-M	DD-FP	CI-M	DD-FP	CI-M	DD-FP
$Ge_{99}H_{100}$								
$N_c = 1$	424.1	27.9	362.8	5.1	155.9	1.36	80.2	0.51
$N_c = 2$	860.9	56.4	714.2	10.7	308.7	2.57	162.3	0.92
$N_c = 3$	1265.9	86.3	1089.4	15.5	461.1	4.26	239.5	1.47
$Si_{41}Ge_{41}H_{72}$								
$N_c = 1$	1276.1	38.8	942.6	10.1	486.1	3.31	230.2	1.52
$N_c = 2$	X	74.5	1888.1	19.8	969.3	6.46	452.7	2.81
$N_c = 3$	X	117.4	X	28.8	1442.5	10.0	691.3	4.40
$Si_{87}H_{76}$								
$N_c = 1$	X	119.8	1726.2	14.5	942.4	1.23	382.1	0.51
$N_c = 2$	X	247.4	X	29.7	1872.8	2.53	758.0	0.94
$N_c = 3$	X	355.1	X	44.6	2853.9	3.82	1127.4	1.61

integration approaches in multi-core architectures and increase their practicability.

For this set of experiments we utilized a block-Jacobi preconditioner,

$$S_{BJ}(\zeta_j) = \texttt{bdiag}(S_1(\zeta_j), \ldots, S_P(\zeta_j)), j = 1, \ldots, N_c. \tag{6.2}$$

Factorizing a distributed approximation of each different Schur complement matrix $S(\zeta_j)$, $j = 1$,...,N_c (for any value of `drop-B` and `drop-S`) introduced a very large computational and memory overhead since it essentially amounts to factorizing matrix $A - \zeta_j I,\ j = 1, \ldots, N_c$, rendering the class of distributed preconditioners largely inefficient.

Table 6.5 shows the time elapsed to perform all N_c factorizations of the form $A - \zeta_j I,\ j = 1, \ldots, N_c$ (CI-M) versus the elapsed time to factorize matrices $A_i(\zeta_j),\ i = 1, \ldots, P$, a pre-process step of the DD-FP scheme (matrices $A_i(\zeta_j)$ are defined in (5.4)). As was discussed in section 5.1.1, the factorization of $A_i(\zeta_j)$ provides the factorization of both $B_i - \zeta_j I$ and $S_i(\zeta_j),\ i = 1, \ldots, P$. We report timings obtained for a varying number of P MPI processes (subdomains). A "X" flag under the CI-M scheme implies that not all N_c factorizations could fit in the memory allocated by each MPI process. We can observe the excessive timings when factorizing matrices $A - \zeta_j I,\ j = 1, \ldots, N_c$.

Table 6.6 shows the elapsed time to solve all N_c linear systems by the CI-M and DD-FP schemes for a random right-hand side $v \in \mathbb{R}^n$, i.e, $\sum_{j=1}^{N_c} (A - \zeta_j I)^{-1} v$. For lower values of P the DD-FP scheme is not a competitive approach, since the cost to apply the block-Jacobi preconditioner is quite high in that case. However, as P increases, the cost to solve a linear system with a single right-hand side, for all N_c quadrature nodes, drops dramatically (the number of iterations is only slightly increased as P increases). Increasing the value of N_c

TABLE 6.6
Time elapsed to perform the computation $\sum_{j=1}^{N_c}(A-\zeta_j I)^{-1}v$ with (DD-FP) and without (CI-M) using the domain decomposition framework. Vector $v \in \mathbb{R}^n$ denotes a random real vector.

	$P=4$		$P=8$		$P=16$		$P=32$	
	CI-M	DD-FP	CI-M	DD-FP	CI-M	DD-FP	CI-M	DD-FP
$Ge_{99}H_{100}$								
$N_c=1$	0.73	5.11	0.66	1.71	0.36	0.62	0.34	0.26
$N_c=2$	1.52	13.1	1.38	3.20	0.78	1.72	0.66	0.53
$N_c=3$	2.27	33.3	1.92	11.8	1.05	4.14	0.98	1.23
$Si_{41}Ge_{41}H_{72}$								
$N_c=1$	1.80	7.50	1.22	3.72	0.69	1.02	0.66	0.51
$N_c=2$	X	32.8	2.53	12.1	1.41	3.53	1.38	0.84
$N_c=3$	X	61.3	X	31.2	2.12	8.61	2.05	2.12
$Si_{87}H_{76}$								
$N_c=1$	X	15.0	1.59	4.33	1.29	0.92	0.90	0.41
$N_c=2$	X	50.2	X	14.0	2.75	3.34	1.88	0.76
$N_c=3$	X	120.5	X	34.8	4.02	7.51	2.65	1.96

results in a proportional increase in computational time for the direct solver but to a much more pronounced increase for the case of preconditioned iterative solvers, owing to the fact that iterative solvers are sensitive to the magnitude of the complex part of each quadrature node. Shifting from $N_c=1$ to $N_c=3$ leads to quadrature nodes $\zeta_1,\ \zeta_2$ and ζ_3, where ζ_1 and ζ_3 have a much smaller imaginary part than ζ_2. As a result, the iterative scheme to solve the Schur complement systems becomes much slower. On the other hand, the solution phase in the DD-FP scheme scales better than the same phase in the CI-M scheme.

7. Conclusion. In this paper we studied contour integration methods for computing eigenvalues and eigenvectors of sparse matrices using a domain decomposition viewpoint. We discussed two different numerical schemes. The first scheme, abbreviated as DD-FP, is basically a flexible implementation of the domain decomposition framework in the context of contour integral-based methods. It is essentially equivalent to a FEAST approach in which domain decomposition-based direct solvers are employed for the solution of the complex linear systems arising from the numerical integration. The second scheme, abbreviated as DD-PP, focuses on approximating the contour integrals only partially by integrating the Schur complement operator along the complex contour. Moreover, we considered the use of domain decomposition in the context of preconditioned iterative solvers as a replacement of the direct solvers. Experiments indicate that this approach can potentially be faster, but that its ultimate effectiveness will be dictated by the performance of the iterative scheme used for solving the linear systems. In particular, the method can be vastly superior when computing eigenvalues on both ends of the spectrum but it may encounter difficulties when the eigenvalues to be computed are located deep inside the spectrum.

Future work includes the incorporation of the block GMRES solver developed for the solution of the complex linear systems with the Schur complement, as well as an implementation of the DD-FP and DD-PP schemes which utilizes more than two levels of parallelism. The numerical methods presented in this paper will become available in the future update release of the FEAST software package.

Acknowledgments. We are grateful to the University of Minnesota Supercomputing Institute for providing us with computational resources to perform our experiments. We thank Ruipeng Li and Yuanzhe Xi for fruitful discussions. We would also like to thank the PETSc

team for providing assistance with the PETSc library, as well as Olaf Schenk and Radim Janalik for their help with the Pardiso library.

REFERENCES

[1] *Intel(r) fortran compiler xe 14.0 for linux*.

[2] MILTON ABRAMOWITZ, *Handbook of Mathematical Functions, With Formulas, Graphs, and Mathematical Tables,*, Dover Publications, Incorporated, 1974.

[3] PATRICK R. AMESTOY, IAIN S. DUFF, JEAN-YVES L'EXCELLENT, AND JACKO KOSTER, *A fully asynchronous multifrontal solver using distributed dynamic scheduling*, SIAM Journal on Matrix Analysis and Applications, 23 (2001), pp. 15–41.

[4] ALEXANDER L. SKOROKHODOV ANDREW V. KNYAZEV, *Preconditioned gradient-type iterative methods in a subspace for partial generalized symmetric eigenvalue problems*, SIAM Journal on Numerical Analysis, 31 (1994), pp. 1226–1239.

[5] JARED L. AURENTZ, VASSILIS KALANTZIS, AND YOUSEF SAAD, *A GPU implementation of the filtered Lanczos procedure*, Tech. Report ys-2015-4, 2015.

[6] SATISH BALAY, SHRIRANG ABHYANKAR, MARK F. ADAMS, JED BROWN, PETER BRUNE, KRIS BUSCHELMAN, LISANDRO DALCIN, VICTOR EIJKHOUT, WILLIAM D. GROPP, DINESH KAUSHIK, MATTHEW G. KNEPLEY, LOIS CURFMAN MCINNES, KARL RUPP, BARRY F. SMITH, STEFANO ZAMPINI, AND HONG ZHANG, *PETSc users manual*, Tech. Report ANL-95/11 - Revision 3.6, Argonne National Laboratory, 2015.

[7] ———, *PETSc Web page*. `http://www.mcs.anl.gov/petsc`, 2015.

[8] SATISH BALAY, WILLIAM D. GROPP, LOIS CURFMAN MCINNES, AND BARRY F. SMITH, *Efficient management of parallelism in object oriented numerical software libraries*, in Modern Software Tools in Scientific Computing, E. Arge, A. M. Bruaset, and H. P. Langtangen, eds., Birkhäuser Press, 1997, pp. 163–202.

[9] C. BEKAS AND Y. SAAD, *Computation of smallest eigenvalues using spectral schur complements*, SIAM J. Sci. Comput., 27 (2006), pp. 458–481.

[10] J. K. BENNIGHOF AND R. B. LEHOUCQ, *An automated multilevel substructuring method for eigenspace computation in linear elastodynamics*, SIAM J. Sci. Comput., 25 (2004), pp. 2084–2106.

[11] L. M. CARVALHO, L. GIRAUD, AND P. LE TALLEC, *Algebraic two-level preconditioners for the schur complement method*, SIAM Journal on Scientific Computing, 22 (2001), pp. 1987–2005.

[12] TIMOTHY A. DAVIS, *Algorithm 832: Umfpack v4.3—an unsymmetric-pattern multifrontal method*, ACM Trans. Math. Softw., 30 (2004), pp. 196–199.

[13] TIMOTHY A. DAVIS AND YIFAN HU, *The university of florida sparse matrix collection*, ACM Trans. Math. Softw., 38 (2011), pp. 1:1–1:25.

[14] JAMES W. DEMMEL, *Applied Numerical Linear Algebra*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.

[15] H. FANG AND Y. SAAD, *A filtered Lanczos procedure for extreme and interior eigenvalue problems*, SIAM Journal on Scientific Computing, 34 (2012), pp. A2220–A2246.

[16] ALAN GEORGE, *Nested dissection of a regular finite element mesh*, 10 (1973), pp. 345–363.

[17] L. GIRAUD AND ET AL., *Sparse approximations of the schur complement for parallel algebraic hybrid solvers in 3d*, 2010.

[18] STEFAN GUTTEL, ERIC POLIZZI, PING TAK PETER TANG, AND GAUTIER VIAUD, *Zolotarev quadrature rules and load balancing for the feast eigensolver*, SIAM Journal on Scientific Computing, 37 (2015), pp. A2100–A2122.

[19] V. KALANTZIS, C. BEKAS, A. CURIONI, AND E. GALLOPOULOS, *Accelerating data uncertainty quantification by solving linear systems with multiple right-hand sides*, Numerical Algorithms, 62 (2013), pp. 637–653.

[20] V. KALANTZIS, R. LI, AND Y. SAAD, *Spectral schur complement techniques for symmetric eigenvalue problems*.

[21] G. KARYPIS AND V. KUMAR, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM Journal on Scientific Computing, 20 (1998), pp. 359–392.

[22] J. KESTYN, E. POLIZZI, AND P. TAK PETER TANG, *Feast eigensolver for non-hermitian problems*.

[23] L. KRONIK, A. MAKMAL, M. L. TIAGO, M. M. G. ALEMANY, M. JAIN, X. HUANG, Y. SAAD, AND J. R. CHELIKOWSKY, *PARSEC–the pseudopotential algorithm for real-space electronic structure calculations: recent advances and novel applications to nano-structures*, Phys. Status Solidi (B), 243 (2006), pp. 1063–1079.

[24] ANDREY KUZMIN, MATHIEU LUISIER, AND OLAF SCHENK, *Fast methods for computing selected elements of the green's function in massively parallel nanoelectronic device simulations*, in Proceedings of the 19th International Conference on Parallel Processing, Euro-Par'13, Berlin, Heidelberg, 2013, Springer-

Verlag, pp. 533–544.
[25] XIAOYE S. LI AND JAMES W. DEMMEL, *Superlu_dist: A scalable distributed-memory sparse direct solver for unsymmetric linear systems*, ACM Trans. Math. Softw., 29 (2003), pp. 110–140.
[26] ZHONGZE LI, YOUSEF SAAD, AND MASHA SOSONKINA, *parms: a parallel version of the algebraic recursive multilevel solver*, Numerical Linear Algebra with Applications, 10 (2003), pp. 485–509.
[27] S.H. LUI, *Kron's method for symmetric eigenvalue problems*, Journal of Computational and Applied Mathematics, 98 (1998), pp. 35 – 48.
[28] ———, *Domain decomposition methods for eigenvalue problems*, Journal of Computational and Applied Mathematics, 117 (2000), pp. 17 – 34.
[29] F. PELLEGRINI, SCOTCH *and* LIBSCOTCH *5.1 User's Guide*, INRIA Bordeaux Sud-Ouest, IPB & LaBRI, UMR CNRS 5800, 2010.
[30] COSMIN G. PETRA, OLAF SCHENK, MILES LUBIN, AND KLAUS GÄRTNER, *An augmented incomplete factorization approach for computing the schur complement in stochastic optimization*, SIAM J. Scientific Computing, 36 (2014).
[31] BERNARD PHILIPPE AND YOUSEF SAAD, *On correction equations and domain decomposition for computing invariant subspaces*, Computer Methods in Applied Mechanics and Engineering, 196 (2007), pp. 1471 – 1483. Domain Decomposition Methods: recent advances and new challenges in engineering.
[32] ERIC POLIZZI, *Density-matrix-based algorithm for solving eigenvalue problems*, Phys. Rev. B, 79 (2009), p. 115112.
[33] S. RAJAMANICKAM, E.G. BOMAN, AND M.A. HEROUX, *Shylu: A hybrid-hybrid solver for multicore platforms*, in Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International, May 2012, pp. 631–643.
[34] F.-H. ROUET J. XIA S. WANG, X. S. LI AND M. V. DE HOOP, *A parallel geometric multifrontal solver using hierarchically semiseparable structure*, ACM Trans. Math. Software, to appear.
[35] Y. SAAD, *Iterative Methods for Sparse Linear Systems*, Society for Industrial and Applied Mathematics, second ed., 2003.
[36] ———, *Numerical Methods for Large Eigenvalue Problems*, Society for Industrial and Applied Mathematics, 2011.
[37] YOUCEF SAAD AND MARTIN H. SCHULTZ, *Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM Journal on Scientific and Statistical Computing, 7 (1986), pp. 856–869.
[38] YOUSEF SAAD AND MARIA SOSONKINA, *Distributed schur complement techniques for general sparse linear systems*, SIAM Journal on Scientific Computing, 21 (1999), pp. 1337–1356.
[39] Y. SAAD AND B. SUCHOMEL, *Arms: an algebraic recursive multilevel solver for general sparse linear systems*, Numerical Linear Algebra with Applications, 9 (2002), pp. 359–378.
[40] TETSUYA SAKURAI AND HIROSHI SUGIURA, *A projection method for generalized eigenvalue problems using numerical integration*, Journal of Computational and Applied Mathematics, 159 (2003), pp. 119 – 128. 6th Japan-China Joint Seminar on Numerical Mathematics; In Search for the Frontier of Computational and Applied Mathematics toward the 21st Century.
[41] TETSUYA SAKURAI AND HIROTO TADANO, *Cirr: a rayleigh-ritz type method with contour integral for generalized eigenvalue problems*, Hokkaido Math. J., 36 (2007), pp. 745–757.
[42] V. SIMONCINI AND E. GALLOPOULOS, *Convergence properties of block gmres and matrix polynomials*, Linear Algebra and its Applications, 247 (1996), pp. 97 – 119.
[43] BARRY F. SMITH, PETTER E. BJØRSTAD, AND WILLIAM D. GROPP, *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*, Cambridge University Press, New York, NY, USA, 1996.
[44] MARC SNIR, STEVE OTTO, STEVEN HUSS-LEDERMAN, DAVID WALKER, AND JACK DONGARRA, *MPI-The Complete Reference, Volume 1: The MPI Core*, MIT Press, Cambridge, MA, USA, 2nd. (revised) ed., 1998.
[45] PING TAK PETER TANG AND ERIC POLIZZI, *Feast as a subspace iteration eigensolver accelerated by approximate spectral projection*, SIAM Journal on Matrix Analysis and Applications, 35 (2014), pp. 354–390.
[46] ANDREA TOSELLI AND OLOF WIDLUND, *Domain decomposition methods: algorithms and theory*, vol. 3, Springer, 2005.
[47] YUANZHE XI AND YOUSEF SAAD, *Least-squares rational filters for the solution of interior eigenvalue problems*, research report.

TABLE .1

Total elapsed time to compute the eigenpairs associated with eigenvalues $\lambda_{101}, \ldots, \lambda_{120}$ and $\lambda_{501}, \ldots, \lambda_{520}$, using the CI-M scheme as the number of quadrature nodes N_c and size of subspace $\hat{r}$ are varied. Value $\hat{r}^*$ implies the optimal value of $\hat{r}$ in terms of the total number of linear systems solved.

	Its	$P = 64$	$P = 128$	$P = 256$
$[\lambda_{101}, \lambda_{120}]$				
$N_c = 1$				
$\hat{r} = 50$	8	1,607.2	841.4	685.0
$\hat{r} = 100$	6	2,073.9	1,092.1	875.2
$\hat{r}^* = 40$	8	1,420.6	741.6	609.1
$N_c = 2$				
$\hat{r} = 50$	5	2,514.6	1,308.0	1,085.1
$\hat{r} = 100$	4	3,214.3	1,682.8	1,370.1
$\hat{r}^* = 33$	5	2,118.8	1,095.2	923.8
$N_c = 3$				
$\hat{r} = 50$	4	3,422.8	1,773.4	1,485.5
$\hat{r} = 100$	3	4,121.1	2,149.0	1,770.6
$\hat{r}^* = 24$	5	2,862.2	1,473.4	1,257.1
$[\lambda_{501}, \lambda_{520}]$				
$N_c = 1$				
$\hat{r} = 50$	9	1,723.9	904.3	732.5
$\hat{r} = 100$	5	1,840.5	966.9	780.0
$\hat{r}^* = 39$	9	1,492.9	780.4	638.5
$N_c = 2$				
$\hat{r} = 50$	5	2,514.4	1,308.1	1,085.6
$\hat{r} = 100$	4	2,214.3	1,682.8	1,370.1
$\hat{r}^* = 33$	5	2,118.0	1,095.7	923.6
$N_c = 3$				
$\hat{r} = 50$	4	3,422.9	1,774.3	1,485.1
$\hat{r} = 100$	3	4,121.8	2,149.7	1,770.1
$\hat{r}^* = 33$	5	3,177.2	1,642.8	1,358.9

TABLE .2

Total elapsed time to compute the eigenpairs associated with eigenvalues $\lambda_{101}, \ldots, \lambda_{200}$ and $\lambda_{501}, \ldots, \lambda_{600}$, using the CI-M scheme as the number of quadrature nodes N_c and size of subspace $\hat{r}$ are varied. Value $\hat{r}^*$ implies the optimal value of $\hat{r}$ in terms of the total number of linear systems solved.

	Its	$P = 64$	$P = 128$	$P = 256$
$[\lambda_{101}, \lambda_{200}]$				
$N_c = 1$				
$\hat{r} = 200$	14	7,206.1	3,845.6	2,965.0
$\hat{r} = 300$	9	6,972.9	3,702.5	2,870.0
$\hat{r}^* = 236$	10	6,179.6	3,294.9	2,547.1
$N_c = 2$				
$\hat{r} = 200$	5	6,013.9	3,185.4	2,510.1
$\hat{r} = 300$	5	8,346.5	4,437.0	3,406.1
$\hat{r}^* = 184$	5	5,640.7	2,983.1	2,354.8
$N_c = 3$				
$\hat{r} = 200$	4	7,628.2	4,027.6	3,195.1
$\hat{r} = 300$	4	10,421.7	5,529.2	4,335.1
$\hat{r}^* = 133$	5	6,673.6	3,520.2	2,810.4
$[\lambda_{501}, \lambda_{600}]$				
$N_c = 1$				
$\hat{r} = 200$	12	6,274.1	3,345.0	2,585.1
$\hat{r} = 400$	7	7,206.4	3,845.6	2,965.0
$\hat{r}^* = 165$	13	5,678.1	3,025.9	2,342.9
$N_c = 2$				
$\hat{r} = 200$	5	6,013.9	3,185.8	2,510.2
$\hat{r} = 400$	4	8,813.7	4,687.8	3,650.0
$\hat{r}^* = 166$	5	5,220.4	2,759.9	2,178.1
$N_c = 3$				
$\hat{r} = 200$	5	7,612.1	4,027.1	3,195.3
$\hat{r} = 400$	3	10,432.7	5,529.2	4,335.4
$\hat{r}^* = 123$	5	6,326.4	3,332.5	2,667.9