Domain Decomposition Approaches for Accelerating Contour Integration Eigenvalue Solvers for Symmetric Eigenvalue Problems

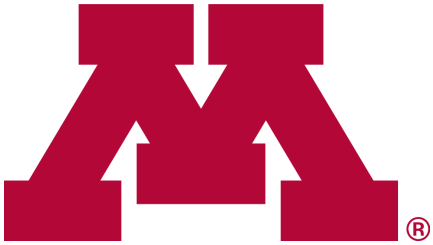
Vassilis Kalantzis, James Kestyn, Eric Polizzi, and Yousef Saad

January 2018

EPrint ID: 2018.2

Department of Computer Science and Engineering
University of Minnesota, Twin Cities

Preprints available from: `http://www-users.cs.umn.edu/kalantzi`



UNIVERSITY OF MINNESOTA

Supercomputing Institute

Domain decomposition approaches for accelerating contour integration eigenvalue solvers for symmetric eigenvalue problems

Vassilis Kalantzis[1*], James Kestyn[2], Eric Polizzi[2] and Yousef Saad[1]

[1]*Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455, USA*
[2]*Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA 01003, USA*

SUMMARY

This paper discusses techniques for computing a few selected eigenvalue-eigenvector pairs of large and sparse symmetric matrices. A recently developed 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 are reported. These results show that domain decomposition approaches can lead to reduced runtimes and improved scalability.

Received ...

KEY WORDS: Domain decomposition, symmetric eigenvalue problem, Cauchy integral formula, parallel computing, FEAST.

1. INTRODUCTION

A common approach for computing all eigenvalues located inside an interval $[\alpha, \beta]$ for a symmetric matrix A 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.

In this paper we consider techniques in which the subspace $\mathcal{U}$ is extracted via an approximation of the spectral projector obtained by numerically integrating the resolvent $(\zeta I - A)^{-1}$, where $\zeta \in \mathbb{C}$, 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 [38, 44, 45, 49]. We focus on distributed computing environments, possibly with a large number of processors, and study contour integration eigenvalue solvers (eigensolvers) from a domain decomposition viewpoint [47, 50]. Domain decomposition techniques for the solution of eigenvalue problems have been studied in the past, see for example [10, 11, 24, 34, 37], but to our knowledge these methods have not yet been considered[†] within a contour integration framework.

*Correspondence to: Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455, USA. E-mail: kalan019@umn.edu

†This work supported jointly by NSF under awards CCF-1505970 and CCF-1510010, 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. Vassilis Kalantzis was also partially supported by a Gerondelis Foundation Fellowship.

†An exception is the approach presented in [25] which appeared while this paper was under review.

Contour integration eigensolvers have mostly been 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 this context (see also [20]). 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 resulting 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, and an implementation of FEAST that uses a standard parallel sparse direct solver. Similar behavior was observed in [27] for using an application-specific domain decomposition linear system solver in FEAST. In contrast, in this paper we consider domain decomposition using algebraic partitionings obtained by a graph partitioner.

Domain decomposition type methods naturally lend themselves to parallelization in multi-core and/or many-core environments. This paper discusses practical aspects of an implementation of the proposed numerical schemes in distributed computing environments. Moreover, we consider different levels of parallelism by combining distributed and shared memory computing. Finally, we discuss a modified contour integration scheme which shares similarities with the work of Beyn in [12] and approximates only certain parts of the contour integral of the matrix resolvent. This approach leads to a numerical scheme that can be computationally more efficient than following the standard approach of numerically integrating the entire resolvent, especially when the linear system solutions associated with the interior variables of the subdomains are relatively expensive.

The organization of this paper is as follows. In Section 2 we describe the main idea behind contour integration eigensolvers, and the 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 $[-1, 1]$. When $[\alpha, \beta] \neq [-1, 1]$, we will be implicitly mapping $[\alpha, \beta]$ to $[-1, 1]$ by

$$A := (A - cI)/e, \quad c = \frac{\alpha + \beta}{2}, \quad e = \frac{\beta - \alpha}{2}. \tag{1}$$

Let A have r eigenvalues, denoted by λ_1,..., λ_r, located inside the interval $[-1, 1]$, and let $X = [x^{(1)}, \ldots, x^{(r)}]$ be the $n \times r$ orthonormal matrix formed by the corresponding (normalized) 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 [40]:

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

where Γ is a smooth, counter-clockwise oriented curve (e.g., a circle) 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 matrix $V \in \mathbb{R}^{n \times \hat{r}}$ such 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 $(\lambda_1, x^{(1)}), \ldots, (\lambda_r, x^{(r)})$.

(2010)
DOI: 10.1002/nla

In practice, the spectral projector in (2) is approximated by numerical quadrature. A basis of the approximate invariant subspace then takes the form:

$$\mathcal{P}V \approx \tilde{\mathcal{P}}V = \sum_{j=1}^{2N_c} \omega_j (\zeta_j I - A)^{-1} V, \tag{3}$$

where $\{\zeta_j, \omega_j\}_{1 \le j \le 2N_c}$ are the quadrature node-weight pairs of the quadrature rule. The numerical integration scheme in (3) approximates the exact spectral projector $\mathcal{P}$ in (2) by the approximate projector $\tilde{\mathcal{P}} = -\rho(A)$, where

$$\rho(\zeta) = \sum_{j=1}^{2N_c} \frac{\omega_j}{\zeta - \zeta_j}. \tag{4}$$

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 $\rho(\lambda_1), \rho(\lambda_2), \ldots, \rho(\lambda_n)$ of $\rho(A)$.

The accuracy of the approximate eigenpairs computed by a Rayleigh-Ritz projection on the subspace created by (3) can be improved by repeating the procedure in (3), 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 (3), this approach essentially amounts to Subspace Iteration with the matrix A replaced by $\rho(A)$, i.e., the FEAST package [27, 28, 38, 49]; see also [3, 29, 31, 53]. 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) [5, 44, 45]. The poles of this scalar function are the eigenvalues of A. See also [6] for a pole-finding eigenvalue solver based on rational interpolation that exploits real arithmetic only, and [8, 9, 12] for applications of contour integration to the solution of nonlinear eigenvalue problems.

3. THE DOMAIN DECOMPOSITION FRAMEWORK

Throughout this paper, we assume a non-overlapping p-way partitioning of the adjacency graph of A, obtained by a graph partitioner [26, 35]. Each vertex is an equation-unknown pair (equation number i and unknown number i) and the partitioner subdivides the vertex set into p non-intersecting subsets. After partitioning, 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 ith subdomain $i = 1, \ldots, p$, a local reordering is applied in which interior points are listed before the interface ones.

With this, the local piece of an eigenvector x of A residing in the ith subdomain, x_i, can be split into two parts: the subvector $u_i \in \mathbb{R}^{d_i}$ of internal components followed by the subvector $y_i \in \mathbb{R}^{s_i}$ of local interface components, where d_i denotes the number of interior variables and s_i the number of interface variables of the ith subdomain, $i = 1, \ldots, p$. If we stack all interior variables $u_1, u_2, \cdots, u_p$ into a vector u, in this order, and 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:

$$\underbrace{\begin{pmatrix} B_1 & & & & E_1 \\ & B_2 & & & E_2 \\ & & \ddots & & \vdots \\ & & & B_p & E_p \\ E_1^T & E_2^T & \ldots & E_p^T & C \end{pmatrix}}_{PAP^T} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_p \\ y \end{pmatrix} = \lambda \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_p \\ y \end{pmatrix}, \tag{5}$$

where $B_i \in \mathbb{R}^{d_i \times d_i}$ represents the couplings between the interior variables of subdomain i, $E_i \in \mathbb{R}^{d_i \times s}$ denotes the matrix that maps all interface variables to the interior variables of subdomain i, and $C \in \mathbb{R}^{s \times s}$ denotes the matrix that represents the couplings between the interface

DOI: 10.1002/nla

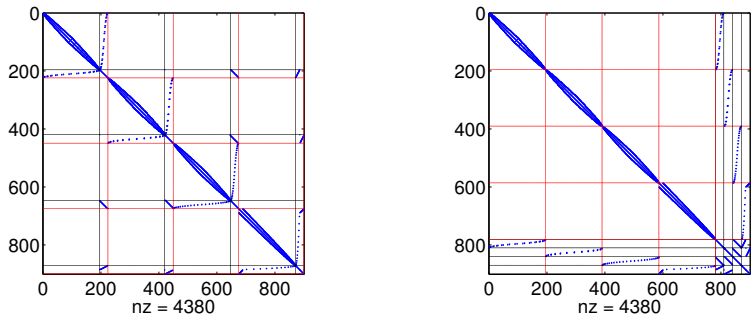


Figure 1. An example of a 2D Laplacian matrix reordered using $p = 4$ subdomains. Local (left) and global (right) viewpoints.

variables, with $s = s_1 + \ldots + s_p$ (we also set $d = d_1 + \ldots + d_p$). The matrix E_i has the form $E_i = [0_{d_i,\ell_i}, \hat{E}_i, 0_{d_i,\nu_i}]$, 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 $\chi \times \psi$. Figure 1 illustrates the above reordering for a 2D Laplacian matrix using $p = 4$ subdomains.

The coefficient matrix of the system (5) can be also written in a more compact form as

$$A = \begin{pmatrix} B & E \\ E^T & C \end{pmatrix}, \tag{6}$$

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 (5), or, equivalently, (6).

4. CONTOUR INTEGRATION IN THE DOMAIN DECOMPOSITION FRAMEWORK

In this section we present two numerical schemes that utilize contour integration approaches from a domain decomposition perspective. Both schemes start with the expression of the resolvent operator $(\zeta I - A)^{-1}$ within 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 [16]:

$$A - \zeta I = \begin{pmatrix} I & 0 \\ E^T(B - \zeta I)^{-1} & I \end{pmatrix} \begin{pmatrix} B - \zeta I & E \\ 0 & S(\zeta) \end{pmatrix}, \tag{7}$$

where

$$S(\zeta) = C - \zeta I - E^T(B - \zeta I)^{-1}E \tag{8}$$

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

$$(A - \zeta I)^{-1} = \begin{pmatrix} (B - \zeta I)^{-1} & -(B - \zeta I)^{-1}ES(\zeta)^{-1} \\ 0 & S(\zeta)^{-1} \end{pmatrix} \begin{pmatrix} I & 0 \\ -E^T(B - \zeta I)^{-1} & I \end{pmatrix}, \tag{9}$$

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 (9), and defining $F(\zeta) = (B - \zeta I)^{-1}E$, we get:

$$(A - \zeta I)^{-1} = \begin{pmatrix} (B - \zeta I)^{-1} + F(\zeta)S(\zeta)^{-1}F(\zeta)^T & -F(\zeta)S(\zeta)^{-1} \\ -S(\zeta)^{-1}F(\zeta)^T & S(\zeta)^{-1} \end{pmatrix}. \tag{10}$$

(2010)
DOI: 10.1002/nla

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). Then, $\mathcal{P}$ can be written in a 2×2 block form, by integrating each block of $(A-\zeta I)^{-1}$ separately:

$$\mathcal{P} = \frac{-1}{2i\pi}\int_\Gamma (A-\zeta I)^{-1}d\zeta \equiv \begin{pmatrix} \mathcal{H} & -\mathcal{W} \\ -\mathcal{W}^\mathcal{T} & \mathcal{G} \end{pmatrix} \tag{11}$$

with

$$\begin{cases} \mathcal{H} = \dfrac{-1}{2i\pi}\int_\Gamma [(B-\zeta I)^{-1} + F(\zeta)S(\zeta)^{-1}F(\zeta)^T]d\zeta \\ \mathcal{G} = \dfrac{-1}{2i\pi}\int_\Gamma S(\zeta)^{-1}d\zeta \\ \mathcal{W} = \dfrac{-1}{2i\pi}\int_\Gamma F(\zeta)S(\zeta)^{-1}d\zeta. \end{cases} \tag{12}$$

In order to extract an eigenspace from the expression of $\mathcal{P}$ in (11), we consider the product $\mathcal{P}V$, where V is a matrix with $\hat{r}\geq r$ columns, written as

$$\mathcal{P}\begin{pmatrix} V_u \\ V_s \end{pmatrix} = \begin{pmatrix} \mathcal{H}V_u - \mathcal{W}V_s \\ -\mathcal{W}^T V_u + \mathcal{G}V_s \end{pmatrix} \equiv \begin{pmatrix} Z_u \\ Z_s \end{pmatrix}, \tag{13}$$

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 interior and interface variables, respectively. We finally get

$$\begin{cases} Z_u = \dfrac{-1}{2i\pi}\int_\Gamma (B-\zeta I)^{-1}V_u d\zeta - \dfrac{-1}{2i\pi}\int_\Gamma F(\zeta)S(\zeta)^{-1}[V_s - F(\zeta)^T V_u]d\zeta \\ Z_s = \dfrac{-1}{2i\pi}\int_\Gamma S(\zeta)^{-1}[V_s - F(\zeta)^T V_u]d\zeta. \end{cases} \tag{14}$$

Assuming that $V\in\mathbb{R}^{n\times\hat{r}}$ is chosen such that V^TX has rank r, (14) captures the exact invariant subspace of A associated with the sought eigenvalues, and $Z\equiv\mathcal{P}V$ can be exploited 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 contour integrals in (14) will have to be approximated numerically. Once a quadrature rule is selected, with quadrature nodes and weights $\{\zeta_j,\omega_j\},\ j=1,\ldots,2N_c$, (14) is approximated by the following summations:

$$\tilde{Z}_u = -\sum_{j=1}^{2N_c}\omega_j(B-\zeta_j I)^{-1}V_u + \sum_{j=1}^{2N_c}\omega_j F(\zeta_j)S(\zeta_j)^{-1}[V_s - F(\zeta_j)^T V_u], \tag{15}$$

$$\tilde{Z}_s = -\sum_{j=1}^{2N_c}\omega_j S(\zeta_j)^{-1}[V_s - F(\zeta_j)^T V_u]. \tag{16}$$

The numerical integration can be performed by one of the available quadrature rules, e.g., the Gauss-Legendre [38] or the trapezoidal [44] rules. Since the eigenvalues of A are real, using a rule in which[‡] the quadrature nodes appear in conjugate pairs, i.e., $\zeta_j=\overline{\zeta_{j+N_c}},\ j=1,\ldots,N_c$, reduces the cost of the numerical approximation by a factor of two, since

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

Viewing contour integration as a form of rational filtering, additional rational filters become possible, e.g., Zolotarev rational filters [22], or least-squares filters [52], however, we do not explore these options in this paper.

[‡]We assume here that none of the quadrature nodes lies on the real axis

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 (15)-(16) 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 set* $\tilde{Z} = [\tilde{Z}_u^T, \tilde{Z}_s^T]^T = 0$
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. $W_s := S(\zeta_j)^{-1} W_s, \qquad \tilde{Z}_s := \tilde{Z}_s - \Re e(\omega_j W_s)$
6. $W_u := W_u - (B - \zeta_j I)^{-1} E W_s, \qquad \tilde{Z}_u := \tilde{Z}_u - \Re e(\omega_j W_u)$
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 matrix $B - \zeta_j I,\ j = 1, \ldots, N_c$ is decoupled into factorizations of the matrices $B_i - \zeta_j I,\ i = 1, \ldots, p$, each one being local to the ith subdomain. Moreover, only the real parts of $\tilde{Z}_s$ and $\tilde{Z}_u$ need be retained. Step 8 of Algorithm 4.1 extracts the approximate eigenpairs of A by a Rayleigh-Ritz projection, and also verifies whether all eigenpairs inside $[-1, 1]$ are approximated up to a sufficient accuracy (this part is omitted from the description of the algorithm). If not satisfied with the accuracy achieved, we can repeat steps 2-7 using the current approximate eigenvectors as the new matrix V. In the latter case, the DD-FP scheme can be seen as a domain decomposition-based Subspace Iteration approach.

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 is equivalent to the FEAST algorithm tied to 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.

Let the spectral projector $\mathcal{P}$, defined in (11), 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:

$$X \equiv \begin{pmatrix} X_u \\ X_s \end{pmatrix}, P = XX^T \rightarrow \mathcal{P} = [\mathcal{P}_1, \mathcal{P}_2] = \begin{pmatrix} X_u X_u^T & X_u X_s^T \\ X_s X_u^T & X_s X_s^T \end{pmatrix}. \tag{17}$$

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 located inside $[\alpha, \beta]$, the range of $\mathcal{P}$ can be captured by the range of $\mathcal{P}_2 = XX_s^T = [X_u^T,\ X_s^T]^T X_s^T$. Equating (17) with (11) shows 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 the above scheme approximates the spectral projector $\mathcal{P}$ only partially, we will refer to it as "Domain Decomposition Partial Projector" (DD-PP).

(2010)
DOI: 10.1002/nla

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

$$\mathcal{G}R = \frac{-1}{2i\pi}\int_\Gamma S(\zeta)^{-1}Rd\zeta, \quad -\mathcal{W}R = \frac{1}{2i\pi}\int_\Gamma (B-\zeta I)^{-1}ES(\zeta)^{-1}Rd\zeta, \tag{18}$$

for any $R \in \mathbb{R}^{s\times\hat{r}}$ such that $\mathrm{rank}(X_s^T R) \equiv \mathrm{rank}(X_s)$.

In practice, (18) will be approximated numerically by a quadrature rule, and thus

$$\mathcal{G}R \approx \tilde{\mathcal{G}}R = -\sum_{j=1}^{2N_c}\omega_j S(\zeta_j)^{-1}R, \quad -\mathcal{W}R \approx -\tilde{\mathcal{W}}R = \sum_{j=1}^{2N_c}\omega_j (B-\zeta_j I)^{-1}ES(\zeta_j)^{-1}R. \tag{19}$$

Combining the contribution of all quadrature nodes together, the final subspace accumulation proceeds as in Algorithm 4.2, which we abbreviate as DD-PP.

ALGORITHM *4.2*
DD-PP

0. *Start with a random* $R \in \mathbb{R}^{s\times\hat{r}}$ *and set* $\tilde{Z} = [\tilde{Z}_u^T, \tilde{Z}_s^T]^T = 0$
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. $\quad W_u := -(B-\zeta_j I)^{-1}EW_s,$ $\qquad \tilde{Z}_u := \tilde{Z}_u - \Re e(\omega_j W_u)$
4. *End*
5. *Rayleigh-Ritz: solve the eigenvalue problem* $\tilde{Z}^T A\tilde{Z}Q = \tilde{Z}^T\tilde{Z}Q\Theta$

4.2.2. Analysis of the DD-PP scheme The matrix inverse $(A-\zeta I)^{-1}$ in (10) can be also written in terms of the eigenvectors of A as

$$(A-\zeta I)^{-1} = \sum_{i=1}^{n}\frac{x^{(i)}(x^{(i)})^T}{\lambda_i - \zeta}, \tag{20}$$

where we assume that $\zeta \notin \Lambda(A)$ with $\Lambda(A)$ denoting the spectrum of A. If we partition each eigenvector of A as $x^{(i)} = [(u^{(i)})^T, (y^{(i)})^T]^T$, $i = 1, \ldots, n$, and combine (20) with (10) for all $\zeta \equiv \zeta_j$, $j = 1, \ldots, 2N_c$, we get

$$\sum_{j=1}^{2N_c}\omega_j(A-\zeta_j I)^{-1} = \sum_{i=1}^{n}\rho(\lambda_i)\begin{bmatrix} u^{(i)}(u^{(i)})^T & u^{(i)}(y^{(i)})^T \\ y^{(i)}(u^{(i)})^T & y^{(i)}(y^{(i)})^T \end{bmatrix} \tag{21}$$

where $\rho(\zeta)$ is defined in (4). Equating the (1,2) and (2,2) blocks of (10) and (21) we finally get

$$\tilde{\mathcal{G}} = -\sum_{j=1}^{2N_c}\omega_j S(\zeta_j)^{-1} = -\sum_{i=1}^{n}\rho(\lambda_i)y^{(i)}(y^{(i)})^T \tag{22}$$

$$-\tilde{\mathcal{W}} = \sum_{j=1}^{2N_c}\omega_j(B-\zeta_j I)^{-1}ES(\zeta_j)^{-1} = -\sum_{i=1}^{n}\rho(\lambda_i)u^{(i)}(y^{(i)})^T. \tag{23}$$

By (22) and (23), we can see that if $\rho(\lambda_{r+1}), \ldots, \rho(\lambda_n)$ damp sufficiently close to zero, then $\mathrm{range}\{\tilde{\mathcal{G}}\} \approx \mathrm{span}\{y^{(1)}, \ldots, y^{(r)}\}$, and $\mathrm{range}\{\tilde{\mathcal{W}}\} \approx \mathrm{span}\{u^{(1)}, \ldots, u^{(r)}\}$, which are exactly the subspaces required to retrieve the sought eigenpairs $(\lambda_1, x^{(1)}), \ldots, (\lambda_r, x^{(r)})$ of matrix A. In the opposite case, an increase in $\hat{r}$, the number of columns in matrix R, becomes necessary.

Figure 2 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). In contrast

DOI: 10.1002/nla

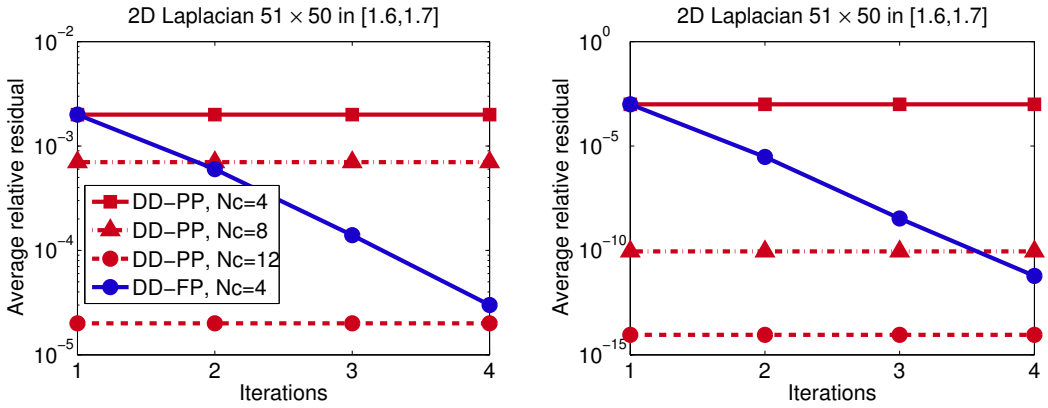


Figure 2. Average residual norm for a 51×50 2D Laplacian in the interval $[1.6, 1.7]$. Left: $\hat{r} = r$. Right: $\hat{r} = 2r$. The Gauss-Legendre quadrature rule was used [2].

to DD-PP, DD-FP can use a smaller number of quadrature nodes and correct the approximate eigenpairs of A 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 close to that of the DD-PP scheme utilizing $N_c = 12$ quadrature nodes.

We note at this point that the above discussion on the accuracy of DD-PP is independent of the number of subdomains p.

4.3. Computational comparison of the DD-FP and DD-PP schemes

From a numerical viewpoint, both the DD-PP and DD-FP schemes can perform similarly, but, from a computational viewpoint, there are some notable differences. DD-PP has a lower computational complexity per quadrature node than DD-FP, since it avoids performing the first two steps of the latter. A straightforward calculation reveals that for each quadrature node, the DD-FP scheme also introduces $n \times \hat{r}$ more floating-point operations (FLOPS) than the DD-PP scheme (the block matrix subtractions in Steps 3 and 5 in Algorithm 4.1). When accounting for all quadrature nodes together, the DD-FP scheme introduces $N_c \times \hat{r} \times [cost\_solve(B - \zeta I) + cost\_MV(E) + n]$ additional FLOPS compared to DD-PP. 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.

Assuming a distributed memory environment, in which each subdomain is assigned to a different processor group, the additional computational cost introduced by the ith subdomain when using DD-FP compared to DD-PP, amounts to $N_c \times \hat{r} \times [cost\_solve(B_i - \zeta I) + cost\_MV(E_i) + d_i]$. Thus, if d_i and/or r are large, the dense matrix operations in Steps 3 and 5 of Algorithm 4.1 become noticeable.

5. SOLVING LINEAR SYSTEMS WITH THE SCHUR COMPLEMENT MATRICES

The major distributed 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 each subdomain is assigned to a different processor, $S(\zeta)$ is distributed by rows among the different processors and has a natural block structure of the form

$$S(\zeta) = \begin{pmatrix} S_1(\zeta) & E_{12} & \dots & E_{1p} \\ E_{21} & S_2(\zeta) & \dots & E_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ E_{p1} & E_{p2} & \dots & S_p(\zeta) \end{pmatrix}, \tag{24}$$

where

$$S_i(\zeta) = C_i - \zeta I - \hat{E}_i^T (B_i - \zeta I)^{-1} \hat{E}_i, \; i = 1, \dots, p,$$

is the "local" Schur complement matrix that corresponds to the ith subdomain, and the off-diagonal blocks E_{ik}, $i, k = 1, \dots, p$, $i \neq k$, are sparse matrices of size $s_i \times s_k$ which account for the coupling among the different subdomains and are nonzero only if subdomains i and k are adjacent.

The standard approach to solve the distributed linear systems with the Schur complement in (24) is to explicitly form $S(\zeta)$ and compute its LU factorization by a call to a parallel sparse direct solver, see [4, 32]. For problems issued from discretizations of 2D domains, forming and factorizing $S(\zeta)$ explicitly is an attractive option since the size of the Schur complement is small even for a large number of subdomains. On the other hand, Schur complements that originate from discretizations of 3D computational domains typically require much more memory since in the 3D case the size of the Schur complement can become exceedingly large [51]. An alternative discussed next is to solve the linear systems without forming $S(\zeta)$,using a preconditioned iterative method (e.g., GMRES [41]).

5.1. Schur complement preconditioners

In this paper we consider sparsified approximations of $S(\zeta)$ which are based on sparsity and/or numerical constraints [13, 21, 39], a procedure summarized in Algorithm 5.1. Other Schur complement preconditioning approaches can be found in [33, 42, 43].

To form the preconditioner, denoted by $S_G(\zeta)$, 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 ith subdomain forms its local Schur complement

$$\hat{S}_i(\zeta) = C_i - \zeta I - (\hat{U}_i^{-T} \hat{E}_i)^T (\hat{L}_i^{-1} \hat{E}_i), \tag{25}$$

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 $B_i - \zeta I$, $i = 1, \dots, p$. Overall, the preconditioner takes the form:

$$S_G(\zeta) = \begin{pmatrix} \hat{S}_1(\zeta) & E_{12} & \dots & E_{1p} \\ E_{21} & \hat{S}_2(\zeta) & \dots & E_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ E_{p1} & E_{p2} & \dots & \hat{S}_p(\zeta) \end{pmatrix}. \tag{26}$$

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

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

(2010)
DOI: 10.1002/nla

Here are 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 [30, 36]. More details will be given in Section 6.

5.2. *Matrix-Vector products with* $S(\zeta)$

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

$$S(\zeta)v = (C - \zeta I)v - E^T(B - \zeta I)^{-1}Ev. \tag{27}$$

Since B and E are block-diagonal and distributed among the set of available processors, 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 (27) are:

1. Compute $E^T(B - \zeta I)^{-1}Ev$ (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, increase communication cost.

6. EXPERIMENTS

All numerical schemes were implemented in C/C++ and built on top of the PETSc [7, 18, 19] and Intel Math Kernel scientific libraries [1]. For PETSc, we used a complex build.[§] The source files were compiled with the Intel MPI compiler `mpiicpc`, using the -O3 optimization level.

Regarding DD-FP and DD-PP, the computational domain was partitioned in p non-overlapping subdomains using METIS [26], and each subdomain was then assigned to a distinct processor group. Communication between different processor groups was achieved by means of the Message Passing Interface standard (MPI) [48]. Throughout this section, the number of subdomains p will also denote the number of MPI processes. The LU factorizations and linear system solutions associated with the block-diagonal matrices $B - \zeta_j I,\ j = 1, \ldots, N_c$, were performed by the shared-memory, multi-threaded version of the Pardiso library (version 5.0.0) [30, 36]. Unless stated otherwise, the default number of threads per MPI process, as denoted by variable τ, will be equal to one.

The quadrature nodes and weights $\zeta_j,\ \omega_j,\ j = 1, \ldots, N_c$ were computed by the Gauss-Legendre quadrature rule of order $2N_c$ [2], retaining only the N_c quadrature nodes (and associated weights) with positive imaginary part. While it is possible to utilize block Krylov subspace solvers [23], e.g., block GMRES [46], throughout the rest of this section, the multiple right-hand sides will be solved one after the other. Whenever we computed an incomplete factorization of the block-diagonal matrices $B - \zeta_j I,\ j = 1, \ldots, N_c$, that was obtained by the UMFPACK [14] library, and the resulting triangular factors were then passed to Pardiso.

Finally, all distributed memory matrix factorizations and triangular substitutions associated with the distributed matrices $A - \zeta_j I$ and $S(\zeta_j),\ j = 1, \ldots, N_c$, were performed by MUMPS [4].

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

Table I. Average amount of time spent on a single quadrature node in DD-PP and DD-FP to approximate the eigenvalues $\lambda_{1001}, \ldots, \lambda_{1200}$ and associated eigenvectors for three discretized 2D Laplacians.

	$p = 8$		$p = 16$		$p = 32$		$p = 64$	
	DD-PP	DD-FP	DD-PP	DD-FP	DD-PP	DD-FP	DD-PP	DD-FP
$n = 500^2$								
$\hat{r} = r + 10$	9.45	13.7	6.77	8.91	5.25	6.34	4.65	5.30
$\hat{r} = 3r/2 + 10$	13.5	19.5	9.65	12.7	7.59	9.01	6.64	7.54
$\hat{r} = 2r + 10$	18.1	26.0	12.9	16.8	10.0	12.1	8.83	10.1
$n = 1000^2$								
$\hat{r} = r + 10$	41.8	62.7	25.3	35.8	17.9	23.1	14.8	19.0
$\hat{r} = 3r/2 + 10$	59.7	89.5	36.0	49.9	25.5	33.1	21.1	26.9
$\hat{r} = 2r + 10$	79.1	119.3	68.1	68.1	34.1	44.2	28.4	36.3
$n = 1500^2$								
$\hat{r} = r + 10$	100.8	140.7	65.2	88.8	39.9	44.2	29.5	37.8
$\hat{r} = 3r/2 + 10$	144.2	201.3	93.1	126.4	57.6	63.9	42.6	54.9
$\hat{r} = 2r + 10$	192.7	268.6	124.5	168.9	76.0	84.3	56.7	72.7

6.1. *Computational system*

The experiments were performed on the `Mesabi` Linux cluster at Minnesota Supercomputing Institute. `Mesabi` consists of 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. 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 following eigenvalue problem,

$$-\Delta u = \lambda u, \tag{28}$$

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 $n = n_x n_y n_z$.

6.3. *A comparison of the DD-FP and DD-PP schemes for 2D domains*

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

The interval of interest was arbitrarily set to $[\alpha, \beta] = [(\lambda_{1000} + \lambda_{1001})/2, (\lambda_{1200} + \lambda_{1201})/2]$ (and thus $r = 200$). We used $N_c = 4$, $N_c = 8$ and $N_c = 12$ quadrature nodes, while we also varied the number of right-hand sides, $\hat{r}$. Table I reports the average time spent on a single quadrature node for the case $N_c = 8$. Per quadrature node timings for the rest of the values of N_c were basically identical. DD-PP was always faster than DD-FP, especially as p obtained smaller values, and n and $\hat{r}$ larger values, respectively. The latter results lie in agreement with the discussion in Section 4.3.

Figure 3 plots the maximum residual norm of the approximate eigenpairs of the 2D Laplacian of size $n = 1000^2$ for all different combinations of N_c and $\hat{r}$ reported in Table I. The residual norms of the DD-FP and DD-PP schemes were of the same order of magnitude, therefore we report results only for the DD-PP scheme.

The last experiment of this section focuses on a comparison between DD-FP and a PETSc-based implementation of the FEAST algorithm that utilizes MUMPS to factorize and solve the

DOI: 10.1002/nla

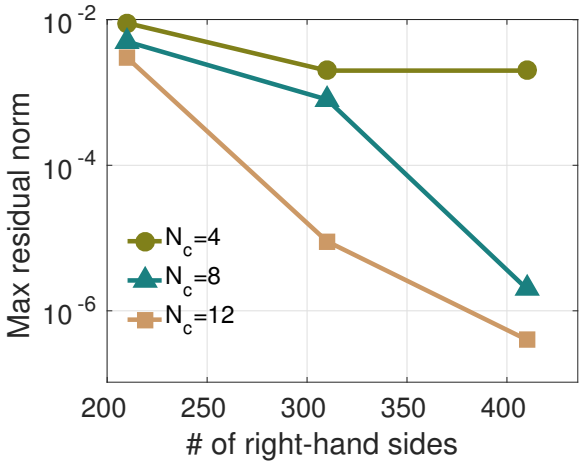


Figure 3. Maximum residual norm of the approximation of the eigenpairs inside the interval $[\alpha, \beta] = [(\lambda_{1000} + \lambda_{1001})/2, (\lambda_{1200} + \lambda_{1201})/2]$, when DD-PP is applied to the $n = 1000^2$ Laplacian.

Table II. Wall-clock time 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, as the values of N_c and $\hat{r}$ vary. "Its" denotes the number of outer iterations required by Subspace Iteration.

	Its	$p = 64$		$p = 128$		$p = 256$	
		CI-M	DD-FP	CI-M	DD-FP	CI-M	DD-FP
$N_c = 2$							
$\hat{r} = 3r/2 + 10$	9	3,922.7	2,280.6	2,624.3	1,242.4	1,911.2	859.5
$\hat{r} = 2r + 10$	5	2,863.2	1,764.5	1,877.7	998.5	1,255.5	615.3
$N_c = 4$							
$\hat{r} = 3r/2 + 10$	5	4,181.5	2,357.0	2,815.7	1,280.2	1,874.1	877.5
$\hat{r} = 2r + 10$	4	4,330.3	2,571.4	2,869.5	1,462.9	2,023.2	1,036.2
$N_c = 6$							
$\hat{r} = 3r/2 + 10$	3	3,710.3	2,068.2	2,504.1	1,122.1	1,790.8	766.5
$\hat{r} = 2r + 10$	3	4,774.8	2,798.5	3,177.7	1,595.2	2,743.6	1,125.1
$N_c = 8$							
$\hat{r} = 3r/2 + 10$	3	4,911.6	2,722.2	3,318.7	1,476.1	2,367.7	1,006.5
$\hat{r} = 2r + 10$	2	4,204.7	2,445.2	2,802.1	1,395.4	1,806,6	982.1

linear systems with matrices $A - \zeta_j I, \ j = 1, \ldots, N_c$. The latter will be referred to as Contour Integration-MUMPS (CI-M). Table II lists the wall-clock times of DD-FP and CI-M to compute all eigenpairs located inside the interval $[\alpha, \beta] = [(\lambda_{1000} + \lambda_{1001})/2, (\lambda_{1200} + \lambda_{1201})/2]$ for the $n = 1500^2$ Laplacian. Each eigenpair was sought to at least eight digits of accuracy, while "Its" denotes the number of outer iterations (same in both schemes). For CI-M, p denotes the number of MPI processes set in MUMPS. Increasing N_c leads to fewer outer iterations, although this does not necessarily imply lower wall-clock times. The performance gap between the DD-FP and CI-M schemes follows a slightly increasing trend as larger values of p are used, mainly because the linear system solution phase scales better in DD-FP than what in CI-M.

6.4. 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, with $n_x = n_y = n_z = 150$ ($n = 3,375,000$). For 3D problems of this size, a direct formation and factorization of the Schur complement matrices can be rather expensive, and, depending on the number of eigenvalues sought,

DOI: 10.1002/nla

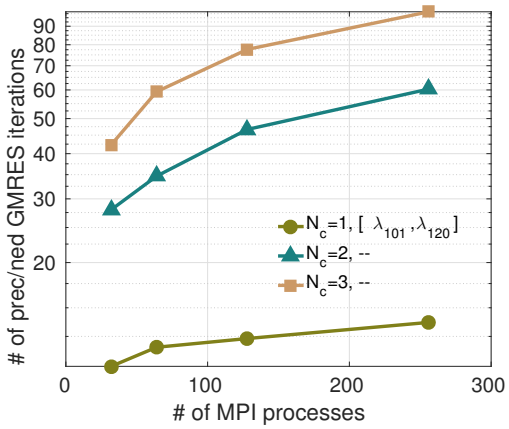


Figure 4. Total number of preconditioned GMRES iterations in order to solve a linear system with a single right-hand side for various values of N_c and p.

as well as their location in the spectrum, preconditioned iterative solvers might form a better alternative.

In contrast with Section 6.3, the solution of linear systems with the Schur complement matrices far dominates the overall computational time, and thus the DD-FP and DD-PP schemes are almost identical when it comes to wall-clock times. Thus, we only compare the DD-FP and CI-M schemes, and consider the problem of computing the smallest r=20 and all r=100 eigenvalues (and associated eigenvectors) located inside the intervals: $[\alpha, \beta] = [(\lambda_{100} + \lambda_{101})/2, (\lambda_{200} + \lambda_{201})/2]$, and $[\alpha, \beta] = [(\lambda_{500} + \lambda_{501})/2, (\lambda_{600} + \lambda_{601})/2]$.

6.4.1. DD-FP with preconditioned iterative linear system solvers Figure 4 lists the total number of preconditioned GMRES iterations to compute $\sum_{j=1}^{N_c} S(\zeta_j)^{-1} v$ for a random $v \in \mathbb{C}^s$. Details on the preconditioner used will be given later in this section. The interval of interest was set to $[\alpha, \beta] = [(\lambda_{100} + \lambda_{101})/2, (\lambda_{120} + \lambda_{121})/2]$. We can observe that as N_c increases, the number of preconditioned GMRES iterations also increases. Indeed, 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 [20]. By construction, higher values of N_c will lead to some quadrature nodes being closer to the real axis. Thus, when iterative solvers are exploited, setting N_c to a low value, e.g. $N_c = 1$ or $N_c = 2$, might in practice be a good choice.

Throughout this section, the iterative linear system solver used by DD-FP will be the right preconditioned GMRES, allowing up to 250 iterations per restart. The linear system solution process will terminate as soon as the norm of the residual of the corresponding approximate solution is ten orders of magnitude smaller compared to the initial residual norm.

6.4.2. Combining the distributed and shared memory paradigms When each subdomain is handled by a distinct MPI process, increasing the amount of parallelism in DD-FP leads to an increase in the size of the Schur complement matrices.[¶] Instead, we can use lower values for p and increase the number of available compute threads, τ, in Pardiso to solve the linear system solutions with matrices $B - \zeta_j I,\ j = 1, \ldots, N_c$.

Table III shows a comparison between a) flat MPI, and b) a hybrid (MPI+Threads) implementation using one thread per compute core, to compute $\sum_{j=1}^{N_c} S(\zeta_j)^{-1} v,\ N_c = 1$, where $v \in \mathbb{C}^s$ is a random vector and $[\alpha, \beta] = [(\lambda_{100} + \lambda_{101})/2, (\lambda_{200} + \lambda_{201})/2]$. To construct the preconditioner $S_G(\zeta_j)$ we set `drop-B=1e-4`, and `drop-S=1e-2`. Although the flat MPI

[¶]An alternative to increase the amount of parallelism without increasing the size of the Schur complement matrices is to assign different quadrature nodes to different groups of processors by exploiting additional levels of MPI parallelism

DOI: 10.1002/nla

Table III. Breakdown of the total time required to compute $\sum_{j=1}^{N_c} S(\zeta_j)^{-1}v$ for a random $v \in \mathbb{C}^s$, where $N_c = 1$ and $[\alpha, \beta] = [(\lambda_{100} + \lambda_{101})/2, (\lambda_{200} + \lambda_{201})/2]$. τ : number of threads per MPI process.

	$p \times \tau = 32$	$p \times \tau = 64$		$p \times \tau = 128$		$p \times \tau = 256$	
	$\tau = 1$	$\tau = 1$	$\tau = 2$	$\tau = 1$	$\tau = 4$	$\tau = 1$	$\tau = 8$
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 $S_G(\zeta_1)^{-1}$	0.28	0.47	0.28	0.58	0.28	0.89	0.28

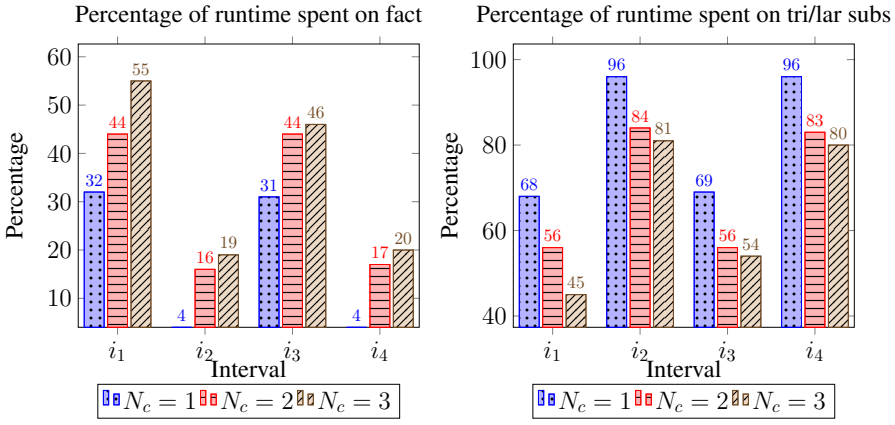


Figure 5. Time breakdown of the CI-M scheme (time spent on factorizations and triangular substitutions) for $N_c = 1$, $N_c = 2$ and $N_c = 3$ quadrature nodes, using the optimal choice of $\hat{r}$ for each different value of N_c, and $p = 128$ MPI processes. For each choice of N_c, we show the breakdown for intervals $i_1 := [(\lambda_{100} + \lambda_{101})/2, (\lambda_{120} + \lambda_{121})/2]$, $i_2 := [(\lambda_{100} + \lambda_{101})/2, (\lambda_{200} + \lambda_{201})/2]$, $i_3 := [(\lambda_{500} + \lambda_{501})/2, (\lambda_{520} + \lambda_{521})/2]$, and $i_4 := [(\lambda_{500} + \lambda_{501})/2, (\lambda_{600} + \lambda_{601})/2]$.

implementation requires less time to compute the MV product with $S(\zeta_1)$, the main advantage of the hybrid implementation is that the cost to apply the preconditioner does not increase as we increase the number of compute cores. When 256 compute cores are available, exploiting 32 MPI processes, each with eight threads, is about three times faster than exploiting 256 MPI processes, each with one thread.

6.4.3. A comparison of the CI-M and DD-FP schemes We now compare the wall-clock times of CI-M and DD-FP to compute the smallest $r = 20$ and all $r = 100$ eigenpairs located inside the intervals $[\alpha, \beta] = [(\lambda_{100} + \lambda_{101})/2, (\lambda_{200} + \lambda_{201})/2]$, and $[\alpha, \beta] = [(\lambda_{500} + \lambda_{501})/2, (\lambda_{600} + \lambda_{601})/2]$. For DD-FP, we kept $p = 32$ fixed, allowing each MPI process to utilize exactly $p/32$ threads.

Table IV lists the best (lowest) wall-clock times achieved by executing CI-M and DD-FP for two different values of N_c, $N_c = 1$ and $N_c = 2$ (setting $N_c > 2$ always led to longer times), and three different values of $\hat{r}$. DD-FP was considerably faster than CI-M in all cases where $r = 20$. When $r = 100$, DD-FP was faster than CI-M for the interval $[\alpha, \beta] = [(\lambda_{100} + \lambda_{101})/2, (\lambda_{200} + \lambda_{201})/2]$, but considerably slower than CI-M for the interval $[\alpha, \beta] = [(\lambda_{500} + \lambda_{501})/2, (\lambda_{600} + \lambda_{601})/2]$. In the latter case, the average time required to solve for a right-hand side by MUMPS was much lower than the amount of time required by preconditioned GMRES in DD-FP.

Note that the wall-clock times of DD-FP (especially its scalability) can generally improve, since, for reasons of fair comparison against CI-M, for MUMPS we used only MPI parallelism, and thus only a fraction of the available compute cores were active in DD-FP when we applied the preconditioner $S_G(\zeta_j)$.

Figure 5 plots the time breakdown of CI-M if we focus on its two main computational procedures, i.e., the amount of time spent on factorizations and triangular substitutions. Results shown are for the optimal choice $\hat{r}$ for each different value of N_c shown, and for $p = 128$ MPI processes. The

Table IV. Best (lowest) wall-clock times achieved by executing CI-M and DD-FP for $N_c = 1$ and $N_c = 2$. For CI-M we kept $\tau = 1$ fixed, while for DD-FP we kept $p = 32$ fixed. Variable "Its" denotes the total number of outer iterations performed by CI-M and DD-FP.

	Its		$p \times \tau = 64$		$p \times \tau = 128$		$p \times \tau = 256$	
	$N_c = 1$	$N_c = 2$	CI-M	DD-FP	CI-M	DD-FP	CI-M	DD-FP
$[\alpha, \beta] \equiv [\lambda_{101}, \lambda_{120}]$								
$\hat{r} = 50$	8	5	1,607.2	324.8	841.4	240.0	685.0	217.9
$\hat{r} = 100$	6	4	2,073.9	473.1	1,092.1	353.2	875.2	313.8
$\hat{r} = 39$	8	5	1,420.6	265.5	741.6	194.8	609.1	166.8
$[\alpha, \beta] \equiv [\lambda_{501}, \lambda_{520}]$								
$\hat{r} = 50$	9	5	1,723.9	1,029.1	904.3	777.4	732.5	685.9
$\hat{r} = 100$	5	4	1,840.5	1,140.3	966.9	862.3	780.0	781.2
$\hat{r} = 39$	9	5	1,492.9	808.9	780.4	609.5	638.5	541.6
$[\alpha, \beta] \equiv [\lambda_{101}, \lambda_{200}]$								
$\hat{r} = 200$	14	5	6,013.9	3,141.9	3,185.4	2,389.6	2,510.1	2,105.4
$\hat{r} = 300$	9	4	6,942.3	3,030.7	3,662.1	2,304.9	2,814.3	2,072.7
$\hat{r} = 236$	10	5	6,179.6	2,652.6	3,294.9	1,989.3	2,547.1	1,766.4
$[\alpha, \beta] \equiv [\lambda_{501}, \lambda_{600}]$								
$\hat{r} = 200$	12	5	6,013.9	13,373.4	3,185.8	10,195.8	2,510.2	9,447.2
$\hat{r} = 400$	7	3	6,950.2	15,596.3	3,664.1	11,892.2	2,876.2	10,564.3
$\hat{r} = 166$	13	5	5,220.4	11,954.5	2,759.9	9,114.0	2,178.1	8,444.6

average (per quadrature node) factorization time required by MUMPS was 679.02, 332.11, and 298.43 seconds, for 64, 128, and 256 MPI processes, respectively. Combining the latter with the results in Table IV, we observe that for the values of N_c reported in this section, the amount of time spent on linear system solutions generally dominates the wall-clock times. Similar observations also hold for DD-FP.

6.4.4. Exploiting additional levels of MPI parallelism One of the main advantages of contour integration eigensolvers is their ability to take advantage of additional levels of MPI parallelism, e.g. by distributing different quadrature nodes and/or right-hand sides to different groups of MPI processes. The latter can be achieved by organizing the set of available MPI processes in a 2D formation, and restricting MPI processes within the same column subgrid to perform computations related to either a fraction of the N_c quadrature nodes, or a fraction of the $\hat{r}$ right-hand sides.

We repeated the experiments in Section 6.4.3, this time organizing the MPI processes in various 2D grid formations. As before, the maximum number of MPI processes was set to 256. For CI-M we tried four different 2D formations; 64×1, 64×2, 64×3, and 64×4. Similarly, for DD-FP, we considered the 2D formations 32×1, 32×2, 32×3, and 32×4, allowing each MPI process to utilize $\tau = 2$ computational threads. For DD-FP, we considered a single quadrature node ($N_c = 1$) and assigned different right-hand sides to different column subgrids of MPI processes (thus each column subgrid of MPI processes was responsible for only a fraction of the $\hat{r}$ right-hand sides in each iteration in DD-FP). For CI-M, we tested two options. First, we considered CI-M with $N_c = 1$ and $N_c = 2$, and assigned different right-hand sides to different column subgrids of MPI processes (thus, similarly to DD-FP, each column subgrid of MPI processes was responsible for all N_c quadrature nodes but only a fraction of the $\hat{r}$ right-hand sides in each iteration in CI-M). We denote this option by CI-M1. Note that CI-M1 is limited only to scenarios where each column subgrid of MPI processes has enough memory to store all N_c matrix factorizations. Second, we considered CI-M with $N_c = 4$ quadrature nodes and assigned the different quadrature nodes to different column subgrids of MPI processes (thus each separate column subgrid of MPI processes was responsible for all $\hat{r}$ right-hand sides, but for one/a few of the N_c quadrature node(s) only). We denote this option by CI-M2.

DOI: 10.1002/nla

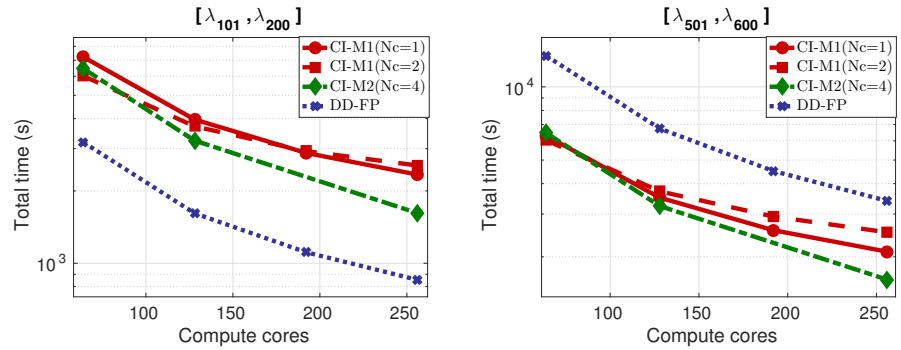


Figure 6. Wall-clock times of CI-M and DD-FP to compute all $r = 100$ eigenpairs inside the intervals $[\alpha, \beta] = [(\lambda_{100} + \lambda_{101})/2, (\lambda_{200} + \lambda_{201})/2]$ and $[\alpha, \beta] = [(\lambda_{500} + \lambda_{501})/2, (\lambda_{600} + \lambda_{601})/2]$ when a 2D grid of MPI processesis exploited. The number of right-hand sides was set to $\hat{r} = 200$.

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

Matrix	n	nnz	$[\alpha, \beta]$	r
$Ge_{99}H_{100}$	112,985	8,451,295	$[-0.65, -0.0096]$	250
$Si_{41}Ge_{41}H_{72}$	185,639	15,011,265	$[-0.64, -0.0028]$	218
$Si_{87}H_{76}$	240,369	10,661,631	$[-0.66, -0.0300]$	213

Figure 6 plots the wall-clock times of CI-M and DD-FP when two levels of MPI parallelism are considered. For simplicity, we considered only the case $\hat{r} = 200$. Exploiting a 32×4 2D grid, DD-FP computed all $r = 100$ eigenpairs inside the intervals $[\alpha, \beta] = [(\lambda_{100} + \lambda_{101})/2, (\lambda_{200} + \lambda_{201})/2]$ and $[\alpha, \beta] = [(\lambda_{500} + \lambda_{501})/2, (\lambda_{600} + \lambda_{601})/2]$, in less than 900 and 3,500 seconds, respectively. The latter wall-clock times constitute a considerable improvement over those obtained by the 1D grid of MPI processes reported in Table IV. For CI-M1, the wall-clock time improvement over the 1D grid of MPI processes was not as pronounced as in DD-FP due to the overhead introduced by the factorizations of matrices $A - \zeta_j I$, $j = 1, \ldots, N_c$. On the other hand, increasing N_c and distributing the quadrature nodes, as in CI-M2, seems to be a more efficient choice when a larger number of computational resources becomes available. Additional details on the performance of FEAST and related approaches when multiple levels of MPI parallelism are considered can be found in [3, 27].

6.5. The PARSEC matrix collection

Our third set of experiments consists of a few matrices originating from applications in Electronic Structure Calculations. The matrices of interest (Hamiltonians) were generated using the PARSEC software package [17], and can be found in the University of Florida Sparse Matrix Collection [15].[‖] Details on the size of the matrices, as well as the interval of interest determined by the Density Functional Theory application, are listed in Table V.

The number of nonzero entries of each Hamiltonian is quite large, a consequence of the high-order discretization used, as well as the addition of a (dense) 'non-local' term. Together with the 3D nature of the problem, this leads to a large number of interface variables, challenging the practicality of direct linear system solvers. In order to increase the efficiency of contour integration eigensolvers for such problems, we consider the replacement of direct solvers by preconditioned iterative solvers. Throughout this section we will only consider block-Jacobi preconditioners

$$S_{BJ}(\zeta_j) = \texttt{bdiag}(S_1(\zeta_j), \ldots, S_p(\zeta_j)), \quad j = 1, \ldots, N_c. \tag{29}$$

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

DOI: 10.1002/nla

Table VI. Time elapsed to perform the N_c LU matrix factorizations $A - \zeta_j I,\ j = 1, \ldots, N_c$ in CI-M versus time elapsed to form and factorize the block-Jacobi preconditioner in DD-FP.

	$p = 4$		$p = 8$		$p = 16$		$p = 32$	
	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
$N_c = 3$	1,265.9	86.3	1,089.4	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	1,442.5	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	2,853.9	3.82	1,127.4	1.61

Table VII. 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 $v \in \mathbb{C}^n$ denotes a random complex 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
$Ge_{99}H_{100}$								
$N_c = 1$	0.7	5.1	0.7	1.7	0.4	0.6	0.3	0.3
$N_c = 2$	1.5	13.1	1.4	3.2	0.8	1.7	0.7	0.5
$N_c = 3$	2.3	33.3	1.9	11.8	1.1	4.1	1.0	1.2
$Si_{41}Ge_{41}H_{72}$								
$N_c = 1$	1.8	7.5	1.2	3.7	0.7	1.0	0.7	0.5
$N_c = 2$	X	32.8	2.5	12.1	1.4	3.5	1.4	0.8
$N_c = 3$	X	61.3	X	31.2	2.1	8.6	2.1	2.1
$Si_{87}H_{76}$								
$N_c = 1$	X	15.0	1.6	4.3	1.3	0.9	0.9	0.4
$N_c = 2$	X	50.2	X	14.0	2.8	3.3	1.9	0.8
$N_c = 3$	X	120.5	X	34.8	4.0	7.5	2.7	2.0

Table VI lists 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 form and factorize the block-Jacobi preconditioner for all $i = 1, \ldots, p$, and $j = 1, \ldots, N_c$ (DD-FP). We report times obtained for a varying number of MPI processes (subdomains). A "X" flag under the CI-M scheme implies that not all N_c matrix factorizations could fit in the memory allocated by each MPI process.

Table VII lists the time elapsed to solve all N_c linear systems by the CI-M and DD-FP schemes for a random right-hand side $v \in \mathbb{C}^n$, i.e, $\sum_{j=1}^{N_c}(A - \zeta_j I)^{-1}v$. For lower values of p, the DD-FP scheme is not competitive, since the cost to apply the block-Jacobi preconditioner is quite high in this case. However, as p increases, the time to solve a linear system by a preconditioned iterative method drops dramatically (we note that the number of iterations is only slightly increased as p increases). Moreover, increasing the value of N_c 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, owed to the fact that iterative solvers are sensitive to the magnitude of the complex part of each quadrature node (see also the discussion in Section 6.4.1).

Figure 7 plots the maximum residual norm of the approximation of the eigenpairs located inside the interval $[\alpha, \beta]$ as a function of the number of outer iterations performed by DD-FP.

DOI: 10.1002/nla

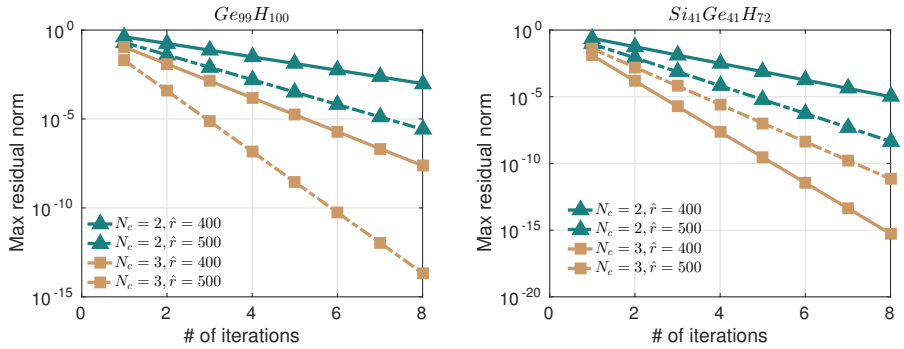


Figure 7. Maximum residual norm of the approximation of the eigenpairs inside the interval $[\alpha, \beta]$, as a function of the number of outer iterations performed by DD-FP. Results are shown for $N_c = 2$, $N_c = 3$, and $\hat{r} = 400$, $\hat{r} = 500$. Solid lines correspond to $\hat{r} = 400$, while dashed lines correspond to $\hat{r} = 500$. Left: $Ge_{99}H_{100}$. Right:$Si_{41}Ge_{41}H_{72}$.

Table VIII. Wall-clock times of CI-M and DD-FP to compute all eigenpairs located inside the intervals $[\alpha, \beta]$ reported in Table V (we set $p = 32$). "Its" denotes the total number of outer iterations performed by DD-FP and CI-M.

	$\hat{r} = 400$			$\hat{r} = 500$		
	Its	CI-M	DD-FP	Its	CI-M	DD-FP
$Ge_{99}H_{100}$						
$N_c = 2$	22	5,990	4,675	9	4,164	3,198
$N_c = 3$	12	3,787	4,438	5	2,750	3,091
$Si_{41}Ge_{41}H_{72}$						
$N_c = 2$	13	7,651	4,392	8	5,996	3,410
$N_c = 3$	5	4,806	4,287	6	6,865	6,360
$Si_{87}H_{76}$						
$N_c = 2$	15	12,059	4,593	10	10,172	3,852
$N_c = 3$	5	6,467	3,960	6	9,114	5,915

Finally, Table VIII reports the total wall-clock time required by CI-M and DD-FP to compute all sought eigenpairs for the case $p = 32$. DD-FP was faster** than CI-M for almost all different combinations of N_c and $\hat{r}$ tested, due to the avoidance of the costly matrix factorizations in CI-M and its lower timings to perform the required linear system solutions.

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 a flexible implementation of the domain decomposition framework in the context of contour integral-based methods. When a direct solver is used for the Schur complement linear systems, DD-FP is 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

**DD-FP also required far less memory than CI-M.

performance of the iterative scheme used for solving the linear systems. In particular, the method can be vastly superior than FEAST with a direct solver 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 a distributed block GMRES linear system solver to solve the complex linear systems with the Schur complement matrices in DD-FP and DD-PP. Another interesting path would be to further study the performance of DD-FP and DD-PP when additional levels of distributed and shared memory parallelism are exploited.

ACKNOWLEDGEMENT

We are grateful to the referees for their comments which led to several improvements of this paper, and to the Minnesota Supercomputing Institute (MSI) at the University of Minnesota for providing resources that contributed to the research results reported within this paper (URL: www.msi.umn.edu). We thank Ruipeng Li, Yuanzhe Xi, Geoffrey Dillon, and Efstratios Gallopoulos 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. Hasan Metin Aktulga, Lin Lin, Christopher Haine, Esmond G. Ng, and Chao Yang. Parallel eigenvalue calculation based on multiple shiftinvert Lanczos and contour integral based spectral projection method. *Parallel Computing*, 40(7):195 – 212, 2014.
4. 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(1):15–41, 2001.
5. Junko Asakura, Tetsuya Sakurai, Hiroto Tadano, Tsutomu Ikegami, and Kinji Kimura. A numerical method for nonlinear eigenvalue problems using contour integrals. *JSIAM Letters*, 1:52–55, 2009.
6. Anthony P. Austin and Lloyd N. Trefethen. Computing eigenvalues of real symmetric matrices with rational filters in real arithmetic. *SIAM Journal on Scientific Computing*, 37(3):A1365–A1387, 2015.
7. S. Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997.
8. Marc Van Barel. Designing rational filter functions for solving eigenvalue problems by contour integration. *Linear Algebra and its Applications*, 502:346 – 365, 2016.
9. Marc Van Barel and Peter Kravanja. Nonlinear eigenvalue problems and contour integrals. *Journal of Computational and Applied Mathematics*, 292:526 – 540, 2016.
10. C. Bekas and Y. Saad. Computation of smallest eigenvalues using spectral Schur complements. *SIAM J. Sci. Comput.*, 27:458–481, 2006.
11. J. K. Bennighof and R. B. Lehoucq. An automated multilevel substructuring method for eigenspace computation in linear elastodynamics. *SIAM J. Sci. Comput.*, 25:2084–2106, 2004.
12. Wolf-Jürgen Beyn. An integral method for solving nonlinear eigenvalue problems. *Linear Algebra and its Applications*, 436(10):3839 – 3863, 2012.
13. L. M. Carvalho, L. Giraud, and P. Le Tallec. Algebraic two-level preconditioners for the Schur complement method. *SIAM Journal on Scientific Computing*, 22(6):1987–2005, 2001.
14. Timothy A. Davis. Algorithm 832: UMFPACK v4.3—an unsymmetric-pattern multifrontal method. *ACM Trans. Math. Softw.*, 30(2):196–199, June 2004.
15. Timothy A. Davis and Yifan Hu. The university of Florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1):1:1–1:25, December 2011.
16. James W. Demmel. *Applied Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.
17. L. Kronik et al. PARSEC–the pseudopotential algorithm for real-space electronic structure calculations: recent advances and novel applications to nano-structures. *Phys. Status Solidi (B)*, 243(5):1063–1079, 2006.
18. S. Balay et al. PETSc users manual. Technical Report ANL-95/11 - Revision 3.6, Argonne National Laboratory, 2015.
19. S. Balay et al. PETSc Web page. http://www.mcs.anl.gov/petsc, 2015.
20. Martin Galgon, Lukas Krämer, Jonas Thies, Achim Basermann, and Bruno Lang. On the parallel iterative solution of linear systems arising in the FEAST algorithm for computing inner eigenvalues. *Parallel Comput.*, 49(C):153–163, November 2015.
21. Luc Giraud, Azzam Haidar, and Yousef Saad. Sparse approximations of the Schur complement for parallel algebraic hybrid solvers in 3D, 2010.

(2010)
DOI: 10.1002/nla

22. 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(4):A2100–A2122, 2015.
23. Vassilis Kalantzis, Costantinos Bekas, Alessandro Curioni, and Efstratios Gallopoulos. Accelerating data uncertainty quantification by solving linear systems with multiple right-hand sides. *Numerical Algorithms*, 62(4):637–653, 2013.
24. Vassilis Kalantzis, Ruipeng Li, and Yousef Saad. Spectral Schur complement techniques for symmetric eigenvalue problems. *Electronic Transactions on Numerical Analysis*, 45:305–329, 2016.
25. Vassilis Kalantzis, Yuanzhe Xi, and Yousef Saad. Beyond AMLS: Domain decomposition with rational filtering. *arXiv preprint arXiv:1711.09487*, 2017.
26. G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.
27. James Kestyn, Vasileios Kalantzis, Eric Polizzi, and Yousef Saad. PFEAST: a high performance sparse eigenvalue solver using distributed-memory linear solvers. In *High Performance Computing, Networking, Storage and Analysis, SC16: International Conference for*, pages 178–189. IEEE, 2016.
28. James Kestyn, Eric Polizzi, and Ping Tak Peter Tang. FEAST eigensolver for non-hermitian problems. *SIAM Journal on Scientific Computing*, 38(5):S772–S799, 2016.
29. Lukas KräMer, Edoardo Di Napoli, Martin Galgon, Bruno Lang, and Paolo Bientinesi. Dissecting the FEAST algorithm for generalized eigenproblems. *J. Comput. Appl. Math.*, 244:1–9, May 2013.
30. 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, pages 533–544, Berlin, Heidelberg, 2013. Springer-Verlag.
31. Ruipeng Li, Yuanzhe Xi, Eugene Vecharynski, Chao Yang, and Yousef Saad. A thick-restart Lanczos algorithm with polynomial filtering for hermitian eigenvalue problems. *SIAM Journal on Scientific Computing*, 38(4):A2512–A2534, 2016.
32. 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(2):110–140, June 2003.
33. Zhongze Li, Yousef Saad, and Masha Sosonkina. pARMS: a parallel version of the algebraic recursive multilevel solver. *Numerical Linear Algebra with Applications*, 10(5-6):485–509, 2003.
34. S.H. Lui. Domain decomposition methods for eigenvalue problems. *Journal of Computational and Applied Mathematics*, 117(1):17 – 34, 2000.
35. F. Pellegrini. SCOTCH *and* LIBSCOTCH *5.1 User's Guide*. INRIA Bordeaux Sud-Ouest, IPB & LaBRI, UMR CNRS 5800, 2010.
36. 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(2):C139–C162, 2014.
37. Bernard Philippe and Yousef Saad. On correction equations and domain decomposition for computing invariant subspaces. *Computer Methods in Applied Mechanics and Engineering*, 196(8):1471 – 1483, 2007.
38. Eric Polizzi. Density-matrix-based algorithm for solving eigenvalue problems. *Phys. Rev. B*, 79:115–112, Mar 2009.
39. 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*, pages 631–643, May 2012.
40. Y. Saad. *Numerical Methods for Large Eigenvalue Problems*. Society for Industrial and Applied Mathematics, 2011.
41. Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986.
42. Y. Saad and B. Suchomel. ARMS: an algebraic recursive multilevel solver for general sparse linear systems. *Numerical Linear Algebra with Applications*, 9(5):359–378, 2002.
43. Yousef Saad and Maria Sosonkina. Distributed Schur complement techniques for general sparse linear systems. *SIAM Journal on Scientific Computing*, 21(4):1337–1356, 1999.
44. Tetsuya Sakurai and Hiroshi Sugiura. A projection method for generalized eigenvalue problems using numerical integration. *Journal of Computational and Applied Mathematics*, 159(1):119 – 128, 2003.
45. Tetsuya Sakurai and Hiroto Tadano. CIRR: a rayleigh-ritz type method with contour integral for generalized eigenvalue problems. *Hokkaido Math. J.*, 36(4):745–757, 11 2007.
46. V. Simoncini and E. Gallopoulos. Convergence properties of block GMRES and matrix polynomials. *Linear Algebra and its Applications*, 247:97 – 119, 1996.
47. 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.
48. 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) edition, 1998.
49. Ping T. P. Tang and E. Polizzi. FEAST as a subspace iteration eigensolver accelerated by approximate spectral projection. *SIAM Journal on Matrix Analysis and Applications*, 35(2):354–390, 2014.
50. Andrea Toselli and Olof Widlund. *Domain decomposition methods: algorithms and theory*, volume 3. Springer, 2005.
51. Shen Wang, Xiaoye S. Li, François-Henry Rouet, Jianlin Xia, and Maarten V. De Hoop. A parallel geometric multifrontal solver using hierarchically semiseparable structure. *ACM Trans. Math. Softw.*, 42(3):21:1–21:21, May 2016.
52. Yuanzhe Xi and Yousef Saad. Computing partial spectra with least-squares rational filters. *SIAM Journal on Scientific Computing*, 38(5):A3020–A3045, 2016.
53. Guojian Yin, Raymond H. Chan, and Man-Chung Yeung. A feast algorithm with oblique projection for generalized eigenvalue problems. *Numerical Linear Algebra with Applications*, 24(4), 2017.

(2010)
DOI: 10.1002/nla