SPECTRAL SCHUR COMPLEMENT TECHNIQUES FOR SYMMETRIC EIGENVALUE PROBLEMS*

VASSILIS KALANTZIS †, RUIPENG LI †, AND YOUSEF SAAD †

Abstract. This paper presents a Domain Decomposition-type method for solving real symmetric (or Hermitian complex) eigenvalue problems in which we seek all eigenpairs in an interval $[\alpha, \beta]$, or a few eigenpairs next to a given real shift ζ. A Newton-based scheme is described whereby the problem is converted to one that deals with the interface nodes of the computational domain. This approach relies on the fact that the inner solves related to each local subdomain are relatively inexpensive. This Newton scheme exploits spectral Schur complements and these lead to so-called eigen-branches, which are rational functions whose roots are eigenvalues of the original matrix. Theoretical and practical aspects of domain decomposition techniques for computing eigenvalues and eigenvectors are discussed. A parallel implementation is presented and its performance on distributed computing environments is illustrated by means of a few numerical examples.

Key words. Domain decomposition, Spectral Schur complements, Eigenvalue problems, Newton's method, Parallel computing.

1. Introduction. We are interested in the partial solution of the symmetric eigenvalue problem

$$Ax = \lambda x, \tag{1.1}$$

where A is an $n \times n$ symmetric (or Hermitian complex) matrix and we assume that it is large and sparse. We assume that the eigenvalues $\lambda_i, i = 1, \cdots, \lambda_n$ of A are labeled increasingly. By "partial solution" we mean one of the two following scenarios:

- Find all eigenpairs (λ, x) of A where λ belongs to the sub-interval $[\alpha, \beta]$ of the sprectrum ($[\alpha, \beta] \subseteq [\lambda_1,\ \lambda_n]$).
- Given a shift $\zeta \in \mathbb{R}$ and an integer k, find the eigenpairs (λ, x) of A for which λ is one of the k closest eigenvalues to ζ. A similar problem is the computation of the k eigenpairs of A located immediatly to the right (or to the left) of the given shift ζ.

The interval $[\alpha, \beta]$ can be located anywhere inside the region $[\lambda_1, \lambda_n]$. When $\alpha := \lambda_1$ or $\beta := \lambda_n$, we will refer to the eigenvalue problem as *extremal*, otherwise we will refer to it as *interior*.

It is typically easier to solve extremal eigenvalue problems than interior ones. Methods such as the Lanczos algorithm [15] and its more sophisticated practical variants such as the Implicitly Restarted Lanczos (IRL) [16], the closely related Thick-restart Lanczos [29, 30], the method of trace minimization [25], its closely related Jacobi-Davidson [27] are powerful methods for solving eigenvalue problems associated with extremal eigenvalues. However, these methods become expensive for interior eigenvalue problems, typically requiring a large number of matrix-vector products or the use of a shift-and-invert strategy to achieve convergence.

A standard approach for solving interior eigenvalue problems is the shift-and-invert technique where A is replaced by $(A-\sigma I)^{-1}$. By this transformation, eigenvalues of A closest to σ become extremal ones for $(A - \sigma I)^{-1}$ and a projection method

* The work of V. Kalantzis and Y. Saad was supported 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 DE-SC0008877. The work of R. Li was supported by the National Science Foundation under grant NSF/DMS-1216366.

†Address: Computer Science & Engineering, University of Minnesota, Twin Cities. {kalantzi,rli,saad} @cs.umn.edu

of choice, be it subspace iteration or a Krylov-based approach, will converge (much) faster. However, a factorization is now necessary and this seriously limits the size and type of problems that can be efficiently solved by shift-and-invert techniques. For example, matrix problems that stem from discretizations of Partial Differential Equations on 3D computational domains are known to generate a large amount of fill-in. An alternative for avoiding the factorization of $(A - \sigma I)$ is to exploit polynomial filtering which essentially consists of replacing $(A-\sigma I)^{-1}$ by a polynomial in A, $\rho(A)$. The goal of the polynomial is to to dampen eigenvalues outside the interval of interest. Such polynomial filtering methods can be especially useful for large interior eigenvalue problems where many eigenvalues are needed, see [9]. Their disadvantage is that they are sensitive to uneven distributions of the eigenvalues and the degree of the polynomial might have to be selected very high in some cases. Recently, contour integration methods, like the FEAST method [22] or the method of Sakurai-Sugiura [24], have gained popularity. The more robust implementations of these utilize direct solvers to deal with the complex linear systems that come from numerical quadrature and this again can become expensive for 3D problems. When iterative methods are used instead, then the number of matrix-vector products can be high as in the case of polynomial filtering.

This paper takes a different perspective from all the approaches listed above by considering a Domain Decomposition (DD) type approach instead. In this framework, the computational domain is partitioned into a number of (non-overlapping) subdomains, which can then be treated independently, along with an interface region that accounts for the coupling among the subdomains. The problem on the interface region is *non-local*, in the sense that communication among the subdomains is required. The advantage of Domain Decomposition-type methods is that they naturally lend themselves to parallelization. Thus, a DD approach starts by determining the part of the solution that lies on the interface nodes. The original eigenvalue problem is then recast into an eigenvalue problem that is to be solved only on the interface nodes, by exploiting spectral Schur complements. This converts the original large eigenvalue problem into a smaller but nonlinear eigenvalue problem. This problem is then solved by a Newton iteration.

The idea of using Domain Decomposition for eigenvalue problems is not new. Though not formulated in the framework of DD, the paper by Abramov and Chishov [28] is the earliest we know that introduced the concept of Spectral Schur complements. Other earlier publications describing approaches that share some common features with our work can be found in [17, 18, 13, 21]. The articles [17, 18] establish some theory when the method is viewed from a Partial Differential Equations viewpoint. The paper [13] also resorts to Spectral Schur complements, but it is not a domain decomposition approach. Rather, it exploits a given subspace, on which the Schur complement is based, to extract approximate eigenpairs. A well-known example of the Domain Decomposition class of methods is the Automated MultiLevel Substructuring method (AMLS) [4] for approximating the lowest eigenvalues of a matrix and our approach can be viewed as an extension of AMLS.

The primary goal of this paper is to further extend current understanding of domain decomposition methods for eigenvalue problems, as well as to develop practical related algorithms. As pointed out earlier, Domain Decomposition goes hand-in-hand with a parallel computing viewpoint and we implemented the proposed scheme in distributed computing environments by making use of the PETSc framework [2].

The paper is organized as follows: Section 2 discusses background concepts on Domain Decomposition and distributed eigenvalue problems. Section 3 proposes a Newton scheme for computing a single eigenpair of A and presents some analysis. Section 4 discusses a generalization to the case of computing multiple eigenpairs of A. Section 5 discusses our parallel implementation within the Domain Decomposition framework and Section 6 presents numerical experiments on both serial and distributed environments. Finally, a conclusion is given in Section 7.

2. Background: Distributed eigenvalue problems. In a Domain Decomposition framework, we typically begin by subdividing the problem into p parts with the help of a graph partitioner [23, 11, 5, 20, 6, 14]. Generally, this consists of assigning sets of variables to subdomains. Figure 2.1 shows a common way of partitioning a graph, where vertices are assigned to subdomains or partitions. A 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. In this situation some edges are cut between domains and so this way of partitioning a graph is also known as edge-separator partitioning.

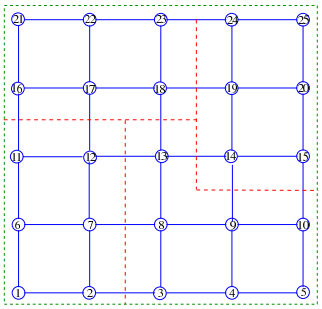

FIG. 2.1. *A classical way of partitioning a graph.*

Once the matrix is partitioned, three types of unknowns appear: (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. Local points in each subdomain are reordered so that the interface points are listed after the interior points. Thus, each local piece x_i of the eigenvector is split into two parts: the subvector u_i of internal vector components followed by the subvector y_i of local interface vector components. When block partitioned according to this splitting, the equation $(A - \lambda I)x = 0$ can be written locally as

$$\underbrace{\begin{pmatrix} B_i - \lambda I & E_i \\ E_i^T & C_i - \lambda I \end{pmatrix}}_{A_i} \underbrace{\begin{pmatrix} u_i \\ y_i \end{pmatrix}}_{x_i} + \begin{pmatrix} 0 \\ \sum_{j \in N_i} E_{ij} y_j \end{pmatrix} = 0. \tag{2.1}$$

Here, N_i is the set of indices for subdomains that are neighbors to the subdomain i. 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. The result of this multiplication affects only local interface equations, which is indicated by a zero in the top part of the second term of the left-hand side of (2.1).

2.1. The interface and Spectral Schur complement matrices. The local equations in (2.1) are naturally split in two parts: the first part (represented by the term $A_i x_i$) involves only local variables and the second contains couplings between these local variables and external interface variables. The second row of the equations in (2.1) is

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

This couples all the interface variables with the local (interior) variable u_i. 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 gathered into the following matrix C

$$C = \begin{pmatrix} C_1 & E_{12} & \dots & E_{1p} \\ E_{21} & C_2 & \dots & E_{2p} \\ \vdots & & \ddots & \vdots \\ E_{p1} & E_{p2} & \dots & C_p \end{pmatrix}. \tag{2.2}$$

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:

$$\underbrace{\begin{pmatrix} B_1 & & & \dots & E_1 \\ & B_2 & & \dots & E_2 \\ \vdots & & \ddots & & \vdots \\ & & & B_p & E_p \\ E_1^T & E_2^T & \dots & E_p^T & C \end{pmatrix}}_{PAP^T} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_p \\ y \end{pmatrix} = \lambda \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_p \\ y \end{pmatrix}. \tag{2.3}$$

The coefficient matrix of the system (2.3) is of the form

$$A = \begin{pmatrix} B & E \\ E^T & C \end{pmatrix} \tag{2.4}$$

where we have kept the same symbol A to represent the unpermuted matrix in (1.1). We will assume that each subdomain i has d_i *interior* variables and s_i interface variables, i.e., the length of u_i is d_i and that of y_i is s_i. We will denote by s the size of y so $s = s_1 + s_2 + \cdots + s_p$. With this notation, each E_i is a matrix of size d_i by s, though most of its columns are zero. An illustration for 4 subdomains is shown in Figure 2.2.

2.2. Spectral Schur complements. Schur complement techniques eliminate interior variables to yield equations associated with the interface variables only. Specifically, we can eliminate the variable u_i from (2.1), which gives $u_i = -(B_i - \lambda I)^{-1} E_i y_i$ and upon substitution in the second equation, we get:

$$S_i(\lambda) y_i + \sum_{j \in N_i} E_{ij} y_j = 0 \tag{2.5}$$

where $S_i(\lambda)$ is the "local" spectral Schur complement

$$S_i(\lambda) = C_i - \lambda I - E_i^T (B_i - \lambda I)^{-1} E_i. \tag{2.6}$$

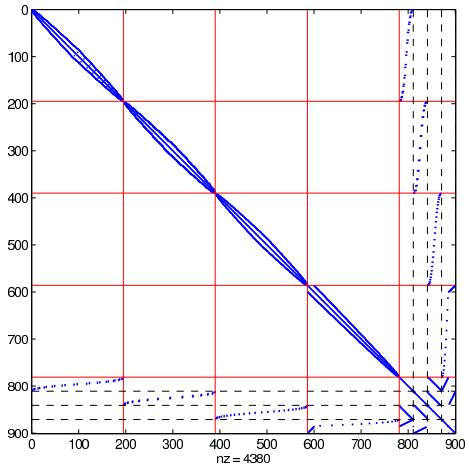

FIG. 2.2. *Laplacian matrix partitioned in $p = 4$ subdomains and reordered according to equation (2.3).*

The equations (2.5) for all subdomains ($i = 1, \ldots, p$) constitute a nonlinear eigenvalue problem, one that involves only the interface unknown vectors y_i. This problem has a natural block structure:

$$\underbrace{\begin{pmatrix} S_1(\lambda) & E_{12} & \ldots & E_{1p} \\ E_{21} & S_2(\lambda) & \ldots & E_{2p} \\ \vdots & & \ddots & \vdots \\ E_{p1} & E_{p,2} & \ldots & S_p(\lambda) \end{pmatrix}}_{S(\lambda)} \underbrace{\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \end{pmatrix}}_{y} = 0 \tag{2.7}$$

The diagonal blocks in this system, the local Schur complement matrices S_i, are dense in general. The off-diagonal blocks E_{ij}, which are identical with those of the local system (2.1), are sparse. Note that the above Spectral Schur complement is nothing but the Spectral Schur complement associated with the decomposition (2.4), i.e., we have:

$$S(\lambda) = C - \lambda I - E^T (B - \lambda I)^{-1} E. \tag{2.8}$$

In other words, (2.7) and (2.8) are identical, the first being more useful for practical purposes and the second more useful for theoretical derivations. Spectral Schur complements were discussed also in [4, 3].

If we can solve the global Schur complement problem (2.7) then the solution to the global eigenproblem (1.1) would be trivially obtained by substituting the y_i's into the first part of (2.1), i.e., by setting for each domain: $u_i = -(B_i - \lambda I)^{-1} E_i y_i$.

3. Solving the spectral Schur complement problem. Our next focus is on the solution of the Spectral Schur Complement problem (2.7). The problem we have is to find a pair $(\sigma, y(\sigma))$ satisfying:

$$S(\sigma) y(\sigma) = 0. \tag{3.1}$$

Then, σ is an eigenvalue of A with $y(\sigma)$ being the bottom part of the associated eigenvector. For an arbitrary value $\sigma \in \mathbb{R}$ (which we will call a shift), we consider the

eigenvalue problem

$$S(\sigma)y(\sigma) = \mu(\sigma)y(\sigma) \tag{3.2}$$

where $\mu(\sigma)$ denotes the eigenvalue of smallest magnitude of $S(\sigma)$. The matrix $S(\sigma)$ has s eigenvalues and they will be denoted as $\mu_i(\sigma),\ i = 1, \ldots, s$ in a sorted (algebraic) ascending order. Each $\mu_i(\sigma)$ is a function of σ which we refer to as the i'th *eigenbranch* of $S(\sigma)$.

The question now becomes how to find a σ for which $S(\sigma)$ is singular. One idea, adopted in [17], is to consider the equation $\det(S(\sigma)) = 0$ but this is not practical for large problems. On the other hand, $S(\sigma)$ is singular exactly when at least one of $\mu_i(\sigma),\ i = 1, \ldots, s$ is zero. With this, the original eigenvalue problem in (1.1) can be reformulated as that of finding a shift σ such that the eigenvalue of smallest magnitude, $\mu(\sigma)$, of $S(\sigma)$ is zero. Thus, $\mu(\sigma)$ is treated as a nonlinear function and we are interested in obtaining a few of its roots, e.g., those located inside a given interval $[\alpha, \beta]$. To find these roots, we would like to exploit a Newton scheme and for this the derivative of the function $\mu(\sigma)$ is needed.

PROPOSITION 3.1. *The function $\mu(\sigma)$ is analytic at any point $\sigma \notin \Lambda(B)$, where $\Lambda(B)$ denotes the spectrum of B. Its derivative at these points is given by*

$$\frac{d\mu(\sigma)}{d\sigma} = \frac{(S'(\sigma)y(\sigma), y(\sigma))}{(y(\sigma), y(\sigma))} = -1 - \frac{\|(B - \sigma I)^{-1}Ey(\sigma)\|_2^2}{\|y(\sigma)\|_2^2}. \tag{3.3}$$

Proof. Differentiating (3.2) we get (the dependence on the variable (σ) is omitted throughout the proof):

$$S'y + Sy' = \mu'y + \mu y'. \tag{3.4}$$

Taking inner products with y yields:

$$(S'y, y) + (Sy', y) = \mu'(y, y) + \mu(y', y) \tag{3.5}$$

Observe that $(Sy', y) = (y', Sy) = (y', \mu y) = \mu(y', y)$. Then the above equality becomes $(S'y, y) = \mu'(y, y)$ which gives the first part of (3.3). The second part is obtained by differentiating $S(\sigma)$ with respect to σ, using expression (2.8):

$$S'(\sigma) = -I - E^T(B - \sigma I)^{-2}E. \tag{3.6}$$

□

Equation (3.6) shows that the matrix $S'(\sigma)$ is negative definite. Note also that the derivative of μ in (3.3) is negative and so the following corollary is immediate.

COROLLARY 3.2. *For any σ located in an interval containing no poles, all eigenbranches $\mu_i(\sigma),\ i = 1, \ldots, s$, are strictly decreasing.*

3.1. An algorithm for computing a single eigenpair. Assume that we are interested in computing a single eigenpair (λ, x), say the one closest to $\zeta \in \mathbb{R}$. Then, a straightforward algorithm based on Newton iteration is as follows.

ALGORITHM 3.1. *Newton-Spectral Schur complement*

1. *Select $\sigma := \zeta$*
2. *Until convergence Do:*
3. *Compute $\mu(\sigma)$ = Smallest eigenvalue in modulus of $S(\sigma)$*
4. *along with the associated unit eigenvector $y(\sigma)$*
5. *Set $\eta := \|(B - \sigma I)^{-1}Ey(\sigma)\|_2$*
6. *Set $\sigma := \sigma + \mu(\sigma)/(1 + \eta^2)$*
7. *End*

Algorithm 3.1 does not specify how to extract the eigenpair $(\mu(\sigma), y(\sigma))$ in Steps 3 and 4 for any σ. All that is needed is the eigenvalue of $S(\sigma)$ closest to zero and its associated eigenvector. This is a perfect setting for a form of inverse iteration. Alternatively, we can use the Lanczos method with partial re-orthogonalization [26] and perform as many steps as needed to compute $(\mu(\sigma), y(\sigma))$. Note that in the latter case, Lanczos will operate on vectors of shorter length (equal to s, the number of interface nodes). In this paper we only consider approximating $(\mu(\sigma), y(\sigma))$ by the inverse iteration approach in which an iterative scheme is used for solving the linear systems. If we were to solve these systems by using an exact factorization of the Schur complement the whole scheme would be quite similar to a form of Rayleigh Quotient Iteration (RQI) whith a DD framework for solving the related linear systems exactly. This is not practical for large 3-D problems because Schur complements can be quite large in these cases.

3.1.1. Analysis of the algorithm. Since Algorithm 3.1 is essentially Newton's method, we expect that if the initial shift σ is "close enough" to an eigenvalue λ, Algorithm 3.1 will converge quadratically to λ. Indeed, $S(\sigma)$ is analytic for all real values except the poles which are the eigenvalues of B [12]. Therefore, each eigenbranch is an analytic branch. By Proposition 3.1, the first derivative of $\mu(\sigma)$ is always non-zero (upper bounded by -1). In addition the second derivative of μ is finite for $\sigma \notin \Lambda(B)$. Therefore, when the scheme converges, it will do so quadratically.

It is interesting to link quantities of the algorithm that are related to the Schur complement with those of the original matrix A. We can think of $y(\sigma)$ as an approximation to the interface part of a global eigenvector. This global approximate eigenvector, which we write in the form $\hat{x}(\sigma) = [\hat{u}(\sigma)^T, y(\sigma)^T]^T$, can be obtained by substituting $y(\sigma)$ into the equation $(A - \sigma I)u = 0$ and exploiting the block structure (2.4) to get:

$$\hat{x}(\sigma) = \begin{bmatrix} -(B - \sigma I)^{-1} E y(\sigma) \\ y(\sigma) \end{bmatrix}. \tag{3.7}$$

By its definition this approximate eigenvector has a special residual. Indeed,

$$(A - \sigma I)\hat{x}(\sigma) = \begin{pmatrix} B - \sigma I & E \\ E^\top & C - \sigma I \end{pmatrix} \begin{pmatrix} -(B - \sigma I)^{-1} E y(\sigma) \\ y(\sigma) \end{pmatrix} = \begin{pmatrix} 0 \\ \mu(\sigma) y(\sigma) \end{pmatrix}. \tag{3.8}$$

In addition, its Rayleigh quotient is equal to the next σ obtained by one Newton step as is stated next.

PROPOSITION 3.3. *Let $\sigma_{j+1} = \sigma_j + \mu(\sigma_j)/(1 + \eta_j^2)$ be the Newton update at the j'th step of Algorithm 3.1, where we set $\eta_j = \|(B - \sigma_j I)^{-1} E y(\sigma_j)\|_2$. Then, $\sigma_{j+1} = \rho(A, \hat{x}(\sigma_j))$, where $\rho(A, \hat{x}(\sigma_j))$ is the Rayleigh Quotient of A associated with the vector $\hat{x}(\sigma_j)$ defined by (3.7).*

Proof. For simplicity, set $\hat{x} \equiv \hat{x}(\sigma_j)$ and assume, without loss of generality, that $\|y(\sigma_j)\| = 1$. We write $\rho(A, \hat{x}) = \sigma_j + \rho(A - \sigma_j I, \hat{x})$. The left term of the right-hand is

$$\rho(A - \sigma_j I, \hat{x}) = \frac{\hat{x}^\top (A - \sigma_j I)\hat{x}}{\hat{x}^\top \hat{x}}, .$$

The expressions (3.7) and (3.8) show that $\hat{x}^\top (A - \sigma_j I)\hat{x} = \mu(\sigma)$ while $\hat{x}^\top \hat{x} = 1 + \eta_j^2$. Thus, $\rho(A - \sigma_j I, \hat{x}) = \mu(\sigma_j)/(1 + \eta_j^2)$ and so $\rho(A, \hat{x}) = \sigma_j + \mu(\sigma_j)/(1 + \eta_j^2) = \sigma_{j+1}$. □

When $\mu(\sigma) = 0$ then σ is an eigenvalue of A. We expect that the closer $\mu(\sigma)$ is to zero, the closer σ should be to an eigenvalue of A. In fact, the relation (3.8) immediately shows that:

$$\frac{\|A\hat{x}(\sigma) - \sigma\hat{x}(\sigma)\|}{\|\hat{x}(\sigma)\|} = \frac{|\mu(\sigma)|}{\sqrt{1+\eta^2}}. \tag{3.9}$$

where $\eta = \|(B - \sigma I)^{-1} E y(\sigma)\|$.

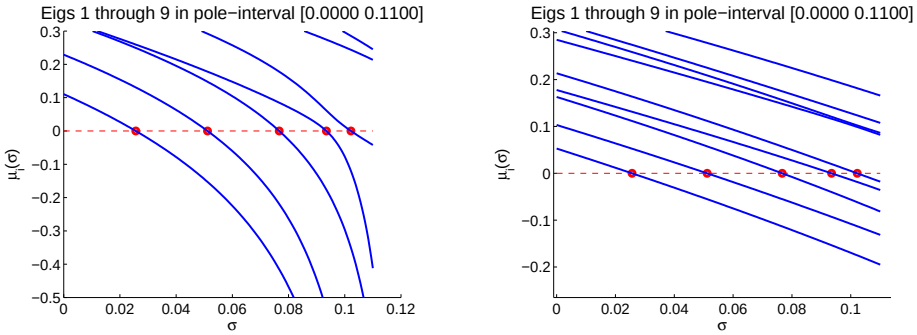

FIG. 3.1. *Eigenbranches $\mu_1(\sigma), \ldots, \mu_9(\sigma)$ in the interval $[0.0, 0.11]$ for a 2D Laplacean. In the left subfigure $p = 4$ while in the right subfigure $p = 16$.*

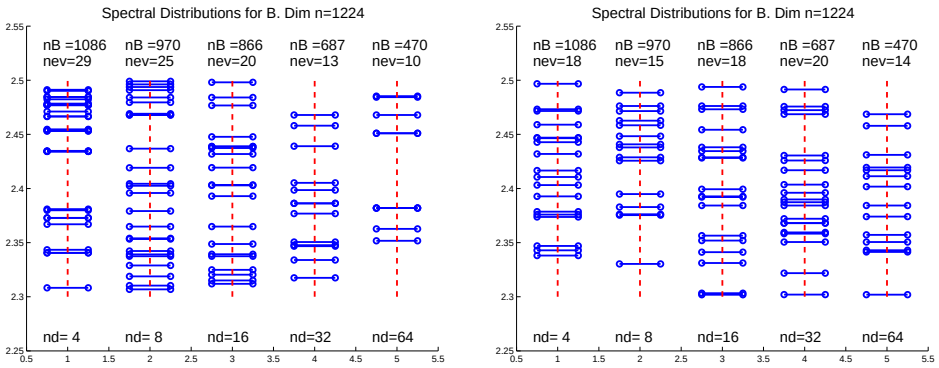

FIG. 3.2. *Eigenvalue Distributions for various $B's$ obtained for a discretized 2D Laplacean of dimension $n = 1224$ for different numbers of subdomains.*

3.1.2. Influence of the number of subdomains. In a Newton scheme, functions that are nearly linear lead to faster convergence. We often observe that the shape of the eigenbranches tend to become more linear as the number p of subdomains increases. This behavior is illustrated in Figure 3.1 with a small example of a 2D discretized Laplacean. We used $p = 4$ and $p = 16$ subdomains and plot the eigenbranches $\mu_1(\sigma), \ldots, \mu_9(\sigma)$ in the interval $[0.0, 0.11]$. Notice the difference in the shape of the eigenbranches: for $p = 16$ the branches are closer to straight lines while for $p = 4$ the poles lie within a shorter distance and the branches tend to bend more.

The shape of the function μ depends on the distance between two consecutive poles. As the number of subdomains increases the size of B decreases thereby reduc-

ing the number of poles. The effect of this is that a given interval $[a,\ b]$ that is under consideration will typically have fewer poles as the number of subdomains increases. An illustration with a standard discretized 2D Laplacean of size $n = 1224$ (corresponding to an 34×36 grid) is shown on the left side of Figure 3.2 where the interval $[a, b]$ is [2.3 2.5]. There is another phenomemon at play in addition to the reduction of the number of eigenvalues of B. As the last two panels ($nd = 32, nd = 64$) show these eigenvalues will also tend to become multiple. For example when $nd = 64$ there are 10 eigenvalues in the interval but a few of these are very close so in effect we have 6 eigenvalues that are numerically distinct. In another instance (grid of size 34×26) we tested, for 64 subdomains there was only one eigenvalue reproduced 10 times (to within 16 digits). One possible explanation for this is that the subdomains are essentially identical and therefore, the subproblems are the same since we are dealing with a pure Laplacean with no variation in the coefficients from one subdomain to the next. To verify this hypothesis we added a strong pseudo-random perturbation to the diagonal of the Lapleacean and repeated the experiment. The result is shown on the right side of Figure 3.2. As can be seen, now the number of poles decreases in an erratic fashion as p increases and the clustering of the eigenvalues is not as pronounced.

4. Computing eigenpairs in an interval. It is possible to extend the framework in Algorithm 3.1, for the computation of multiple eigenpairs, e.g., for searching all eigenpairs inside a given interval $[\alpha, \beta]$, or a few consecutive eigenvalues next to a shift $\zeta \in \mathbb{R}$. The main question is how to select a good starting point for the next unconverged eigenpair of A. For this, we need to take a closer look at the eigenbranches of the spectral Schur complement.

Figure 4.1 shows the relevant eigenvalues of $S(\sigma)$ in the interval $[2.358, 2.4]$ for a sample 2D Laplacean. The separating dashed spikes are eigenvalues of B, i.e., poles of $S(\sigma)$. An interesting property is revealed when observing the eigenvalues across the borderlines. While the matrix $S(\sigma)$ does not exist at a pole, the plot reveals that individual eigenvalues may exist and, *quite interestingly, seem to define differentiable branches accross the poles.*

4.1. Eigenbranches. The above observation can be explained in the following manner. Let $\theta_1,\ \cdots,\ \theta_m$ be the eigenvalues of B with the associated eigenvectors $v_1, \cdots, v_m$. Consider $S(\sigma)$ defined by (2.8), and write it in the following spectral form

$$S(\sigma) = C - \sigma I - E^T(B - \sigma I)^{-1}E = C - \sigma I - \sum_{i=1}^{m} \frac{w_i w_i^T}{\theta_i - \sigma}, \quad \text{with} \quad w_i \equiv E^T v_i. \tag{4.1}$$

The operator $S(\sigma)$ is not formally defined when σ equals one eigenvalue θ_k of B. However, it can be defined on a restricted subspace. Informally, we can say that in this situation $S(\sigma)$ has one infinite eigenvalue and $s - 1$ finite ones. We may be interested in these finite ones and ignore the infinite one. This can be achieved by using, for example, pseudo-inverses. Specifically, when $\sigma = \theta_k$, the linear mapping $S(\sigma)$ can be defined on any subspace that is orthogonal to the span of w_k.

Consider now the eigenvalues $\mu_1(\sigma), \mu_2(\sigma), \cdots, \mu_s(\sigma)$ of $S(\sigma)$ when σ is in between two poles θ_{k-1} and θ_k. As σ moves toward θ_k from the left, one of the eigenvalues, in this case $\mu_1(\sigma)$, will move to $-\infty$. Similarly, as σ moves toward θ_{k-1} from the right, the eigenvalue $\mu_s(\sigma)$ will move to ∞. However, μ_s is perfectly well defined at the next pole and when it crosses to the next panel.

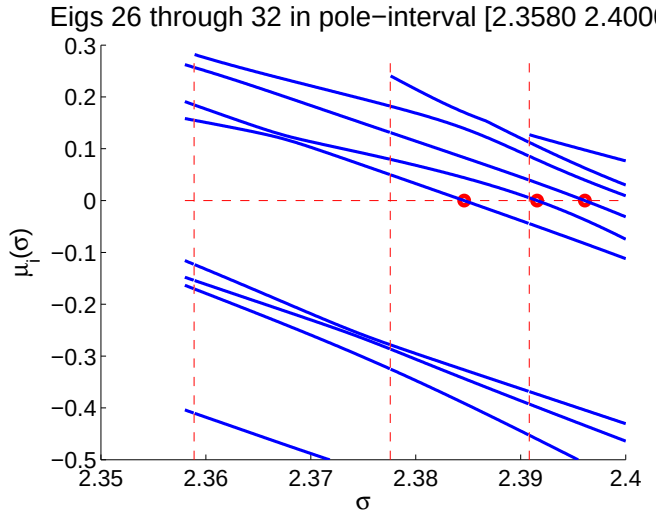

FIG. 4.1. *Eigenbranches $\mu_i(\sigma)$ for $i = 26, \cdots, 32$ obtained from a Domain Decomposition applied to a Laplacean matrix partitioned in 4 subdomains. The eigenvalues are followed on 3 different panels bordered by poles. There is one eigenvalue of A in the second panel and two in the third.*

4.2. The inertia theorem in a domain decomposition framework. The tool of choice for counting the number of eigenvalues of a Hermitian matrix in a given interval is the Sylvester inertia theorem [10]. However, this theorem requires the factorization of the whole matrix A and there are instances when this factorization is too expensive to compute. A sort of distributed version of the theorem can be exploited and it can be obtained from a block factorization of the matrix. Recall that the inertia of a matrix X is a triplet $[\nu_-(X), \nu_0(X), \nu_+(X)]$ consisting of the numbers of of negative, zero, and positive eigenvalues of X, respectively. In the following proposition, the sum of two inertias is defined as the sum of these inertias viewed as vectors of 3 components.

PROPOSITION 4.1. *Let σ be a shift such that $\sigma \notin \Lambda(B)$. Then the inertia of $A - \sigma I$ is the sum of the inertias of $B - \sigma I$ and $S(\sigma)$.*

Proof. Assume without loss of generality that $\sigma = 0$. The result follows immediately by applying the congruence transformation :

$$\begin{pmatrix} I & 0 \\ -E^T B^{-1} & I \end{pmatrix} \begin{pmatrix} B & E \\ E^T & C \end{pmatrix} \begin{pmatrix} I & -B^{-1}E \\ & I \end{pmatrix} = \begin{pmatrix} B & \\ & S(0) \end{pmatrix} \tag{4.2}$$

Since congruences do not alter inertias the inertia of A is the same as that of the block-diagonal matrix at the end of the equation shown above. □

The inertia of $S(\sigma)$ can be computed either by explicitly forming and factorizing $S(\sigma)$ or by using the Lanczcos algorithm and computing all negative eigenvalues. This result can now be utilized to count the number of eigenvalues of A in the interval $[\alpha, \beta]$.

COROLLARY 4.2. *Assume that neither a nor b is an eigenvalue of B and let ν_a be the number of negative eigenvalues of $S(a)$, ν_b the number of non-positive eigenvalues of $S(b)$, and $\mu_{(a,b)}(B)$ the number of eigenvalues of B located in (a, b). Then the number of eigenvalues of A in $[a, b]$ is equal to $\nu_b - \nu_a + \mu_{(a,b)}(B)$.*

Proof. Using the previous proposition, the number of eigenvalues of A that are $\leq b$ is equal to $\nu_-(S(b)) + \nu_-(B - bI) + \nu_0(S(b)) + \nu_0(B - bI)$. By assumption b is

not an eigenvalue of B so $\nu_0(B - bI) = 0$. The number of eigenvalues of A that are $< a$ is equal to $[\nu_-(S(a)) + \nu_-(B - aI)]$. Taking the difference yields,

$$\nu_-(S(b)) + \nu_0(S(b)) - \nu_-(S(a)) + [\nu_-(B - bI) - \nu_-(B - aI)] = \nu_b - \nu_a + \mu_{(a,b)}(B)$$

as desired. □

It is possible to generalize this result to the situation where a or b are eigenvalues of B, interpreting $S(\sigma)$ in the way discussed in subsection 4.1. The result is omitted.

4.3. Branch-hopping algorithm. The discussion above suggests an algorithm for computing all eigenvalues in an interval by selecting the shifts carefully. We start with a shift σ equal to a then iterate as in Algorithm 3.1 until convergence. Once the first eigenvalue has converged we need to catch the next branch of eigenvalues. Since we are moving from left to right we will just select *the next positive eigenvalue of $S(\sigma)$ after the zero eigenvalue $\mu(\sigma)$ which has just converged*. We would then extract the corresponding eigenvector and compute the next σ by Newton's scheme as in Algorithm 3.1.

ALGORITHM 4.1. *Branch hopping Newton-Spectral Schur complement*

0. Given a, b. Select $\sigma = a$
1. While $\sigma < b$
2. Until convergence Do:
3. Compute $\mu(\sigma)$ = Smallest eigenvalue in modulus of $S(\sigma)$
4. If ($|\mu(\sigma)| < tol$)
5. σ and associated eigenvector have converged – save them
6. Obtain $\mu(\sigma)$ = smallest positive eigenvalue of $S(\sigma)$
7. end
8. Obtain unit eigenvector $y(\sigma)$ associated with $\mu(\sigma)$
9. Set $\eta := \|(B - \sigma I)^{-1} E y(\sigma)\|_2$
10. Set $\sigma := \sigma + \mu(\sigma)/(1 + \eta^2)$
11. $\sigma := \sigma + \mu(\sigma)/(1 + \eta^2)$
12. End
13. End

An interesting observation made in our experiments is that when we restart it is often better to compute η not from the eigenvector $y(\sigma)$ obtained in line 8 but the one associated with the eigenvalue in line 3.

As stated earlier, Step 8 is performed with the inverse iteration algorithm with the matrix $S(\sigma)$, in which an iterative method (with or without preconditioning) is invoked to solve the linear systems. A few details will be provided in the numerical experiments section.

Example. In Figure 4.2 we plot the eigenpairs' residual norm obtained by Algorithm 4.1 when searching for the $k = 8$ consecutive eigenvalues (from left to right only) next to a pre-selected shift $\zeta \in \mathbb{R}$ for a 2D Laplacean. We used two different shifts, $\zeta = 2.0$ and $\zeta = 4.15$. The horizontal line visualizes the threshold where the norm of the relative residual is equal to 1e-8. The quadratic convergence of the scheme is evident in both plots where the number of correct digits is roughly doubled at each step. Note also that Algorithm 4.1 provides a good starting point for the next targeted eigenpair.

5. The Branch-hopping Newton scheme in a parallel framework. In a parallel implementation we assume that the matrix A is distributed across a set of p processors, with each processor holding a certain part of its rows as the local

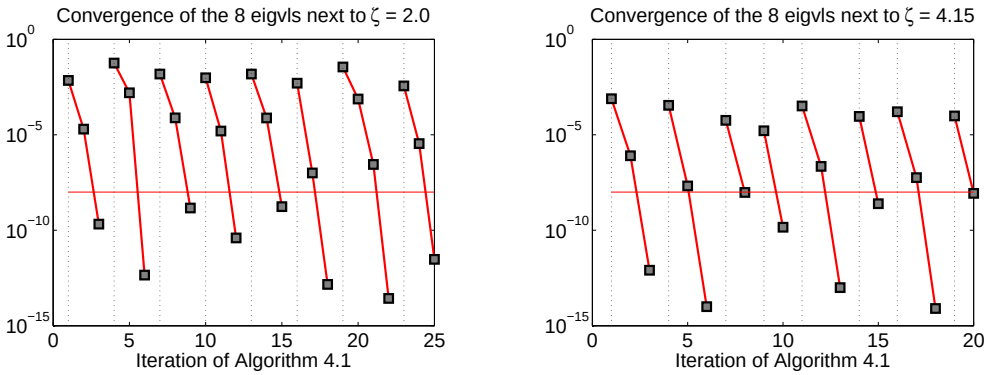

FIG. 4.2. *Residual norm for a few consecutive eigenpairs next to an initial shift* $\zeta \in \mathbb{R}$ *when using Algorithm 4.1. In the left subfigure* $\zeta = 2.0$ *while in the right subfigure* $\zeta = 4.15$.

system (2.1). The smallest eigenvalue $\mu(\sigma)$ of each spectral Schur complement $S(\sigma)$ is computed by a matrix-free method and all computational kernels, like Matrix-Vector (MV) products, are executed in parallel using p different processes each handling one of the p subdomains.

Conceptually, the global spectral Schur complement $S(\sigma)$ in (2.7) is also distributed across the p processors. The matrix-vector product with $S(\sigma)$ is computed as

$$S(\sigma)x = (C - \sigma I)x - E^T(B - \sigma I)^{-1}Ex. \tag{5.1}$$

An important property from the DD method with edge-separation is that the second term on the right-hand side of (5.1) is entirely local, as can be easily seen from the structure of $S(\sigma)$ in (2.7). In summary, the computations involved in (5.1) are:

1. matrix-vector multiplication with $C - \sigma I$ (non-local),
2. matrix-vector multiplication with E and E^T (local),
3. system solution with $B - \sigma I$ (local),

where only the first computation requires communication, between adjacent subdomains. Hence, the communication cost of the matrix-vector product with $S(\sigma)$ is the same as that with C and is point-to-point. The other major communication cost is introduced by the vector dot-products used in the projection method of choice and are of of the all-reduction kind. The local solve with the block-diagonal $(B - \sigma I)$ is carried out by using a sparse direct method and $(B - \sigma I)$ is factored once for each shift σ.

6. Numerical experiments. This section reports on numerical results with the proposed Newton schemes listed in Algorithms 3.1 and 4.1. All numerical experiments were performed on the Itasca Linux cluster at Minnesota Supercomputing Institute. Itasca is an HP Linux cluster with 1,091 HP ProLiant BL280c G6 blade servers, each with two-socket, quad-core 2.8 GHz Intel Xeon X5560 "Nehalem EP" processors sharing 24 GB of system memory, with a 40-gigabit QDR InfiniBand (IB) interconnect. In all, Itasca consists of $8,728$ compute cores and 24 TB of main memory.

The numerical experiments are organized in three different sets. In the first set, we assume that each eigenpair $(\mu(\sigma), y(\sigma))$ in Algorithms 3.1 and 4.1 is computed with the highest possible accuracy in order to study the algorithm theoretically. The second

TABLE 6.1

Number of Newton iterations when computing all eigenvalues inside the interval $[\alpha, \beta]$ by Algorithm 4.1. Different matrix sizes and number of subdomains are used.

		$[\alpha,\beta] := [0, 0.5]$		$[\alpha,\beta] := [2, 2.2]$		$[\alpha,\beta] := [4.1, 4.2]$	
		#Eigvls	It	#Eigvls	It	#Eigvls	It
$\mathbf{n = 21 \times 20 \times 9}$							
# of subdomains (p)	2	15	41	39	85	46	124
	4		26		74		80
	8		32		60		70
	16		32		55		70
$\mathbf{n = 21 \times 20 \times 19}$							
# of subdomains (p)	2	35	60	81	152	117	360
	4		43		130		172
	8		35		116		152
	16		39		96		148
$\mathbf{n = 41 \times 20 \times 19}$							
# of subdomains (p)	2	73	210	156	342	217	424
	4		170		292		314
	8		154		273		310
	16		138		241		300
$\mathbf{n = 41 \times 40 \times 20}$							
# of subdomains (p)	2	155	385	319	703	472	802
	4		354		540		647
	8		296		502		592
	16		270		451		533

set of experiments consists of results obtained in distributed computing environments where we are interested in measuring actual wall-clock timings and $(\mu(\sigma), y(\sigma))$ is only approximately computed. The third set of experiments consists of a brief experimental framework with matrices from electronic structure calculations.

To compute a single eigenpair we use Algorithm 3.1 and to compute a few consecutive eigenpairs, or all eigenpairs inside a given interval, we will use Algorithm 4.1.

6.1. Model problem. The test matrices in this section originate from discretizations of elliptic PDEs on two (and three) dimensional 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 in each corresponding dimension, we obtain a a matrix A, the discretized version of Δ, which is of size $n = n_x n_y n_z$.

6.2. Algorithm validation. This section focuses on the numerical behavior of Algorithms 3.1 and 4.1 and so the results described here were produced by using a MATLAB prototype implementation. In addition, the Lanczos algorithm with partial re-orthogonalization is used to compute $(\mu(\sigma), y(\sigma))$.

Table 6.1 shows the numerical performance of Algorithm 4.1 for a few discretized Laplaceans of size $n \approx 4000$, 8000, 16000 and $n \approx 32000$, when computing all

eigenpairs inside a given interval $[\alpha, \beta]$. The intervals were selected as $[\alpha, \beta] = [0, 0.5]$, $[2, 2.2]$ and $[4, 4.1]$. Note that the last two intervals lie well inside the spectrum. For the purposes of this experiment we enforced a strict requirement for convergence by setting tol=10^{-12}. For each different discretization we also used a varying number of subdomains p. For each interval $[\alpha, \beta]$ we list the number of eigenvalues inside the interval (denoted as "#Eigvls") as well as the total number of Newton iterations in Algorithm 4.1 to compute all eigenvalues (denoted as "It"). We observe that, on average, a few Newton iterations per eigenvalue are enough to compute each eigenpair close to machine precision. Moreover, when using larger values for p, we typically need only one or two Newton steps per eigenpair. This was expected from our observations in Section 3, since the shape of the eigenbranches of $S(.)$ is better approximated by a linear function when we use more subdomains and thus Newton's method converges faster. Note that when using a large number of subdomains, the initial guess for the next unconverged eigenpair is usually much more accurate.

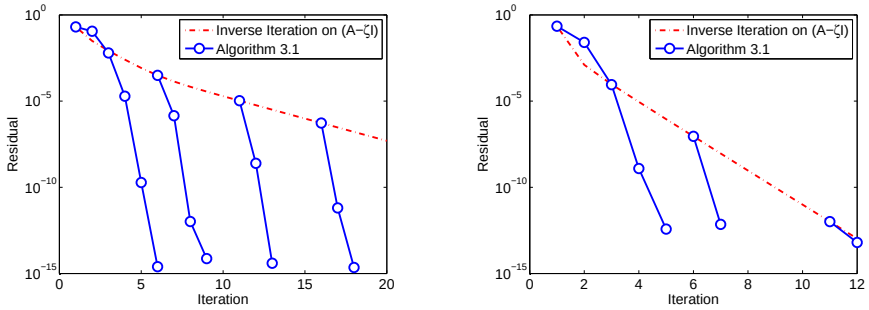

FIG. 6.1. *Convergence behavior for computing the eigenpair closest to ζ when combining inverse iteration and Newton's method. In the left subfigure $\zeta = 0.0$ while in the right subfigure $\zeta = 3.0$.*

Figure 6.1 shows the convergence behavior of Algorithm 3.1 when searching for a single eigenpair closest to ζ, for a Laplacean discretized as $n_x = 11,\ n_y = 10,\ n_z = 9$. In the pre-processing phase, we applied a few steps of inverse iteration with $(A - \zeta I)$ to generate initial guesses of varying accuracy for Algorithm 3.1. The dashed line shows the convergence of inverse iteration on $(A - \zeta I)$ while the circled curves shows the convergence of Algorithm 3.1. Note that the leftmost circled curve stands for Algorithm 3.1 with $\sigma := \zeta$ as the starting shift. As expected when the initial guess is not very accurate, more iterations of Algorithm 3.1 may be needed for convergence. As the initially provided approximation becomes more accurate, one or two iterations of Algorithm 3.1 are usually enough to reduce the residual norm of the approximate eigenpair below 1e-10.

6.3. Distributed computing environments. In this subsection we present results obtained on distributed computing environments, using our parallel implementation of Algorithms 3.1 and 4.1. We focus on parallel execution timings and report results both for extreme and interior eigenvalue problems.

6.3.1. Experimental setup. We implemented and tested three different methods: a) (inexact) inverse iteration applied to A (each time with the appropriate shift), b) Newton's method as described in Algorithms 3.1 and 4.1, and c) a combination of the first two approaches where we first perform a few steps of inverse iteration with A, followed by the Newton's scheme. We chose to compare against inverse iteration

mainly because its simplicity and the fact that it represents the most likely contender to our approach. As in the proposed Newton approach, inverse iteration approximates an eigenpair "in-place", e.g. without building a subspace. All algorithms were implemented in C/C++ linked with the Intel Math Kernel [1] and PETSc [2] libraries, compiled under the Intel MPI compiler using the -O3 optimization level. For Algorithms 3.1 and 4.1, the number of subdomains used will always equal the number of cores used and each distinct subdomain is handled by a single core (distributed-memory model only). The matrix $(B - \sigma I)$ was factored by the LDL factorization, obtained by the associated routine in the CHOLMOD [7] package. We did not consider any further high-performance computing optimizations.

For the inverse iteration method applied to A, the inner tolerance at each solve was set equal to the residual of the approximate eigenpair at the current step and thus it was gradually decreasing. For Newton's method, used either as a standalone method as in Algorithms 3.1 and 4.1, or pre-processed by inverse iteration with A, each update of σ was made after approximating $(\mu(\sigma), y(\sigma))$ by solving a linear system with $S(\sigma)$, using a fixed tolerance of 1e-2. For all (iterative) linear system solutions, we used the MINimum RESisudal (MINRES) [19] method as the iterative solver (we used the existing implementation in PETSc). By default we used no preconditioning for the inner solution in any of the methods tested. The residual norm tolerance for accepting an approximate eigenpair was set to tol = 1e-8, although the proposed Newton method almost all the times returned an approximation with residual norm less than 1e-10 or 1e-11. The residual norm was always evaluated directly by using the formula $\|r\| = \|A\hat{x}(\sigma) - \sigma\hat{x}(\sigma)\|$. All experiments were repeated multiple times and the average execution time is reported. As an additional note, we also performed the experiments to follow by using Lanczos with partial re-orthogonalization as the method to evaluate $(\mu(\sigma), y(\sigma))$. For practical reasons, we do not report these results but, as a sidenote, the Lanczos-based version proved competitive for extremal eigenvalue problems but became extremely expensive as we moved deeper in the spectrum.

6.3.2. Results. Table 6.2 shows the actual wall-clock timings when computing $k = 1$ and $k = 5$ consecutive eigenpairs next to $\zeta \in \mathbb{R}$ for a set of 2D model problems, while Table 6.3 presents similar results for a set of 3D model problems. The values of ζ were selected such that the eigenvalue problem was either extremal or slightly interior for both problems. For the 2D problems we chose $\zeta = 0$ and $\zeta = 0.01$ while for the 3D problems we set $\zeta = 0$ and $\zeta = 0.1$. The number in parentheses next to ζ (when $\zeta > 0$) denotes the number of negative eigenvalues of $(A - \zeta I)$. As previously, p denotes the number of subdomains, s denotes the size of the Schur complement matrix $S(.)$, and "It" denotes the total number of Newton steps performed by Algorithm 3.1 (when $k = 1$) or Algorithm 4.1 (when $k = 5$). We denote the total time spent in Algorithm 3.1 or Algorithm 4.1 by "T_{NT}" while the time spent in inverse iteration applied to A is denoted as "T_{II}". All timings are listed in seconds. Because $(\mu(\sigma), y(\sigma))$ is computed only approximately, extra care must be taken in order to avoid divergence when $\zeta \neq 0$. We always performed three steps of inverse iteration with A in order to "lock" the correct eigenpair(s) before we switch to Newton's scheme. In any case, the times reported are total running times and include all phases.

For 2D problems there is a signifigant advantage of the Newton-based method, for both $\zeta = 0$ and $\zeta = 0.01$ values, as can be seen in Table 6.2. By using 256 subdomains, we can compute the lowest eigenpair of a $n \approx 10^6$ 2D Laplacean in 0.2 seconds, while the five lowest eigenpairs can be computed in about one second. The severe difference in the runtimes when changing from $\zeta = 0.0$ to $\zeta = 0.01$ is due the fact that $(A - 0.01I)$

TABLE 6.2

Computing $k = 1$ and $k = 5$ eigenvalues next to ζ for a set of 2D problems. Times are listed in seconds.

n = 601 × 600

		$\zeta = 0.0$			$\zeta = 0.01$ (269)		
(p,k)	s	T_{NT}	It	T_{II}	T_{NT}	It	T_{II}
(16,1)	7951	1.21	3	7.60	70.2	4	221.2
(16,5)	- -	6.10	15	38.2	363.0	19	1154.0
(32,1)	12377	0.79	3	2.08	18.3	3	142.4
(32,5)	- -	5.41	14	11.3	85.2	14	723.5
(64,1)	18495	0.28	3	0.95	13.9	3	74.3
(64,5)	- -	1.94	14	5.23	47.3	14	354.0

n = 801 × 800

		$\zeta = 0.0$			$\zeta = 0.01$ (488)		
(p,k)	s	T_{NT}	It	T_{II}	T_{NT}	It	T_{II}
(64,1)	24945	0.73	3	2.61	37.4	2	197.1
(64,5)	- -	4.05	15	14.2	198.7	12	1052.2
(128,1)	36611	0.17	2	1.31	24.0	2	122.8
(128,5)	- -	1.15	9	6.61	125.0	11	601.0
(256,1)	52319	0.22	2	0.84	11.2	2	83.1
(256,5)	- -	1.29	9	4.42	61.3	10	439.0

n = 1001 × 1000

		$\zeta = 0.0$			$\zeta = 0.01$ (764)		
(p,k)	s	T_{NT}	It	T_{II}	T_{NT}	It	T_{II}
(128,1)	46073	0.42	3	2.40	95.3	2	114.1
(128,5)	- -	2.84	15	14.3	482.7	10	587.2
(256,1)	65780	0.20	2	1.59	54.2	2	84.4
(256,5)	- -	1.29	10	8.12	281.3	9	432.2
(512,1)	93440	0.21	2	1.83	49.4	2	106.0
(512,5)	- -	1.19	10	8.59	256.7	10	511.2

is now indefinite and has a large number of clustered eigenvalues very close to zero. Results for 3D problems are different from those of the 2D case and inverse iteration becomes more competitive, relative to the Newton-based method. The main reason is that now iterating with the spectral Schur complement is more expensive because the number of interface nodes, s, is larger and also the factorization of the $(B - \sigma I)$ matrix is more expensive. The Newton-based approach becomes faster when we use enough subdomains.

As a general comment regarding the results, for both the 2D and 3D problems, the overall cost typically scales linearly with the number of eigenpairs sought. This

TABLE 6.3

Computing $k = 1$ and $k = 5$ eigenvalues next to ζ for a set of 3D problems. Times are listed in seconds.

n = 41 × 40 × 39

		$\zeta = 0.0$			$\zeta = 0.1$ (19)		
(p,k)	s	T_{NT}	It	T_{II}	T_{NT}	It	T_{II}
(16,1)	15423	0.21	3	0.20	1.07	4	1.72
(16,5)	- -	1.39	15	1.12	5.85	19	8.80
(32,1)	20037	0.06	3	0.11	0.27	2	1.20
(32,5)	- -	0.34	14	0.71	1.52	14	6.11
(64,1)	24789	0.04	3	0.07	0.14	3	0.73
(64,5)	- -	0.32	14	0.51	1.01	15	3.84

n = 71 × 70 × 69

		$\zeta = 0.0$			$\zeta = 0.1$ (137)		
(p,k)	s	T_{NT}	It	T_{II}	T_{NT}	It	T_{II}
(64,1)	83358	0.60	3	0.91	15.4	2	15.1
(64,5)	- -	3.20	14	4.26	80.4	10	78.4
(128,1)	108508	0.19	3	0.42	3.12	2	9.31
(128,5)	- -	1.10	14	2.12	15.1	10	42.9
(256,1)	136159	0.10	3	0.47	5.99	2	13.9
(256,5)	- -	0.68	13	2.15	25.3	10	58.2

n = 101 × 100 × 99

		$\zeta = 0.0$			$\zeta = 0.1$ (439)		
(p,k)	s	T_{NT}	It	T_{II}	T_{NT}	It	T_{II}
(128,1)	230849	1.73	3	3.68	48.1	3	104.1
(128,5)	- -	7.24	15	15.2	233.2	16	504.8
(256,1)	293626	0.80	3	1.85	23.4	3	67.2
(256,5)	- -	3.80	14	9.11	117.0	16	340.0
(512,1)	369663	0.32	2	1.45	32.4	2	81.1
(512,5)	- -	1.61	12	7.71	168.9	12	385.2

is a natural consequence of the fact that our method is not a projection method and each eigenpair of A is computed on its own. Also, note that increasing the number of subdomains does not lead to a great reduction in the number of total Newton steps (as was the case in Table 6.1) because now $(\mu(\sigma), y(\sigma))$ is computed only approximately.

To conclude this set of experiments, we also present a detailed run-time analysis of Algorithm 4.1 when computing the $k = 5$ lowest eigenpairs. Figure 6.2 shows the portion of the time spent solving the linear systems with $S(\sigma)$ using MINRES, while Figure 6.3 shows the portion of time spent factorizing $(B - \sigma I)$. For the case of 2D problems, the cost inherited by the local factorizations is much smaller compared to

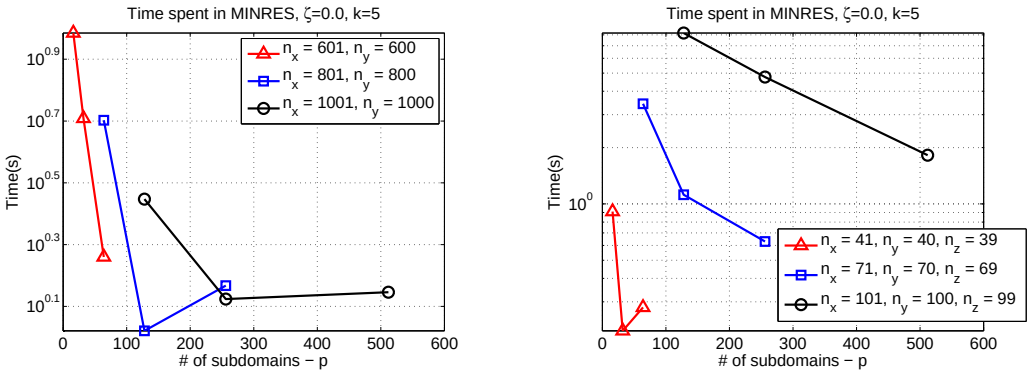

FIG. 6.2. *Time spent inside MINRES when computing the $k = 5$ lowest eigenpairs by Algorithm 4.1. Left subfigure: 2D case, Right subfigure: 3D case.*

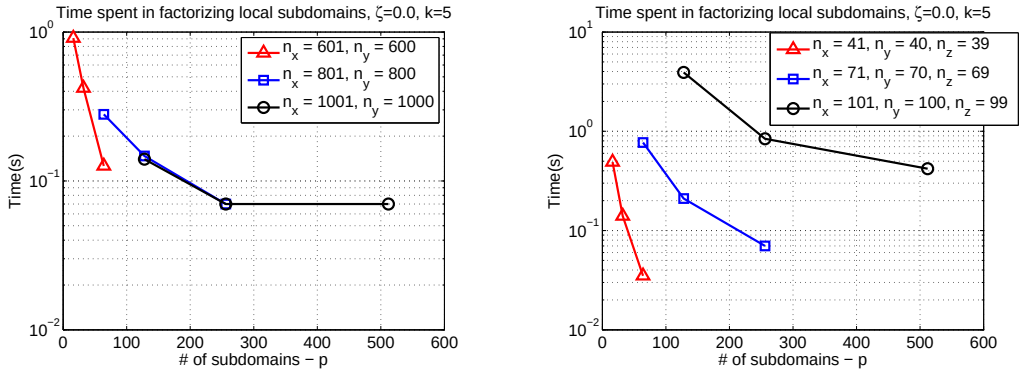

FIG. 6.3. *Time spent factorizing the local matrix $(B - \sigma I)$ for all the different values of σ produced by Algorithm 4.1 when computing the $k = 5$ lowest eigenpairs. Left subfigure: 2D case, Right subfigure: 3D case.*

that for the 3D problems. Figure 6.4 shows the average number of MV products with $S(\sigma)$ per computed eigenpair of A when solving for the $k = 5$ lowest eigenpairs. Note that for the case of 2D problems, the reduced average number of MV products for the two largest matrices is due the fact that Algorithm 4.1 converges in only two steps per eigenpair.

6.4. A comparison with ARPACK. This subsection provides a brief comparison between the Newton scheme and ARPACK [16], a broadly used software package based on an implicitly restarted Arnoldi/Lanczos process. By their nature the two methods are not comparable in a strict sense. However, it is useful to give an idea of the timings obtained when a small number of eigenvalues are to be computed. For the Newton scheme we used only one node of Itasca (8 cores). Thus, now each core actually handles multiple subdomains. Under this framework we can also test the performance of the Newton scheme in "serial" environments. For ARPACK, we set the size of the search subspace equal to twenty and we used only one execution core. The tolerance tol for the requested eigenpairs set again to $\text{tol} = 1e-8$ for both methods. As a demonstration, we used a single test 3D Laplacean with $n_x = 71, n_y = 70$,

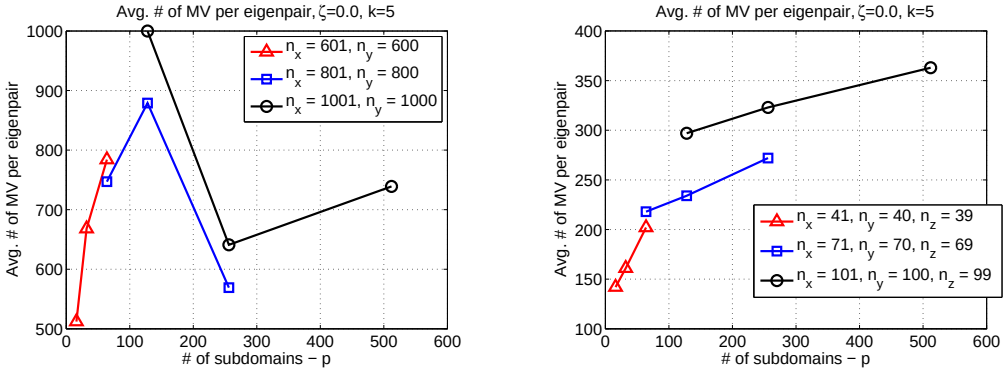

FIG. 6.4. *Average number of MV products per eigenpair when computing the $k = 5$ lowest eigenpairs by Algorithm 4.1. Left subfigure: 2D case, Right subfigure: 3D case.*

TABLE 6.4

Computing $k = 1$ and $k = 5$ eigenvalues next to ζ with the proposed Newton scheme and ARPACK. The discretization selected as $n_x = 71, n_y = 70$, and $n_z = 69$. Times are listed in seconds.

	$\zeta = 0.0$		$\zeta = 0.1$ (137)	
(p,k)	T_{NT}	T_{ARP}	T_{NT}	T_{ARP}
(64,1)	5.5	35.4	170.0	351.5
(128,1)	3.4	35.4	105.1	–
(256,1)	5.3	35.4	122.5	–
(64,5)	28.3	94.1	884.7	416.3
(128,5)	15.3	94.1	532.3	–
(256,5)	25.9	94.1	605.3	–

and $n_z = 69$ discretization points in each dimension. As previously, we selected $\zeta = 0$ and $\zeta = 0.1$.

Table 6.4 compares the execution times obtained of the Newton (T_{NT}) and ARPACK (T_{ARP}) schemes when searching for $k = 1$ and $k = 5$ eigenpairs of A, and using a different number of subdomains. Because ARPACK is a purely serial method and operates on A directly, it is oblivious to the number of subdomains used. On the other hand, the performance of the Newton scheme varies as the number of subdomains changes.

As expected, ARPACK becomes more efficient than the Newton-based method as we increase the number of eigenpairs sought for the specific problem, since it is a subspace method and can derive simultaneous approximations for multiple eigenpairs. On the other hand, the Newton method approximates each eigenpair in its own and the total cost is approximately linear to the number of eigenpairs sought.

6.5. Matrices from electronic structure calculations. In this subsection we report experiments with matrices from the PARSEC matrix set available from the University of Florida sparse matrix collection [8]. These matrices were originally obtained from the PARSEC code for electronic structure calculations using a Density Functional Theory (DFT) approach. The Hamiltonians are sparse and symmetric,

TABLE 6.5

Computing a single eigenpair closest to shifts ζ_1, ζ_2, and ζ_3, for a set of Hamiltonian matrices from the PARSEC matrix group. Time is listed in seconds.

Si10H16, $n = 17077$, $nnz = 875923$						
	$\zeta_1 = 0.29$		$\zeta_2 = 0.51$		$\zeta_3 = 1.11$	
p	T_{NT}	T_{II}	T_{NT}	T_{II}	T_{NT}	T_{II}
4	3.82	0.92	6.9	4.85	13.3	11.9
8	0.45	0.65	1.3	3.36	4.6	9.46

SiO, $n = 33401$, $nnz = 1317655$						
	$\zeta_1 = 0.75$		$\zeta_2 = 1.02$		$\zeta_3 = 2.17$	
p	T_{NT}	T_{II}	T_{NT}	T_{II}	T_{NT}	T_{II}
4	9.91	7.31	30.1	31.3	41.0	57.3
8	2.11	4.14	15.2	21.2	22.4	32.3

H2O, $n = 67024$, $nnz = 2216736$						
	$\zeta_1 = 1.35$		$\zeta_2 = 1.88$		$\zeta_3 = 3.84$	
p	T_{NT}	T_{II}	T_{NT}	T_{II}	T_{NT}	T_{II}
16	15.2	14.0	17.5	32.4	29.3	22.7
32	7.02	11.2	7.39	19.1	11.4	17.8

with multiple (and clustered) eigenvalues. Each Hamiltonian has a number of occupied states, say n_0, which is the number of smallest eigenvalues requested. While extensions are possible, as it is described the proposed scheme cannot compute multiple eigenvalues. In this set of experiments, we restrict our attention in computing a *single* eigenpair of each Hamiltonian matrix.

The results are shown in Table 6.5. We kept the same notation as in the previous subsection. Next to each matrix we also list its size n, as well as nnz the total number of non-zero entries. Unlike the model problem, the number of non-zero entries per row for the Hamiltonians is much larger ($57.3, 39.4$ and 33.07 non-zeros/row from smallest to largest matrix), and so the Schur complement tends to be larger, which actually leads to a harder problem for the Newton scheme. For each test matrix we used three different shifts $\zeta_1, \zeta_2, \zeta_3$, determined so that the shifted matrix had 40, 70, and 200 negative eigenvalues respectively. Similarly with the results obtained in the previous subsection, raising the number of subdomains typically leads to faster convergence for the Newton scheme. The Newton scheme is generally faster than inverse iteration, however both methods start to deteriorate as we move deeper into the spectrum. We should also note that, as was observed in our experiments, both methods can capture five or six digits of accuracy in a relatively short amount of time, but after this point convergence experiences a plateau until the requested tolerance tol $= 1e-8$ is finally met.

7. Conclusion. The method presented in this paper for solving symmetric eigenvalue problems, combines Newton's method with spectral Schur complements in a Domain Decomposition framework. The scheme essentially amounts to solving the eigenvalue problem along the interface points only, and exploits the fact that solves with the local subdomains are relatively inexpensive. A parallel implementation was presented and its performance evaluted for model Laplacean problems and general matrices from electronic structure calculations. The proposed method can be quite fast when only one or a very small number of extremal eigenpairs are sought. It can be combined with a few steps of inverse iteration to provide again a fast technique for solving what might be termed moderately interior eigenproblems, i.e., problems with eigenvalues not too deep inside the spectrum. One might compare the proposed approch to a Rayleigh quotient iteration, whereby the consecutive linear systems are handled by an iterative method in a domain decomposition framework. However, the focus on the Schur complement provides additional insights and leads to the Newton procedure presented here.

A key issue still requiring further investigation is to find effective ways to approximate the eigen-pairs $(\mu(\sigma), y(\sigma))$ of the Schur complement. We used inverse iteration implemented with the MINRES iterative method but did not consider any specific preconditioners. Preconditioning will become mandatory when solving interior eigenvalue problems where the desired eigenvalues are deep inside the spectrum, e.g., toward its middle. Such problems can be extremely difficult to solve if sparse direct solvers are ruled out, as is the case for very large 3-D problems. Eigenvectors of previous spectral Schur complements can again be used to accelerate the iterative solver as the shift changes. Because these ideas require a separate and rather involved study we opted to leave them for future work.

8. Acknowledgments. We are grateful to the University of Minnesota Supercomputing Institute for providing us with computational resources and assistance with the computations. We also thank Andreas Stathopoulos for fruitful discussions.

REFERENCES

[1] *Intel(r) fortran compiler xe 14.0 for linux*.
[2] S. Balay, J. Brown, K. Buschelman, V. Eijkhout, W. Gropp, D. Kaushik, M. Knepley, L. Curfman McInnes, B. Smith, and H. Zhang, *Petsc users manual revision 3.2*.
[3] C. Bekas and Y. Saad, *Computation of smallest eigenvalues using spectral schur complements*, SIAM J. Sci. Comput., 27 (2006), pp. 458–481.
[4] 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.
[5] E. G. Boman, U. V. Catalyurek, C. Chevalier, and K. D. Devine, *The Zoltan and Isorropia parallel toolkits for combinatorial scientific computing: Partitioning, ordering, and coloring*, Scientific Programming, 20 (2012), pp. 129–150.
[6] Ü. V. Çatalyurek and C. Aykanat, *Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication*, IEEE Transactions on Parallel and Distributed Systems, 10 (1999), pp. 673–693.
[7] Y. Chen, T. A. Davis, W. W. Hager, and S. Rajamanickan, *Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate*, ACM TOMS, 35 (2008), pp. 22:1–22:14.
[8] T. A. Davis, *University of florida sparse matrix collection, na digest*, 1994.
[9] H. Fang and Y. Saad, *A filtered lanczos procedure for extreme and interior eigenvalue problems*, SIAM J. Sci. Comput., 34 (2012), p. A2220A2246.
[10] G. H. Golub and C. F. Van Loan, *Matrix Computations, 4th edition*, Johns Hopkins University Press, Baltimore, MD, 4th ed., 2013.

[11] 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.
[12] T. Kato, *Perturbation Theory for Linear Operators*, Springer–Verlag, New–York, 1976.
[13] A. Knyazev and A. Skorokhodov, *Preconditioned gradient-type iterative methods in a subspace for partial generalized symmetric eigenvalue problems*, SIAM J. Numer. Anal., 31 (1994), pp. 1226–1239.
[14] T. G. Kolda, *Partitioning sparse rectangular matrices for parallel processing*, Lecture Notes in Computer Science, 1457 (1998), pp. 68–79.
[15] C. Lanczos, *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*, J. Res. Natl Bur. Std., 45 (1950), pp. 225–282.
[16] R. B. Lehoucq, D. C. Sorensen, and C. Yang, *Arpack users guide: Solution of large-scale eigenvalue problems by implicitely restarted arnoldi methods*, SIAM, Philadelphia, PA, 1998.
[17] S. H. Lui, *Kron's method for symmetric eigenvalue problems*, J. of Comput. and Appl. Math., 98 (1998), pp. 35–48.
[18] ———, *Domain decomposition methods for eigenvalue problems*, J. of Comput. and Appl. Math., 117 (2000), pp. 17–34.
[19] C. C. Paige and M. A. Saunders, *Solution of sparse indefinite systems of linear equations*, SIAM J. Numer. Anal., 12 (1975), pp. 617–629.
[20] F. Pellegrini, Scotch *and* libScotch *5.1 User's Guide*, INRIA Bordeaux Sud-Ouest, IPB & LaBRI, UMR CNRS 5800, 2010.
[21] B. Philippe and Y. 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.
[22] E. Polizzi, *Density-matrix-based algorithms for solving eigenvalue problems*, Phys. Rev. B., 79 (2009).
[23] A. Pothen, H. D. Simon, and K. P. Liou, *Partitioning sparse matrices with eigenvectors of graphs*, SIAM Journal on Matrix Analysis and Applications, 11 (1990), pp. 430–452.
[24] T. Sakurai and H. Sugiura, *A projection method for generalized eigenvalue problems using numerical integration*, J. of Comp. and App. Math., 159 (2003), pp. 119–128.
[25] A. Sameh and J. Wisniewski, *A trace minimization algorithm for the generalized eigenvalue problem*, SIAM J. Numer. Anal., 19 (1982), pp. 1243–1259.
[26] H. Simon, *The lanczos algorithm with partial reorthogonalization*, Mathematics of Computation, 42 (1984), pp. 115–142.
[27] G. Sleijpen and H. Van der Vorst, *A jacobi-davidson iteration method for linear eigenvalue problems*, SIAM Review, 42 (2000), pp. 267–293.
[28] A. Stathopoulos, Y. Saad, and C. Fischer, *A schur complement method for eigenvalue problems*, 7th Copper Mountain Conference on Multigrid Methods, NASA Conference Proceedings, NASA, (1995).
[29] A. Stathopoulos, Y. Saad, and K. Wu, *Dynamic thick restarting of the Davidson and the implicitly restarted Arnoldi methods*, SIAM J. Sci. Comput., 19 (1998), pp. 227–245.
[30] K. Wu and H. Simon, *Thick-restart Lanczos method for large symmetric eigenvalue problems*, SIAM J. Matrix Anal. Appl., 22 (2000), pp. 602–616.