Parallel Two-level Domain Decomposition Based Jacobi-Davidson Algorithms for Pyramidal Quantum Dot Simulation

Tao Zhao[a], Feng-Nan Hwang[b], Xiao-Chuan Cai[c,*]

[a] *Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen 518055, China*
[b] *Department of Mathematics, National Central University, Jhongli 320, Taiwan*
[c] *Department of Computer Science, University of Colorado Boulder, CO 80309, USA*

Abstract

We consider a quintic polynomial eigenvalue problem arising from the finite volume discretization of a quantum dot simulation problem. The problem is solved by the Jacobi-Davidson (JD) algorithm. Our focus is on how to achieve the quadratic convergence of JD in a way that is not only efficient but also scalable when the number of processor cores is large. For this purpose, we develop a projected two-level Schwarz preconditioned JD algorithm that exploits multilevel domain decomposition techniques. The pyramidal quantum dot calculation is carefully studied to illustrate the efficiency of the proposed method. Numerical experiments confirm that the proposed method has a good scalability for problems with hundreds of millions of unknowns on a parallel computer with more than 10,000 processor cores.

Keywords: Polynomial eigenvalue problem, two-level Schwarz preconditioner, Jacobi-Davidson algorithm, parallel performance, quantum dot simulation

1. Introduction

Polynomial eigenvalue problems are of great interests because there are many important applications in science and engineering, such as the stability analysis in fluid mechanics, the vibration problem in solid mechanics, the quantum dot problem in nanotechnology; see [1–7] and references therein. The JD algorithm, originally proposed by Sleijpen and Van der Vorst for solving algebraic linear eigenvalue problems [8, 9], has been shown to be effective for polynomial eigenvalue problems [1, 2, 10, 11], especially for the case when several interior eigenvalues are of interests. The JD algorithm belongs to a class of subspace iterative methods, which consists of two key steps: first increase the search space

*Corresponding author
Email address: cai@cs.colorado.edu (Xiao-Chuan Cai)

by adding a new basis vector and then extract the approximate eigenpair from the search space through a Rayleigh-Ritz procedure. To obtain a new basis vector for the search space, at each JD iteration, one needs to solve inexactly a large sparse linear system of equations, which is referred to as the correction equation, by a preconditioned Krylov subspace type method, such as GMRES or CG methods [12].

The numerical experiences suggest that the robustness and the efficiency of the JD algorithm depend on the following three factors: (1) the initial search space, (2) the Ritz pair selection strategy, and (3) the solution quality of the correction equation. Similar to Newton-type methods for solving nonlinear systems, the JD algorithm is a locally convergent iterative method, i.e., if the initial guess is not close enough to the exact solution, the convergence often exhibits some stagnation behavior, or even worse, is not achieved. One important feature of the JD algorithm is that at each JD iteration, only mild solution accuracy of the correction equation is required. Several recent publications were related to the correction equation solvers. Feng [6] applied a multilevel JD method for the generalized eigenvalue problem with application in the finite element analysis of structural dynamic problems. A multigrid-type preconditioner was used in conjugation with FGMRES as the correction equation solver. The incomplete Cholesky factorization without fill-ins was employed as the pre- and post-smoothers and the coarse grid problem was solved by a direct method. Arbenz et al. [5] proposed a hybrid preconditioner combining a hierarchical basis preconditioner and an algebraic multigrid preconditioner for the correction equation in the JD algorithm for solving symmetric generalized Maxwell eigenvalue problem; they reported good parallel scalability using up to 16 processors.

The aim of the paper is to develop and study a two-level JD algorithm for the large sparse polynomial PDE eigenvalue problems and its applications in quantum dot simulations. The concept of two-level approach fits in the JD algorithm in three aspects. The first idea is to use a coarse mesh to construct the initial vector in the search space since for many applications, the smooth eigenvector corresponding to the low frequency can be well represented by the coarse mesh solution. The second idea is to use the coarse solution as a guideline for selecting a proper Ritz-pair. For example [2], a similarity measure was applied using the coarse eigenvalues and eigenvectors as a reference to avoid picking un-physical spurious root introduced during the Raleigh-Ritz projection procedure. The third idea is in the construction of the preconditioner using the domain decomposition method for the Krylov subspace iterative-type correction equation solver. Our proposed two-level preconditioner is based on the Schwarz framework [13–15], which has a long successful history for linear elliptic PDEs. We also compare numerically the proposed approach with a one-level method, and a popular two-level orthogonal Arnoldi (TOAR) method using PETSc [16] and SLEPc [17]. Our approach outperforms both of them in terms of the total compute time, and the strong scalability on a machine with a large number of processor cores.

The rest of the paper is organized as follows. Section 2 briefly introduces the pyramidal quantum dot problem. In Section 3, we propose a projected two-level

domain decomposition based Jacobi-Davidson algorithm. Numerical results of the proposed algorithm and comparison with other methods are reported in Section 4. Some final remarks are given in Section 5.

2. Pyramidal quantum dot problem

We consider polynomial eigenvalue problems arising from quantum dot (QD) simulations. An example of QD is a pyramid dot embedded in a cuboid as shown in Figure 1. Due to the confinement effect, the pyramidal quantum dot has discrete energy states. This type of quantum dot can be produced by a few manufacturing procedures and has many applications, such as lasers and single-electron devices [18].

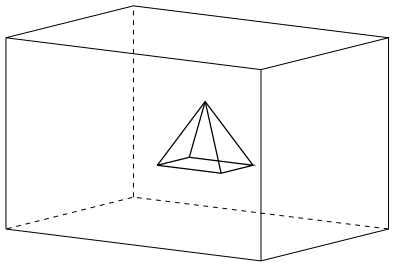

Figure 1: Structure of a pyramidal quantum dot embedded in a cuboid.

The central task is to compute some energy states and their corresponding wave functions, by solving an eigenvalue problem [1, 3, 4, 18–22]. The quantum states of a pyramidal quantum dot with a single electron can be described by the time-independent 3D Schrödinger equation

$$-\nabla \cdot \left(\frac{\hbar^2}{2m(\mathbf{r},\lambda)} \nabla u \right) + V(\mathbf{r})u = \lambda u, \tag{1}$$

on the domain Ω, where λ is called an energy state or eigenvalue, and u is the corresponding wave function or eigenvector. In (1), $\hbar$ is the reduced Plank constant, $\mathbf{r}$ is the space variable, $m(\mathbf{r},\lambda)$ is the effective electron mass, and $V(\mathbf{r})$ is the confinement potential. Taking the effect of the spin-orbit splitting into account, the effective mass model

$$m(\mathbf{r},\lambda) = \begin{cases} m_1(\lambda) & \text{in the pyramid} \\ m_2(\lambda) & \text{in the cuboid} \end{cases}, \quad V(\mathbf{r}) = \begin{cases} V_1 & \text{in the pyramid} \\ V_2 & \text{in the cuboid} \end{cases}$$

can be derived from the eight-band $\mathbf{k} \cdot \mathbf{p}$ analysis and the effective mass theory [3, 22]. More precisely,

$$\frac{1}{m_i(\lambda)} = \frac{P_i^2}{\hbar^2} \left(\frac{2}{\lambda + \ell_i - V_i} + \frac{1}{\lambda + \ell_i - V_i + \delta_i} \right), \quad i = 1, 2, \tag{2}$$

where P_i, ℓ_i and δ_i are the momentum, main energy gap and spin-orbit splitting corresponding to the pyramid and the cuboid, respectively.

Since the pyramidal QD is a heterostructure, the Ben Daniel-Duke condition [1, 3] is imposed on the interface of the two materials:

$$\left(\frac{1}{m(\mathbf{r},\lambda)}\frac{\partial u}{\partial \mathbf{n}}\right)\Big|_{\partial D_-} = \left(\frac{1}{m(\mathbf{r},\lambda)}\frac{\partial u}{\partial \mathbf{n}}\right)\Big|_{\partial D_+} \tag{3}$$

where D denotes the domain of the pyramid dot and $\mathbf{n}$ is the unit outward normal for each surface of ∂D. Since the corresponding wave functions decay exponentially outside the pyramid dot, the homogeneous Dirichlet boundary condition

$$u = 0 \tag{4}$$

is imposed on the boundary of the cuboid $\partial\Omega$.

A cell-centered second-order finite volume method [3] on an uniform mesh in Cartesian coordinates is applied to discretize the Schrödinger equation with non-parabolic effective mass model. With this finite volume method, the interface condition (3) is applied implicitly. The resulting system is a polynomial eigenvalue problem

$$(\lambda^5 A_5 + \lambda^4 A_4 + \lambda^3 A_3 + \lambda^2 A_2 + \lambda A_1 + A_0)x = 0, \tag{5}$$

where $\lambda \in \mathbb{C}$, $x \in \mathbb{C}^N$, $A_i \in \mathbb{R}^{N\times N}$, and N is the total number of unknowns. The matrices A_5 and A_4 are diagonal, and all other matrices are nonsymmetric.

3. Jacobi-Davidson algorithm with a projected two-level Schwarz preconditioner for the correction equation

We begin with some notations. For given $A_i \in \mathbb{C}^{N\times N}$, $i = 0, 1, \cdots, m$, such that their null spaces only have a trivial intersection, we define

$$\mathcal{A}_\phi = \sum_{i=0}^{m} \phi^i A_i$$

as a matrix polynomial of $\phi \in \mathbb{C}$. If there exist $\lambda \in \mathbb{C}$ and nonzero $x \in \mathbb{C}^N$ such that

$$\mathcal{A}_\lambda x = 0, \tag{6}$$

then λ is called an eigenvalue of $\mathcal{A}_\phi$ and x is the right eigenvector of $\mathcal{A}_\phi$ associated with the eigenvalue λ. In general, (6) is referred to as the polynomial eigenvalue problem of degree m.

The Jacobi-Davidson algorithm is a powerful approach for solving the polynomial eigenvalue problem (6). The details of the JD algorithm is summarized in Algorithm 1. Since the correction equation is the most expensive part of computation, the parallel performance of the JD algorithm is determined mostly by how the correction equation is solved. We introduce a projected two-level Schwarz preconditioner that improves greatly the convergence of the correction equation solver and is scalable.

Algorithm 1 JD for polynomial eigenvalue problems

Input: A_i $(i = 0, \cdots, m)$, the maximum number of iterations k, the nonzero stopping criteria ξ.

1: Choose an initial guess eigenpair (θ, v) with $\|v\|_2 = 1$. Let $V = [v]$.
For $n = 0, \cdots, k$
2: Compute $W_i = A_i V$ and $M_i = V^H W_i$ for $i = 0, \cdots, m$.
3: Solve the projected polynomial eigenvalue problem $\left(\sum_{i=0}^{m} \phi^i M_i\right) s = 0$, then obtain the desired eigenpair (ϕ, s) such that $\|s\|_2 = 1$.
4: Compute the Ritz vector $u = Vs$, and the residual vector $r = \mathcal{A}_\phi u$.
5: If the stopping criteria is satisfied, i.e., $\|r\|_2 \le \xi$, then stop.
6: Compute $p = \mathcal{A}'_\phi u = \left(\sum_{i=1}^{m} i\phi^{i-1} A_i\right) u$.
7: Solve approximately the correction equation

$$\left(I - \frac{pu^H}{u^H p}\right) \mathcal{A}_\phi (I - uu^H) z = -r, \quad z \perp u.$$

8: Orthogonalize z against V, set $v = z/\|z\|_2$, then expand $V \leftarrow [V, v]$.
End for

In Algorithm 1, Step 3 implies a Galerkin condition that r is orthogonal to the subspace $span\{V\}$. At each JD iteration, the projected polynomial eigenproblem is solved by the QZ method with linearization, see [1, 17] for details. Then ϕ is chosen to be the desired eigenvalue that is closest to the initial guess eigenvalue θ, that is, $|\phi - \theta|$ is minimum among all the eigenvalues of the projected polynomial eigenproblem. If the accuracy of (ϕ, u) is satisfied, we use (ϕ, u) as an approximate eigenpair, otherwise, the correction equation needs to be formed and solved. Once the correction z is obtained, the subspace $span\{V\}$ is expanded by adding z after orthonormalization. This procedure goes back and forth until either the approximate eigenpair with sufficient accuracy is found, or the number of JD iterations exceeds the maximum number.

Assume that $\tilde{M}^{-1}$ is a right preconditioner applied to the correction equation in Step 7 of Algorithm 1, that is, we solve

$$\left(I - \frac{pu^H}{u^H p}\right) \mathcal{A}_\phi (I - uu^H) \tilde{M}^{-1} t = -r \tag{7}$$

by a Krylov subspace method where $t = \tilde{M} z$, $z \perp u$. As suggested in [8, 9], we consider $\tilde{M}$ of the form

$$\tilde{M} = \left(I - \frac{pu^H}{u^H p}\right) \mathcal{M} (I - uu^H) \tag{8}$$

where $\mathcal{M}$ is a nonsingular matrix as an approximation of $\mathcal{A}_\phi$. Since $\tilde{M}$ in (8) is singular, $\tilde{M}^{-1}$ should be a pseudo-inverse of $\tilde{M}$. It has been shown that $\tilde{M}^{-1}\tilde{M}$ is the orthogonal projection onto the range of $\tilde{M}^H$ [23] that is the orthogonal complement of span$\{u\}$ because of the right projection. It follows from $z \perp u$

that $\tilde{M}^{-1}\tilde{M}z = z$. Thus the equation (7) is equivalent to the correction equation in Step 7. As we shall see, we do not need to form $\tilde{M}^{-1}$ explicitly.

In the Krylov subspace solver, for a given vector y,

$$\left(I - \frac{pu^H}{u^H p}\right)\mathcal{A}_\phi(I - uu^H)\tilde{M}^{-1}y \tag{9}$$

is orthogonal to u due to the left projection. Note that r is orthogonal to u as well because of the Galerkin condition in Step 3. Therefore, the initial residual vector of the Krylov subspace solver is orthogonal to u. As a result, the Krylov subspace is in the orthogonal complement of span$\{u\}$. Because of this, we choose an initial solution of the equation (7) such that it is orthogonal to u. To compute the matrix-vector product in (9) at each iteration of the Krylov subspace solver, $x = (I - uu^H)\tilde{M}^{-1}y$ is carried out first. Since $\tilde{M}\tilde{M}^{-1}$ is the orthogonal projection onto the range of $\tilde{M}$ [23] that is also the orthogonal complement of span$\{u\}$, we have

$$\tilde{M}x = \tilde{M}\tilde{M}^{-1}y - \tilde{M}u\left(u^H\tilde{M}^{-1}y\right) = y.$$

Since x is orthogonal to u, it can be computed in the following way

$$x = \mathcal{M}^{-1}y - \frac{u^H\mathcal{M}^{-1}y}{u^H\mathcal{M}^{-1}p}\mathcal{M}^{-1}p, \tag{10}$$

for details, see [24, 25] and references therein. Thus, for solving each correction equation, we need to compute $s = \mathcal{M}^{-1}y$ at each iteration of the Krylov subspace solver. Obviously, $\tilde{M}^{-1}$ is a good preconditioner when $\mathcal{M}^{-1}$ approximates $\mathcal{A}_\phi^{-1}$ well. Therefore, we apply the two-level technique to $\mathcal{M}^{-1}$ instead of $\tilde{M}^{-1}$. As a result, we avoid the projections on the coarse level. Once t in (7) is computed, z can be obtained as x in (10) because of $z \perp u$.

Before introducing the two-level Schwarz preconditioner, we first introduce the one-level preconditioner that is applied on the fine level. We partition the fine mesh Ω_f into non-overlapping subdomains ω_i, $i = 1, \cdots, n_p$, then generate the overlapping subdomain ω_i^δ by including δ layers of mesh cells in the neighboring subdomains of ω_i, i.e., $\omega_i \subset \omega_i^\delta$ in Ω_f. Here, n_p is the number of subdomains that is equal to the number of processor cores. This means each processor core is assigned one subdomain. Let R_i^0 be the restriction operator to the non-overlapping subdomain ω_i and R_i^δ be the restriction operator to the overlapping subdomain ω_i^δ, respectively. The matrix on each overlapping subdomain ω_i^δ is defined as

$$\mathcal{J}_i = R_i^\delta\mathcal{A}_\phi(R_i^\delta)^T.$$

Then the one-level RAS preconditioner [26–29] on the fine level reads as

$$M_f^{-1} = \sum_{i=1}^{n_p}(R_i^0)^T\mathcal{J}_i^{-1}R_i^\delta. \tag{11}$$

We mention that for the pyramidal quantum dot simulation, the subdomain matrix $\mathcal{J}_i$ can also be obtained using the following method. The Schrödinger equation (1) is discretized on ω_i^δ with a homogenous Dirichlet boundary condition on $\partial\omega_i^\delta$. If ω_i^δ has a part of the pyramidal dot, then the interface condition (3) should be imposed. We then obtain a polynomial eigenvalue problem

$$\left(\sum_{i=0}^{m} \hat{\lambda}^i \hat{A}_i\right) \hat{x} = 0$$

on each ω_i^δ. With the Ritz value ϕ computed on Ω_f at the current JD iteration, $\mathcal{J}_i$ is then computed as

$$\mathcal{J}_i = \sum_{i=0}^{m} \phi^i \hat{A}_i.$$

In practice, $\mathcal{J}_i^{-1}$ is not formed explicitly, instead its multiplication with a vector is obtained approximately by solving a subdomain linear system with ILU factorization.

Next, we consider preconditioning on the coarse level. Let I_c^f be an interpolation from Ω_c to Ω_f, R_f^c be a restriction from Ω_f to Ω_c and M_c^{-1} be the preconditioner on Ω_c. To obtain M_c, we discretize the Schrödinger equation (1) with the boundary condition (4) and the interface condition (3) on Ω_c by the finite volume method described in Section 2 and then obtain a coarse mesh polynomial eigenvalue problem

$$\left(\sum_{i=0}^{m} \tilde{\lambda}^i \tilde{A}_i\right) \tilde{x} = 0.$$

The matrix $\tilde{A}_i$ $(i = 0, \cdots, m)$ is much smaller than A_i in (5), but they have similar sparsity structure. Using the Ritz value ϕ computed on Ω_f at the current JD iteration, we define M_c as

$$M_c = \sum_{i=0}^{m} \phi^i \tilde{A}_i. \tag{12}$$

Since the exact computation of $M_c^{-1}(R_f^c y)$ is not required by the two-level Schwarz preconditioner, there is no need to form M_c^{-1}; see [14, 15, 30]. To compute $w = M_c^{-1}(R_f^c y)$ with a low accuracy, we solve a linear system $M_c w = R_f^c y$ approximately using a Krylov subspace method with the one-level RAS preconditioner defined on the coarse mesh Ω_c using the same number of processor cores as on the fine mesh. The way to build the one-level preconditioner on Ω_c is the same as that on Ω_f.

Now we are ready to define the two-level preconditioner. Let $\mathcal{M}^{-1}$ be a two-level multiplicative type Schwarz preconditioner [14, 15, 28] that requires a fine mesh Ω_f and a coarse mesh Ω_c. Its multiplication with a vector y can be carried out in two steps:

$$\begin{aligned} s &\leftarrow I_c^f M_c^{-1} R_f^c y, \\ s &\leftarrow s + M_f^{-1}(y - \mathcal{A}_\phi s). \end{aligned}$$

It can also be written as a one-step formula

$$s \leftarrow (M_f^{-1} - M_f^{-1}\mathcal{A}_\phi I_c^f M_c^{-1} R_f^c + I_c^f M_c^{-1} R_f^c)y,$$

that is,

$$\mathcal{M}^{-1} = M_f^{-1} - M_f^{-1}\mathcal{A}_\phi I_c^f M_c^{-1} R_f^c + I_c^f M_c^{-1} R_f^c.$$

If ϕ in (12) is very close to the exact eigenvalue on Ω_f, then $\mathcal{A}_\phi$ is ill-conditioned, moreover, ILU or LU factorization of $\mathcal{A}_\phi$ will encounter the stability issue since some diagonal elements are close to zero. Fortunately, in this case, the Ritz vector associated with ϕ is a good approximate eigenvector. In fact, this Ritz vector is used to prevent the operator in the correction equation from being singular; see [25, 31]. If ϕ is also a good approximation on Ω_c, then M_c in (12) is ill-conditioned. In this case, the one-level RAS preconditioner on Ω_c may not be efficient for solving the coarse linear system. An interesting observation is that a good approximate eigenvector on Ω_c corresponding to ϕ can be easily obtained by restricting the Ritz vector corresponding to ϕ from Ω_f to Ω_c. Then we can use the obtained vector on Ω_c to build a coarse grid correction preconditioner in an algebraic way so as to remove or shift the small eigenvalue of M_c; for details, see [30, 32] and references therein. Therefore, the stability problem can be avoided with respect to M_c^{-1}. Numerical results in Section 4 show that the one-level RAS preconditioner is sufficient for the linear system on Ω_c.

In terms of the number iterations, the two-level Schwarz preconditioner becomes more efficient if the overlapping size δ increases, or the coarse mesh Ω_c is refined. On the other hand, a large overlap will require more communication time, and a finer coarse mesh will also increase the compute time. Note that we have two goals in mind that are (1) near quadratic convergence in terms of the JD iterations; and (2) near linear speedup in terms of the total compute time. To realize both goals simultaneously, we need to carefully select the sizes of the coarse meshes, as well as the stopping conditions and the overlapping size.

4. Numerical results

In this section, we use Algorithm 1 with the projected two-level Schwarz preconditioner to compute 6 smallest positive eigenvalues and the corresponding eigenvectors of the pyramidal quantum dot problem as shown in Figure 1. The software is implemented using PETSc [16] consisting of a variety of libraries for the scalable solution of partial differential equations. The projected polynomial eigenproblem is solved by a QZ method implemented in SLEPc [17] which is a scalable library for the solution of various types of eigenvalue problems. All numerical tests are run on an IBM iDataPlex cluster, which consists of 72,576 Intel Sandy Bridge processors interconnected by an Infiniband network with 144.6 TB of memory.

4.1. Test case setup

The size of the cuboid is 24.8nm×24.8nm×18.6nm. The width of the pyramid base is 12.4nm and the height of the pyramid is 6.2nm. The physical parameters of the non-parabolic effective mass model (2) are described in Table 1. The mesh in the discretization has L, M, N mesh cells in each of x, y, z

Table 1: Physical parameters in the effective mass model.

	P_i	ℓ_i	δ_i	V_i
$i = 1$ (pyramid)	0.8503	0.42	0.48	0.00
$i = 2$ (cuboid)	0.8878	1.52	0.34	0.77

directions, thus $\Delta x = 24.8/L$, $\Delta y = 24.8/M$, and $\Delta z = 18.6/N$. For simplicity, we set $\Delta x = \Delta y = \Delta z$ that leads to $4L = 4M = 3N$. Due to the Dirichlet boundary condition (4), the total number of unknowns of the polynomial eigenvalue problem (5) is $(L-1) \times (M-1) \times (N-1)$. We consider two fine meshes. The first fine mesh is 320×320×240 with 24,320,879 unknowns, and the second is 600×600×450 with 161,101,649 unknowns. In the rest of the paper, we refer to the meshes as **Mesh1** and **Mesh2**, respectively.

For solving the Schrödinger equation on the fine mesh Ω_f, we stop the JD iteration when the absolute residual norm is below 10^{-10}. The correction equation is solved by the flexible GMRES (FGMRES) without restarting [12] preconditioned by either one-level or two-level preconditioners. The maximum number of iterations of FGMRES on Ω_f is 400. The relative tolerance ε_n of FGMRES on Ω_f is 10^{-4}. In the two-level Schwarz preconditioner, the linear system on the coarse mesh Ω_c is solved by FGMRES with a coarse mesh RAS preconditioner. FGMRES on Ω_c is stopped when the relative residual norm is below 10^{-1}. At each iteration of FGMRES on Ω_f, the solution obtained on Ω_c is interpolated to Ω_f by a trilinear interpolation. Since Ω_c is generated independent of Ω_f, Ω_c is not necessarily a subset of Ω_f. The restriction from Ω_f to Ω_c is defined as follows. For each mesh point in Ω_c, we find an element of Ω_f that contains the point, then the value at the coarse mesh point is obtained by a trilinear interpolation using the values at the vertices of the fine mesh element. For the RAS preconditioners on both Ω_c and Ω_f, ILU(0) is applied to solve the linear system on each subdomain, with overlap 1. In Section 4.4, we also use an adaptive relative tolerance ε_n for FGMRES on Ω_f and different overlap size δ to investigate the JD convergence.

4.2. Initial guess calculation

Let Ω_0 be a coarse mesh covering Ω, and we discretize (1)-(4) on the coarse mesh using the same discretization as described in Section 2. Since the coarse mesh Ω_0 for the initial guess calculation is quite small, we solve the corresponding polynomial eigenvalue problems redundantly on all processor cores for the 6 smallest positive eigenvalues and eigenvectors using JD with the locking technique [1, 3] in which the one-vector is used as the initial guess. The JD iteration is stopped when the absolute residual norm is below 10^{-8}. Once the 6 eigenpairs

on Ω_0 are obtained, the eigenvalues are used as the targets for the fine mesh problem, and the eigenvectors are interpolated to the fine mesh by a trilinear interpolation.

For both fine meshes, we use the same coarse mesh $12\times12\times9$ with 968 unknowns. The coarse mesh for the two-level preconditioner has to be sufficiently fine for the coarse preconditioner to be effective, but our numerical experiments show that the coarse mesh for generating the initial guess doesn't need to be as fine for the pyramidal quantum dot problem.

4.3. Parallel performance studies

We emphasize that the calculations for different energy states are carried out separately after the initial guess is computed. Therefore we can choose different coarse meshes for each eigenpair to balance the computation on the fine and coarse meshes so that better parallel performance can be achieved. Table 2 reports the computed eigenvalues and the coarse meshes used in the two-level Schwarz preconditioner for each eigenvalue for **Mesh1**. Since the imaginary parts of the computed eigenvalues are less than 10^{-13}, only the real parts are reported. Table 3 contains results obtained by the one-level preconditioner and the two-level preconditioner for **Mesh1**.

Table 2: On **Mesh1**, the computed eigenvalue e_i and the coarse mesh Ω_c in the two-level preconditioner for each eigenvalue.

	e_i	Ω_c
0	0.4162409037031	$44 \times 44 \times 33$
1	0.5990985865304	$56 \times 56 \times 42$
2	0.5990985865304	$48 \times 48 \times 36$
3	0.7179653490658	$112 \times 112 \times 84$
4	0.7295464275825	$52 \times 52 \times 39$
5	0.7925271797673	$104 \times 104 \times 78$

As shown in Table 3, JD with the one-level and two-level preconditioners has almost the same number of iterations when the relative residual norm of the solution of the correction equation drops below ε_n, but in terms of the compute time and the average number of FGMRES iterations, the two-level preconditioner offers a great improvement.

We also use the two-level orthogonal Arnoldi (TOAR) method to solve the polynomial eigenvalue problem. TOAR applies an implicit linearization process and is the default eigensolver in SLEPc for polynomial eigenvalue problems with arbitrary degree [17, Chapter 5]. We restart the Arnoldi process every 100 iterations. The tolerance of TOAR is 10^{-10}. The linear systems in the spectral transformation is solved by the full GMRES with the one-level preconditioner. The maximum number of iterations of GMRES is 400. It is suggested in [17, p. 46] that the appropriate tolerance of the linear solver in the spectral transformation is usually slightly more stringent than the tolerance of the eigenvalue calculation. So the relative tolerance of GMRES is chosen to be 10^{-11}.

Table 3: On **Mesh1**, comparison of the results obtained by the one-level and two-level preconditioners. e_i is the i^{th} computed eigenvalue, n_p is the number of processor cores, JD is the number of JD iterations, FGMRES is the average number of FGMRES iterations for the solution of the correction equations, and Time is the total compute time in seconds.

		One-level			Two-level		
	n_p	JD	FGMRES	Time	JD	FGMRES	Time
e_0	128	4	102.75	109.34	4	21.50	23.06
	256	4	102.75	51.44	4	22.00	10.39
	512	4	102.75	24.94	4	21.75	5.75
	1024	4	103.75	11.96	4	22.25	2.86
	2048	4	104.00	7.29	4	22.50	1.59
e_1	128	4	138.75	162.98	4	20.00	20.90
	256	4	138.75	80.58	4	20.75	9.82
	512	4	137.75	38.93	4	20.75	6.22
	1024	4	134.75	17.46	4	21.25	2.88
	2048	4	135.75	10.26	4	21.75	1.77
e_2	128	4	133.00	158.06	4	30.50	32.26
	256	4	133.50	75.98	4	29.75	13.54
	512	4	131.25	36.61	4	27.75	6.73
	1024	4	140.75	18.44	4	32.25	5.03
	2048	4	141.50	10.86	4	30.25	2.75
e_3	128	7	355.29	976.40	7	9.43	39.20
	256	7	356.29	461.61	7	9.57	18.78
	512	7	337.29	249.40	7	9.43	9.89
	1024	7	353.29	151.58	7	9.57	5.27
	2048	7	354.29	80.45	7	9.29	3.35
e_4	128	4	230.00	276.27	4	41.00	49.29
	256	4	230.50	149.24	4	42.50	21.02
	512	4	202.00	72.78	4	41.75	11.25
	1024	4	223.00	38.97	5	42.40	6.35
	2048	5	260.80	33.02	4	42.50	3.92
e_5	128	5	304.20	593.02	5	12.80	36.31
	256	5	312.80	319.10	5	17.00	21.20
	512	5	272.00	159.19	5	13.00	7.74
	1024	5	308.20	81.95	5	8.80	3.24
	2048	5	309.60	43.38	5	9.00	2.69

The 6 smallest positive eigenvalues are obtained without restart. The computation is carried out with real arithmetic. Table 4 reports the total compute time of TOAR in seconds. Compared with Tables 3, we can see that TOAR is much slower than JD with the one-level and two-level preconditioners. Because the Arnoldi method converges slowly and the linear systems of the spectral transformation need to be solved very accurately, TOAR spends more compute time than JD. It should be noted that for the nonsymmetric eigenproblem, Algorithm 1 requires complex arithmetic since the eigenvector may be complex, while Arnoldi-type method works with real arithmetic.

Table 4: On **Mesh1**, the total compute time of the TOAR method in seconds.

n_p	128	256	512	1024	2048
Time	4223.48	2239.12	1177.46	742.80	504.83

Figures 2 and 3 plot the speedup curves of JD with the one-level and two-level preconditioners and TOAR for **Mesh1**. Since JD takes one more iteration (see Table 3) when e_4 is computed on 2048 processor cores, the speedup of JD with one-level preconditioner encounters a big drop. As we see, for other eigenpairs, JD with both one-level and two-level preconditioners are more scalable than TOAR.

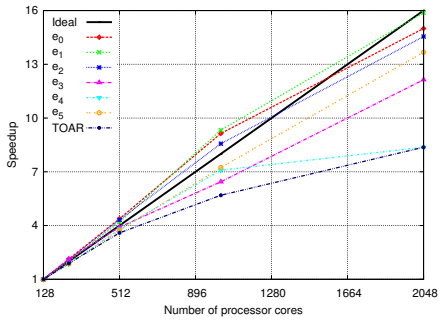

Figure 2: Speedup for **Mesh1** with one-level preconditioner.

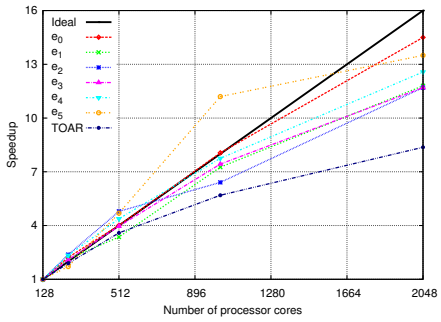

Figure 3: Speedup for **Mesh1** with two-level Schwarz preconditioner.

Table 5 reports the computed eigenvalues and the coarse meshes used in the two-level Schwarz preconditioner for each eigenvalue for **Mesh2**. As before, we only report the real part of the computed eigenvalue since the imaginary parts are less than 10^{-13}. Table 6 shows the numerical performance of JD with the one-level and two-level preconditioners for **Mesh2** in terms of the number of JD iterations, the average number of FGMRES for solving the correction equations and the compute time. It is clear that the two-level preconditioner is much better than the one-level preconditioner in the aspects of the average number of FGMRES iterations and the compute time. However, comparing Figure 4 with 5, it can be seen that the one-level preconditioner has better scalability than the

two-level preconditioner using over ten thousand processor cores although both preconditioners are scalable. Because the linear system on the coarse mesh Ω_c is very small and solved in parallel on all processor cores, the overall scalability of the two-level method suffers to a certain extent. The performance of the two-level method may be improved if a more efficient preconditioner, for example, two-level or multilevel preconditioner, is applied to the linear system on the coarse mesh.

Table 5: On **Mesh2**, the computed eigenvalue e_i and the coarse mesh Ω_c in the two-level preconditioner for each eigenvalue.

	e_i	Ω_c
0	0.4162094856604	$56 \times 56 \times 42$
1	0.5990754117523	$80 \times 80 \times 60$
2	0.5990754117522	$80 \times 80 \times 60$
3	0.7179731206719	$128 \times 128 \times 96$
4	0.7295254300372	$120 \times 120 \times 90$
5	0.7925318512082	$140 \times 140 \times 105$

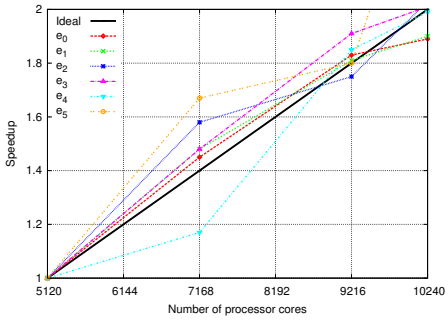

Figure 4: Speedup for **Mesh2** with one-level preconditioner.

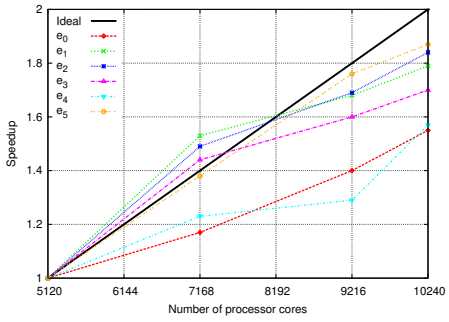

Figure 5: Speedup for **Mesh2** with two-level Schwarz preconditioner.

4.4. Convergence rate studies

To understand the convergence rate of JD, especially how it depends on the accuracy of the solution of the correction equation, we solve the Schrödinger equation (1) discretized on **Mesh1** using Algorithm 1 with the two-level Schwarz preconditioner on 128 processor cores. The coarse mesh Ω_c for the two-level preconditioner is $120 \times 120 \times 90$.

We first test the case when the stopping parameter ε_n for the correction equation on the fine mesh Ω_f changes with the residual of eigenpair. Let $\varepsilon_n = 0.1\|r_n\|_2$ if $\|r_n\|_2 \geq 10^{-3}$; otherwise, $\varepsilon_n = 10^{-4}$. We report the histories of the residual norm $\|r_n\|_2$ of the six eigenpairs in Table 7. It is shown that except

Table 6: On **Mesh2**, comparison of the results obtained by the one-level and two-level preconditioners. e_i is the i^{th} computed eigenvalue, n_p is the number of processor cores, JD is the number of JD iterations, FGMRES is the average number of FGMRES iterations for the solution of the correction equations, and Time is the total compute time in seconds.

		One-level			Two-level		
	n_p	JD	FGMRES	Time	JD	FGMRES	Time
e_0	5120	4	185.25	42.48	4	38.75	7.48
	7168	4	185.25	29.20	4	39.50	6.36
	9216	4	186.00	23.23	4	38.75	5.34
	10240	4	186.00	22.46	4	39.75	4.84
e_1	5120	4	251.75	71.10	4	34.75	9.47
	7168	4	255.75	47.96	4	34.25	6.20
	9216	4	254.50	39.29	4	34.50	5.64
	10240	4	256.50	37.37	4	34.00	5.29
e_2	5120	4	258.25	71.90	4	34.25	9.99
	7168	4	243.25	45.42	4	34.25	6.71
	9216	4	258.00	40.99	4	35.50	5.92
	10240	4	250.50	35.42	4	34.50	5.44
e_3	5120	6	391.50	220.52	6	35.00	21.29
	7168	6	391.67	148.76	6	33.33	14.77
	9216	6	392.00	115.58	6	32.67	13.29
	10240	6	391.67	109.52	6	34.17	12.53
e_4	5120	5	359.20	158.55	4	32.75	13.69
	7168	6	366.17	135.42	5	34.00	11.12
	9216	5	360.00	85.73	5	34.40	10.58
	10240	5	360.00	79.69	4	32.25	8.72
e_5	5120	9	394.89	333.04	5	30.40	18.34
	7168	8	396.25	199.63	5	33.20	13.33
	9216	9	400.00	185.20	5	25.40	10.43
	10240	7	393.86	126.93	5	26.20	9.82

Table 7: Residual norms of eigenpairs at each JD iteration for **Mesh1** using 128 processor cores in the case that ε_n changes with n. n represents the number of JD iteration.

n	e_0	e_1	e_2	e_3	e_4	e_5
0	1.581	2.436	2.436	6.426	3.215	4.908
1	$1.980 \cdot 10^{-1}$	$6.390 \cdot 10^{-1}$	$6.390 \cdot 10^{-1}$	1.874	$7.346 \cdot 10^{-1}$	2.224
2	$4.272 \cdot 10^{-3}$	$5.815 \cdot 10^{-2}$	$5.806 \cdot 10^{-2}$	$4.670 \cdot 10^{-1}$	$6.212 \cdot 10^{-2}$	$6.828 \cdot 10^{-1}$
3	$7.533 \cdot 10^{-7}$	$2.767 \cdot 10^{-4}$	$2.978 \cdot 10^{-4}$	$3.214 \cdot 10^{-2}$	$4.277 \cdot 10^{-4}$	$2.836 \cdot 10^{-2}$
4	$4.516 \cdot 10^{-11}$	$1.885 \cdot 10^{-8}$	$1.027 \cdot 10^{-8}$	$2.221 \cdot 10^{-4}$	$1.406 \cdot 10^{-8}$	$9.985 \cdot 10^{-5}$
5		$6.930 \cdot 10^{-13}$	$5.042 \cdot 10^{-13}$	$4.011 \cdot 10^{-8}$	$1.733 \cdot 10^{-11}$	$6.069 \cdot 10^{-9}$
6				$3.802 \cdot 10^{-12}$		$1.122 \cdot 10^{-12}$

the last JD iteration, JD achieves the quadratic convergence for e_1, e_2, e_3 and e_4, and super-quadratic convergence for e_0 and e_5.

Next, we consider a constant stopping criteria. In this case, we choose $\varepsilon_n = 10^{-4}$. Table 8 shows the histories of the residual norm of the six eigenpairs. It can be seen that JD achieves the super-quadratic convergence for e_0, e_1 and e_2 except the last iteration. Since the initial eigenpair of e_3 is less accurate, JD does not show the quadratic convergence immediately for e_3. The convergence rate of JD is very interesting for e_4 and e_5, it converges super quadratically in the first four iterations, and converges linearly in the last three iterations with a damping factor around 10^{-4}, which is equal to the stopping condition of the correction equation solver.

Table 8: Residual norms of eigenpairs at each JD iteration for **Mesh1** using 128 processor cores in the case that ε_n is equal to 10^{-4}. n represents the number of JD iteration.

n	e_0	e_1	e_2	e_3	e_4	e_5
0	1.581	2.436	2.436	6.426	3.215	4.908
1	$5.187 \cdot 10^{-2}$	$1.367 \cdot 10^{-1}$	$1.367 \cdot 10^{-1}$	3.817	$3.551 \cdot 10^{-1}$	$7.379 \cdot 10^{-1}$
2	$1.170 \cdot 10^{-4}$	$2.038 \cdot 10^{-4}$	$2.044 \cdot 10^{-4}$	4.297	$3.556 \cdot 10^{-3}$	$2.770 \cdot 10^{-2}$
3	$3.816 \cdot 10^{-9}$	$6.366 \cdot 10^{-9}$	$2.445 \cdot 10^{-8}$	1.035	$4.961 \cdot 10^{-7}$	$1.387 \cdot 10^{-5}$
4	$4.151 \cdot 10^{-13}$	$4.991 \cdot 10^{-13}$	$1.806 \cdot 10^{-12}$	$4.651 \cdot 10^{-2}$	$1.753 \cdot 10^{-10}$	$2.844 \cdot 10^{-9}$
5				$1.222 \cdot 10^{-4}$	$5.232 \cdot 10^{-13}$	$9.982 \cdot 10^{-13}$
6				$1.009 \cdot 10^{-8}$		
7				$9.956 \cdot 10^{-13}$		

To understand how the convergence rates of JD and FGMRES are related to the overlapping size δ, we run Algorithm 1 for **Mesh1** on 128 processor cores with the one-level and two-level preconditioners. It can be seen from Tables 9 and 10 that JD with various overlap sizes takes almost the same number of iterations for each eigenvalue when the residual norm of the solution of the correction equation is less than ε_n. For FGMRES convergence, the overlapping size has a greater impact on the one-level preconditioner than on the two-level preconditioner, since the coarse mesh of the two-level preconditioner provides more global information.

Table 9: The number of JD iterations and the average number of iterations of FGMRES with the one-level preconditioner (shown in brackets) for **Mesh1** using 128 processor cores for different δ. The stopping criteria ε_n of FGMRES is equal to 10^{-4}.

δ	e_0	e_1	e_2	e_3	e_4	e_5
0	4(127.25)	4(179.75)	4(179.75)	6(388.5)	4(286.5)	6(382.83)
1	4(102.75)	4(138.75)	4(133)	7(355.29)	4(230)	5(304.2)
2	4(100.25)	4(134.75)	4(126)	7(330)	4(224)	5(288.2)
3	4(99.5)	4(134)	4(126.75)	7(333)	4(224.5)	5(305.2)
4	4(99.5)	4(134)	4(126.75)	7(335.43)	4(223.5)	5(304)

We report the compute time and the number of JD iterations in Table 11 for the cases that the number of iterations of FGMRES with the two-level precon-

Table 10: The number of iterations of JD and the average number of iterations of FGMRES with the two-level preconditioner (shown in brackets) for **Mesh1** using 128 processor cores for different δ. The stopping criteria ε_n of FGMRES is equal to 10^{-4} and the coarse mesh is $120 \times 120 \times 90$.

δ	e_0	e_1	e_2	e_3	e_4	e_5
0	4(10.25)	4(10.25)	4(10.25)	8(14.5)	5(29.2)	5(12.8)
1	4(6.75)	4(6.75)	4(6.5)	7(9.86)	5(8.2)	5(8.6)
2	4(6.75)	4(6.75)	4(6.75)	7(9.43)	4(7.5)	5(8.4)
3	4(6.75)	4(6.75)	4(6.5)	8(9.63)	5(8.2)	5(8.6)
4	4(6.75)	4(6.75)	4(6.5)	8(14.25)	5(12.2)	5(8.8)

ditioner is fixed. Compared to Table 3, we see that one can find a fixed number that leads to less compute time, if it is chosen carefully.

Table 11: The number of JD iterations and compute time in seconds (shown in brackets) for **Mesh1** using 128 processor cores with various coarse meshes as showed in Table 2 and several fixed numbers of iterations of FGMRES. The notation *it* denotes the fixed number of iterations of FGMRES and N/A means that Algorithm 1 fails to converge within 50 JD iterations.

it	e_0	e_1	e_2	e_3	e_4	e_5
5	13(23.99)	15(27.97)	38(79.83)	8(27.57)	N/A	6(21.23)
10	7(18.28)	7(20.49)	11(25.39)	7(44.57)	32(96.20)	5(24.61)
20	4(21.03)	4(19.83)	5(22.28)	7(99.93)	10(41.22)	4(39.70)
30	4(33.27)	4(29.43)	4(30.75)	7(157.70)	7(51.28)	4(65.56)
40	4(41.58)	4(41.90)	4(41.04)	7(225.73)	6(64.01)	4(109.93)
50	4(63.99)	4(56.04)	4(54.45)	7(265.38)	4(66.11)	4(130.19)
60	4(73.78)	5(94.95)	4(72.19)	7(356.07)	4(70.71)	4(194.22)

In addition, we test JD with the various initial guess eigenvectors, but still use the eigenvalue obtained on Ω_0 as the initial guess eigenvalue. For **Mesh1**, JD converges to the undesired eigenvalue when the one-vector is used as the initial guess vector, and JD can not converge when the random vector is used as the initial guess vector.

5. Conclusion

In this paper, a parallel domain decomposition based JD algorithm was introduced and studied for the pyramidal quantum dot simulation. In the proposed method, the correction equation is solved efficiently to a certain level of accuracy by FGMRES with a projected two-level Schwarz preconditioner. Compared with the one-level method and TOAR, the two-level method reduces greatly the total compute time and the number of iterations for solving the correction equations. It is worth mentioning that, in addition to the coarse mesh for the preconditioner, another coarse mesh is introduced for generating the initial guess for the JD iterations. Numerical experiments confirmed that our method converges quadratically, and also is scalable for problems with over 160

millions unknowns on a parallel computer with over 10,000 processor cores. The algorithm can be extended to multilevels if the coarse problem in the two-level preconditioner is solved by two-level or multilevel methods.

Acknowledgement

We would like to thank the referees for the helpful comments that improve the paper. The first author and the third author were supported by the Shenzhen Peacock Plan grant KQCX20130628112914303. The second author was supported by the grant MOST-100-2115-M-008-008-MY2.

References

[1] F.-N. Hwang, Z.-H. Wei, T.-M. Huang, W. Wang, J. Comput. Phys. 229 (2010) 2932–2947.

[2] M. Hochbruck, D. Löchel, SIAM J. Sci. Comput. 32 (2010) 3151–3169.

[3] T.-M. Hwang, W.-W. Lin, W.-C. Wang, W. Wang, J. Comput. Phys. 196 (2004) 208–232.

[4] Y. Saad, J. R. Chelikowsky, S. M. Schontz, SIAM Rev. 52 (2010) 3–54.

[5] P. Arbenz, M. Bečka, R. Geus, U. Hetmaniuk, T. Mengotti, Parallel Computing 32 (2006) 157–165.

[6] Y.-T. Feng, Comput. Methods in Appl. Mech. Engrg. 190 (28) (2001) 3543–3563.

[7] Y. Su, Z. Bai, SIAM J. Matrix Anal. Appl. 32 (2011) 201–216.

[8] G. L. G. Sleijpen, H. van der Vorst, SIAM J. Matrix Anal. Appl. 17 (1996) 401–425.

[9] G. L. G. Sleijpen, A. G. L. Booten, D. R. Fokkema, H. van der Vorst, BIT 36 (1996) 595–633.

[10] T.-M. Huang, F.-N. Hwang, S.-H. Lai, W. Wang, Z.-H. Wei, Comput. Fluids 45 (2011) 207–214.

[11] T.-M. Hwang, W.-W. Lin, J.-L. Liu, W. Wang, Numer. Linear Algebra Appl. 12 (2005) 605–624.

[12] Y. Saad, Iterative Methods for Sparse Linear Systems, SIAM, Philadelphia, 2003.

[13] A. Quarteroni, A.Valli, Domain Decomposition Methods for Partial Differential Equations, Clarendon Press, Oxford, 1999.

[14] B. F. Smith, P. E. Bjørstad, W. Gropp, Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations, Cambridge University Press, New York, 1996.

[15] A. Toselli, O. Widlund, Domain Decomposition Methods: Algorithms and Theory, Springer-Verlag, Berlin, 2005.

[16] S. Balay, J. Brown, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, H. Zhang, PETSc Users Manual, v. 3.5.2, Argonne National Laboratory (2014).

[17] J. E. Roman, C. Campos, E. Romero, A. Tomas, SLEPc Users Manual, v. 3.5.3, Universitat Politècnica de València (2014).

[18] L.-W. Wang, J. Kim, A. Zunger, Phys. Rev. B 59 (1999) 5678–5687.

[19] J.-L. Liu, J.-H. Chen, O. Voskoboynikov, Comput. Phys. Commun. 175 (2006) 575–582.

[20] X. Dai, X. Gong, Z. Yang, D. Zhang, A. Zhou, Multiscale Model. Simul. 9 (2011) 208–240.

[21] C. Vömel, S. Z. Tomov, O. A. Marques, A. Canning, L.-W. Wang, J. J. Dongarra, J. Comput. Phys. 227 (2008) 7113–7124.

[22] W. Wang, T.-M. Hwang, W.-W. Lin, J.-L. Liu, J. Comput. Phys. 190 (2003) 141–158.

[23] G. H. Golub, C. F. V. Loan, Matrix Computation, 3rd Edition, John Hopkins University Press, Baltimore, 1996.

[24] J. Olsen, P. Jørgensen, J. Simons, Chem. Phys. Lett. 169 (1990) 463–472.

[25] G. L. G. Sleijpen, W. Wubs, SIAM J. Sci. Comput. 25 (2003) 1249–1272.

[26] X.-C. Cai, O. Widlund, SIAM J. Numer. Anal. 30 (1993) 936–952.

[27] X.-C. Cai, X. Li, SIAM J. Sci. Comput. 33 (2011) 746–762.

[28] X.-C. Cai, M. Sarkis, SIAM J. Sci. Comput. 21 (1999) 792–797.

[29] X.-C. Cai, W. Gropp, D. Keyes, R. Melvin, D. Young, SIAM J. Sci. Comput. 19 (1998) 246–265.

[30] P. Havé, R. Masson, F. Nataf, M. Szydlarski, H. Xiang, T. Zhao, SIAM J. Sci. Comput. 35 (2013) C284–C302.

[31] M. E. Hochstenbach, G. L. G. Sleijpen, Numer. Lin. Alg. Appl. 15 (2008) 35–54.

[32] P. N. Brown, H. F. Walker, SIAM J. Matrix Anal. Appl. 18 (1997) 37–51.