Journal of Computational Physics 229 (2010) 2932–2947

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

A parallel additive Schwarz preconditioned Jacobi–Davidson algorithm for polynomial eigenvalue problems in quantum dot simulation

Feng-Nan Hwang [a], Zih-Hao Wei [a], Tsung-Ming Huang [b], Weichung Wang [c,*]

[a] Department of Mathematics, National Central University, Jhongli 320, Taiwan
[b] Department of Mathematics, National Taiwan Normal University, Taipei 116, Taiwan
[c] Department of Mathematics, National Taiwan University, Taipei 106, Taiwan

ARTICLE INFO

Article history:
Received 9 July 2009
Received in revised form 3 December 2009
Accepted 21 December 2009
Available online 4 January 2010

Keywords:
Parallel computing
Restricted additive Schwarz preconditioning
Jacobi–Davidson methods
Polynomial eigenvalue problems
Schrödinger's equation
Quantum dot simulation

ABSTRACT

We develop a parallel Jacobi–Davidson approach for finding a partial set of eigenpairs of large sparse polynomial eigenvalue problems with application in quantum dot simulation. A Jacobi–Davidson eigenvalue solver is implemented based on the Portable, Extensible Toolkit for Scientific Computation (PETSc). The eigensolver thus inherits PETSc's efficient and various parallel operations, linear solvers, preconditioning schemes, and easy usages. The parallel eigenvalue solver is then used to solve higher degree polynomial eigenvalue problems arising in numerical simulations of three dimensional quantum dots governed by Schrödinger's equations. We find that the parallel restricted additive Schwarz preconditioner in conjunction with a parallel Krylov subspace method (e.g. GMRES) can solve the correction equations, the most costly step in the Jacobi–Davidson algorithm, very efficiently in parallel. Besides, the overall performance is quite satisfactory. We have observed near perfect superlinear speedup by using up to 320 processors. The parallel eigensolver can find all target interior eigenpairs of a quintic polynomial eigenvalue problem with more than 32 million variables within 12 minutes by using 272 Intel 3.0 GHz processors.

1. Introduction

Polynomial eigenvalue problems (PEPs) arise in many scientific and engineering applications. A PEP can be written as

$$\mathcal{A}(\lambda)f \equiv \left(\sum_{i=0}^{\tau} \lambda^i A_i\right) f = 0, \tag{1}$$

where (λ, f) is an eigenpair that $\lambda \in \mathbb{C}$ and $f \in \mathbb{C}^{\mathcal{N}}$, τ is the degree of the matrix polynomial, and $A_i \in \mathbb{R}^{\mathcal{N}} \times \mathcal{N}$ are the corresponding coefficient matrices. Solving some PEPs can be a computational challenge, especially when the PEPs are very large and only a small number of eigenpairs are of interest. In this article, we develop a parallel Jacobi–Davidson approach for solving PEPs with a particular application in quantum dot (QD) simulation.

Nanoscale semiconductor QD is a particular electronic structure in which the carriers are confined within the dots in three dimensions. A great deal of research related to the laboratory fabrication, theoretical investigation, and real-world applications of QDs has been conducted. Among these studies, numerical simulations based on Schrödinger equations play an important role in exploring properties of QDs. A computational focus of QD simulations is to solve the eigenvalue prob-

* Corresponding author. Tel.: +886 2 33662871; fax: +886 2 23914439.
E-mail addresses: hwangf@math.ncu.edu.tw (F.-N. Hwang), socrates.wei@gmail.com (Z.-H. Wei), min@math.ntnu.edu.tw (T.-M. Huang), wwang@math.ntu.edu.tw (W. Wang).

0021-9991/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2009.12.024

lems that are derived from the discretized Schrödinger equations. Various discretization schemes such as finite differences [25,32], finite volumes [22], finite elements [28,45], boundary elements [15], scanning methods [12], and pseudo-spectral methods [14,31] have been used. Furthermore, various physical models and numerical schemes result in different standard [19,22], generalized [21], polynomial [19,49], or rational [45,46] eigenvalue problems. Such eigenvalue problems are then solved to find approximate energy levels (eigenvalues) and wave or envelope functions (eigenvectors).

Many eigensolvers have been proposed to solve the eigenvalue problems associated with the QD simulations [6,7,13,19–22,27,40,41,43,44,48,49]. In the literature, some sequential and parallel eigensolvers have been proposed for use in solving standard or generalized eigenvalue problems. On the other hand, sequential Jacobi–Davidson type methods have also been used to solve the PEPs. As an extension of these previous efforts, we aim to develop and implement an efficient parallel Jacobi–Davidson (JD) algorithm for solving PEPs.

Use of the JD algorithms to compute interior eigenpairs of PEPs can be justified as follows. To solve a PEP, one can recast the original PEP as a linearized eigenvalue problem. The linearized eigenvalue problem can be solved using Arnoldi or Lanczos type methods. From the numerical viewpoint, however, this approach has some potential drawbacks. For example, since the dimension of the enlarged matrix is much larger than the original problem, the memory requirement is more demanding and the linearized eigenvalue problem is more ill-conditioned [39]. Furthermore, in order to find selected interior eigenvalues, Arnoldi or Lanczos type methods need to be used in conjunction with a shift-and-invert technique, which requires to solve linear systems with a certain degree of accuracy. The computational cost of this type of operation is quite expensive. On the other hand, the use of a JD type algorithm is an efficient alternative. JD was proposed by Sleijpen and Van der Vorst for solving linear eigenvalue problems [35] and later extended for higher degree PEPs [34,47,49]. The JD algorithm belongs to a class of subspace methods that consists of two key steps. One first extracts an approximate eigenpair from a given search space using the Rayleigh–Ritz procedure. If the approximate eigenpair is not close enough, one must enlarge the search space by adding a new basis vector, which is the approximate solution of a large sparse linear system of equations, known as the correction equation. The major advantages of the JD algorithm are that it deals directly with PEPs, the interior spectrum can be found without using any shift-and-invert technique, and an inexact correction equation solve (very often requiring only a few linear iterations) is sufficient for rapid JD convergence.

Parallelism is a natural choice as a method for accelerating large-scale JD eigensolvers. Some variants of parallel JD algorithms have been studied for use in solving linear and quadratic eigenvalue problems. In most existing parallel JD research, the focus has been on design of appropriate parallel preconditioners for the correction equations, the most expensive part of JD algorithms. For example, Nool and Ploeg [29,30] first used a shift-and-invert technique to convert a generalized magnetohydrodynamics eigenvalue problem into a standard eigenvalue problem so that the Alfven spectrum, (which are certain interior eigenvalues) is shifted to dominant one then solved it by a parallel JD methods, in which Generalized Minimal Residual Method (GMRES) [33] with an inexact block Cholesky's based preconditioner was used for the correction equation. Bergamaschi et al. [4,5] reported some computational experiences with a parallel approximate inverse (AINV) preconditioned JD method for solving the standard symmetric eigenvalue problem using a variety of applications in the 3D porous media flow, the 2D Richard's equation and the 3D Laplacian equation. The parallel efficiency of their parallel JD code achieved ranges from 26% to 62%. Arbenz et al. [1] 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; their parallel code did not attain below 65% parallel efficiency using up to 16 processors. In addition, Gijzen [16] implemented a parallel JD method for a quadratic acoustic with damping eigenvalue problem, in which the GMRES method is used without any preconditioning for the solution of the correction equation and showed that the method scales almost linearly up to 64 processors. Note that several state-of-the-art parallel eigensolver packages such as PARPACK [24], BLOPEX [23], PRIMME [37,38], and Scalable Library for Eigenvalue Problem Computation (SLEPc) [17] are publicly available for standard or generalized eigenvalue problems.

In order to develop an efficient parallel preconditioned Jacobi–Davidson algorithm for solving PEPs arising in QD simulations, we address the following issues:

- *Preconditioning for the correction equations.* We propose using additive Schwarz type preconditioners to solve the correction equations; this represents the most expensive part of the proposed algorithm. This type of preconditioner is efficient, widely used, and is well-understood as it applies to solving linear systems arising in partial differential equations such as symmetric positive elliptic problems, non-symmetric and indefinite elliptic problems, parabolic convection-diffusion, and hyperbolic problems [36,42]. However, its use in solving eigenvalue problems for specific applications has not yet been studied.
- *Parallel implementation.* We discuss our parallel implementations on a distributed memory parallel computer in terms of a parallel correction equation solver and parallel basic linear algebraic operations. Our implementation is mainly based on the Portable, Extensible Toolkit for Scientific Computation (PETSc) [2] and thus shares all the convenient features provided by PETSc.
- *Parameter tuning.* We identify tunable parameter combinations for achieving best performances. The parameters include overlapping size of additive Schwarz preconditioner, solution quality of subdomain problems, inner JD iteration (for the correction equations) stopping criterion, and scalability with respect to number of processors and problem sizes.
- *Parallel performance.* Numerical results exhibit superior performance in speedup and parallel efficiency, since the additive Schwarz preconditioner in conjunction with a Krylov subspace method (e.g. GMRES) can improve the overall convergence

rate of the proposed algorithm. For example, we have observed superlinear speedup and over 100% parallel efficiency up to 320 processors.

- *Performance on a large-scale problem.* Our parallel eigensolver also solves a quintic PEP with more than 32 million variables. The eigensolver computes all the five target eigenpairs within 12 minutes by using 272 Intel 3.0 GHz processors.

The remainder of this paper is organized as follows: In Section 2, we describe the mathematical model of QDs and the corresponding PEPs. Section 3 describes the parallel additive Schwarz preconditioned JD algorithm for solving PEPs. Section 4 discusses our implementation of the proposed algorithm using PETSc and SLEPc on a distributed memory parallel computer. Section 5 reports parallel performance of the proposed algorithms. The concluding remarks are given in Section 6.

2. Mathematical model for quantum dot and its discretization

The governing equation for three-dimensional QDs with single particle in the conduction band is the time-independent Schrödinger equation.

$$-\nabla\cdot\left(\frac{\hbar^2}{2m(\mathbf{x})}\nabla f\right)+c(\mathbf{x})f=\lambda f, \tag{2}$$

where $\hbar$ is the reduced Planck constant; the eigenvalue λ and eigenvector f represent the total electron energy and the corresponding wave function, respectively; $m(\mathbf{x})$ represents the electron effective mass and $c(\mathbf{x})$ is the confinement potential.

The above equation describes the hetero-structures as shown in Fig. 1. In the interface of the hetero-junction, both $m(\mathbf{x})$ and $c(\mathbf{x})$ are discontinuous. In cases in which constant effective mass models are considered, $m(\mathbf{x})$ is a piecewise constant function with respect to the space variable $\mathbf{x}$:

$$m(\mathbf{x})=\begin{cases} m_1 & \text{in the dot,}\\ m_2 & \text{in the matrix.}\end{cases}$$

Alternatively, the nonparabolicity of the electron's dispersion relation [11] gives the effective mass as a rational function of the energy. In particular, the effective mass model becomes

$$\frac{1}{m_\ell(\lambda)}=\frac{P_\ell^2}{\hbar^2}\left(\frac{2}{\lambda+g_\ell-c_\ell}+\frac{1}{\lambda+g_\ell-c_\ell+\delta_\ell}\right), \tag{3}$$

for $\ell=1,2$. Here, P_ℓ, g_ℓ, and δ_ℓ are the momentum, main energy gap, and spin-orbit splitting in the ℓth potential $c(\mathbf{x})$ is a piecewise constant function that

$$c(\mathbf{x})=\begin{cases} c_1 & \text{in the dot,}\\ c_2 & \text{in the matrix.}\end{cases}$$

We impose the Ben Daniel–Duke interface conditions

$$f|_{D_+}=f|_{D_-},\quad \frac{\hbar^2}{2m_2}\frac{\partial f}{\partial n}|_{\partial D_+}=\frac{\hbar^2}{2m_1}\frac{\partial f}{\partial n}|_{\partial D_-}\left(\text{or }\frac{\hbar^2}{2m_2(\lambda)}\frac{\partial f}{\partial n}|_{\partial D_+}=\frac{\hbar^2}{2m_1(\lambda)}\frac{\partial f}{\partial n}|_{\partial D_-}\right). \tag{4}$$

By letting D be the QD domain, we use D_+ and D_- to denote the outward normal derivatives of the interface for the matrix and dot, respectively. Additionally, we apply homogeneous Dirichlet conditions on the boundary of the quantum matrix.

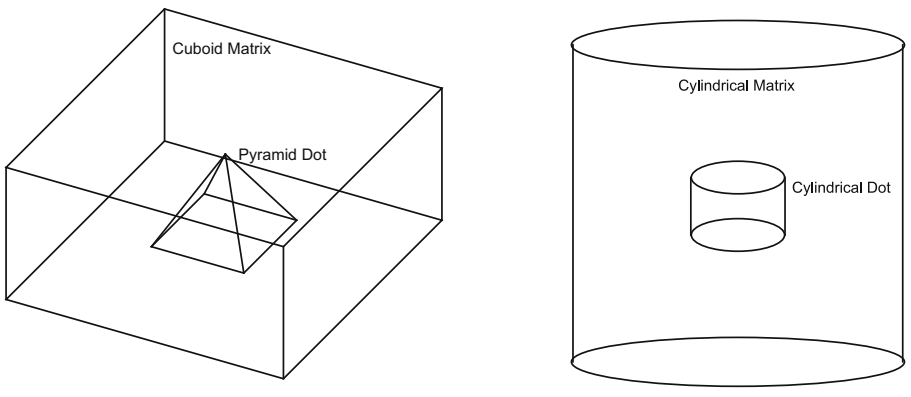


Fig. 1. Structure schema of a pyramidal and a cylindrical quantum dot. Each of the quantum dots is embedded in a hetero-structure matrix.

Table 1
Some representative QD eigenvalue problems available in literature. FDM: finite differences methods, FVM: finite volumes methods, and FEM: finite elements methods.

Type	QD geometry	Effective mass model	Discretization	Reference(s)
Standard	Cylindrical	Constant	FVM	[22]
Standard	Pyramidal	Constant	FVM	[19]
Generalized	Irregular	Constant	FDM	[21]
Cubic	Cylindrical	Nonparabolic	FDM	[49]
Quintic	Pyramidal	Nonparabolic	FVM	[19]
Rational	Conical/pyramidal	Nonparabolic	FEM/FDM	[26,45,46]

This QD model can be derived from the following approximations. We first take account of the Hamiltonian acting on the envelope functions for multi-band electrons, in which the envelope functions are equivalent to the $\mathbf{k}\cdot\mathbf{p}$ bandstructure Kane model [3,10]. The effective mass theory is further applied to project the multi-band Hamiltonian onto the conduction band and consequently results in the single Hamiltonian eigenvalue problem (2). Such approximations are desirable as they can reduce the intensive computational burden from more complicated yet complete models like the *ab initio* simulations of many-electron Schrödinger equations. However, the simplifications also limit the applicability of the model and consequently the numerical methods considered in this article.

A variety of numerical schemes for simulating QDs with different geometries have been developed for cylinders [20,22,49], cones [26,46], pyramids [12,19,32,48], and irregular shapes [21]. Based on the variants of the QD model and numerical schemes, PEPs with different degrees are derived. Table 1 lists six representative QD eigenvalue problems available in the literature. Numerical investigations on the comparison of our proposed eigensolver with other state-of-the-art parallel eigensolvers for first three linear problems in the table will be reported elsewhere. Instead, we focus on the two higher degree PEPs: quintic and cubic eigenvalue problems, which correspond to the pyramidal and cylindrical QD, respectively and both problems assume the nonparabolic effective mass model. Alternatively, the QD eigenvalue problem (2) in the rational form considered in [45,46] is solved by the nonlinear Arnoldi methods, the JD methods, and fully approximation methods.

For the pyramidal QD case, the width of the QD base is 12.4 nm, the height of the QD is 6.2 nm and the cuboid matrix of the dimension $24.8 \times 24.8 \times 18.6$ nm is uniformly partitioned into $(L+1)$, $(M+1)$, and $(N+1)$ grids in each direction so that the grid size $\Delta x = 24.8/L$, $\Delta y = 2.48/M$, and $\Delta z = 18.6/N$. Since homogeneous Dirichlet boundary conditions are imposed, the total number of unknowns, or the dimension of $A_i's$, is therefore $(L-1)\times(M-1)\times(N-1)$. The resulting quintic PEPs are obtained using a finite volumes method. For the cylindrical QD case, the cylindrical QD of diameter 15 nm and height 2.5 nm is placed in the cylindrical matrix of radius 75 nm and height 12.5 nm. The matrix is non-uniformly partitioned into $(\rho+1)$, η, and $(\zeta+1)$ grids in the radial, azimuthal, and axial directions, respectively, where the Schrödinger equation is discretized by using finite differences schemes. By applying the periodicity in the azimuthal direction, suitable permutations, and Fourier Transformation, η independent 2D cubic PEPs can be decoupled from a 3D eigenvalue problem. As a test case, only the grid associated with azimuthal index 1 is considered. The derivations of these two PEPs are quite tedious; we refer interested readers to the references given in Table 1.

3. A parallel additive Schwarz preconditioned Jacobi–Davidson algorithm

We propose a parallel additive Schwarz preconditioned Jacobi–Davidson algorithm (ASPJD) for solving the PEP (1) as shown in Algorithm 1. The ASPJD shares a similar framework to that of other polynomial Jacobi–Davidson type algorithms such as [19]. Here, we focus on the efficiency of the algorithm on parallel computers. Its parallel implementation will be discussed in the next section. The algorithm contains two loops. In the inner *while*-loop between line 4 and 12, we compute the Ritz pair that approximates the jth desired eigenpair of (1). In the outer *for*-loop between line 2 and 16, we compute the k desired eigenpairs one-by-one using a locking technique.

Algorithm 1. Parallel Additive Schwarz Preconditioned Jacobi–Davidson Algorithm (ASPJD) for Polynomial Eigenvalue Problems

Input: Coefficient matrices A_i for $i = 0,\ldots,\tau$, the number of desired eigenvalues k, an initial orthonormal vector V_{ini}
Output: The desired eigenpairs (λ_j, f_j) for $j = 1,\ldots,k$
1 Set $V = [V_{int}]$, $V_F = [\ ]$, and $\Lambda = \emptyset$
2 **for** $j = 1$ to k **do**
3 Compute $W_i = A_i V$ and $M_i = V^* W_i$ for $i = 0,\ldots,\tau$
4 **while** (user-defined stopping criteria are not satisfied) **do**
5 Compute the eigenpairs (θ, s) of $\left(\sum_{i=0}^{\tau}\theta^i M_i\right)s = 0$
6 Select the desired eigenpair (θ, s) with $\|s\|_2 = 1$ and $\theta \notin \Lambda$

(*continued on next page*)

7 Compute $u = Vs, p = \mathcal{A}'(\theta)u, r = \mathcal{A}(\theta)u$
8 Solve the correction equation $\left(I - \frac{pu^*}{u^*p}\right)\mathcal{A}(\theta)(I - uu^*)t = -r$ approximately for $t \perp u$ by a Krylov subspace method with an additive Schwarz type preconditioner
9 Orthogonalize t against $V, v = t/\|t\|_2$
10 Compute $w_i = A_i v$, $M_i = \begin{bmatrix} M_i & V^*w_i \\ v^*W_i & v^*w_i \end{bmatrix}$ for $i = 0, \ldots, \tau$
11 Expand $V = [V, v]$ and $W_i = [W_i, w_i]$
12 **end while**
13 Set $\lambda_j = \theta$, $f_j = u, \Lambda = \Lambda \cup \{\lambda_j\}$
14 Perform locking by orthogonalizing f_j against V_F; Compute $f_j = f_j/\|f_j\|_2$; Update $V_F = [V_F, f_j]$
15 Choose an orthonormal matrix $V_{ini} \perp V_F$; Set $V = [V_F, V_{ini}]$
16 **end for**

3.1. Inner loop

The inner loop contains the following components:

- *Small size projected eigenvalue problems*. In the **while**-loop starting from line 4, we compute the eigenpair (θ, s), where $\|s\|_2 = 1$ of the projected eigenvalue problem, $(\sum_{i=0}^{\tau} \theta^i M_i)s = 0$, by solving the corresponding linearized projected eigenvalue problem,

$$M_A z = \theta M_B z, \tag{5}$$

where

$$M_A = \begin{bmatrix} 0 & I & 0 & \ldots & 0 \\ 0 & 0 & I & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & I \\ M_0 & M_1 & M_2 & \ldots & M_{\tau-1} \end{bmatrix}, \quad M_B = \begin{bmatrix} I & 0 & 0 & \ldots & 0 \\ 0 & I & 0 & \ldots & 0 \\ 0 & 0 & I & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & -M_\tau \end{bmatrix}, \quad z = \begin{bmatrix} s \\ \theta s \\ \theta^2 s \\ \vdots \\ \theta^{\tau-1} s \end{bmatrix}.$$

Let Θ be the set of the eigenvalues θ. For a given target value, μ, which is near the desired eigenvalue, we select one of eigenvalues, say θ_j, from the set $\Theta \backslash \Lambda$ so that

$$|\mu - \theta_j| = \min_{\theta \in \Theta \backslash \Lambda} |\mu - \theta|. \tag{6}$$

(θ_j, s_j) refers to the desired eigenpair in line 6 of Algorithm 1. Depending on the application some additional condition for (6) may need to be imposed. For example, we typically select $\mu = 0$ and require θ to be positive for our QD simulation. Note that the dimension of $V^T\mathcal{A}(\theta)V$ is usually small and not larger than a user-defined restarting number.

- *Restarting*. To avoid loss of numerical orthogonality in V and to keep a manageable size of the generalized eigenvalue problem being solved in (5), we restart the eigenpair search and choose a new orthogonal V after a certain number of inner iterations.
- *Correction vector*. The main purpose of the inner loop is to keep searching a Ritz pair with small residual over the space V that is gradually expanded. As shown in line 7 and 8 of Algorithm 1, we compute

$$p = \mathcal{A}'(\theta)u, \tag{7}$$

where $\mathcal{A}'(\theta) = \sum_{i=1}^{\tau} i\theta^{i-1}A_i$ and then solve the correction equation

$$\left(I - \frac{pu^*}{u^*p}\right)\mathcal{A}(\theta)(I - uu^*)t = -r \tag{8}$$

for a correction vector t in each inner *while*-loop and then append the normalized correction vector to V. The derivation of Eq. (8) is based on Taylor's expansion and the fact that we can find the orthogonal complement $t \perp u$ to correct the error of the current approximation u such that

$$\mathcal{A}(\lambda)(u + t) = 0. \tag{9}$$

See [18] for a detailed explanation regarding the correction vector.

Here, the correction Eq. (8) is solved approximately by a Krylov subspace method (e.g. GMRES) with the preconditioning operator, B_d^{-1}, where

$$B_d = \left(I - \frac{pu^*}{u^*p}\right)B(I - uu^*) \approx \left(I - \frac{pu^*}{u^*p}\right)\mathcal{A}(\theta)(I - uu^*), \tag{10}$$

and B is an approximation of $\mathcal{A}(\theta)$. In practice, there is no need to explicitly form B_d to solve $B_d z = y$ with $z \perp u$ for a given y, as it can be done equivalently by computing

$$z = B^{-1}y - \eta B^{-1}p, \quad \text{with} \quad \eta = \frac{u^*B^{-1}y}{u^*B^{-1}p}. \tag{11}$$

Note that the preconditioning operation $B^{-1}p$ and inner product $u^*B^{-1}p$ need be computed only once for solving each correction equation; there is no need to re-compute them in the Krylov subspace iteration. Consequently, the Krylov subspace iteration in line 8 needs only preconditioning operations in the form of $B^{-1}y$.

While some sequential preconditioners such as SSOR were studied in [19,20], we consider next a parallel domain-decomposed preconditioner B^{-1} based on an additive Schwarz framework.

- *Construction of the additive Schwarz preconditioner* B^{-1}. To define parallel Schwarz-type preconditioners, we begin by partitioning the computational quantum dot domain D into N_s disjoint subdomains $\{D_i^0, i = 1, \ldots, N_s\}$, whose union covers the entire domain. Then, overlapping subdomains can be obtained by expanding each subdomain D_i^0 to a larger subdomain D_i^δ with the boundary ∂D_i^δ. Here δ is a nonnegative integer indicating the level of overlap. Let n_i^δ and n be the total number of interior grid points on D_i^δ and D, respectively. We define R_i^δ as a $n_i^\delta \times n$ *restriction* matrix with a value of either 0 or 1. The *interpolation* operator $(R_i^\delta)^T$ can then be defined as the transpose of R_i^δ. The multiplication of R_i^δ (and $(R_i^\delta)^T$) with a vector does not involve any arithmetic operations, but does involve some communication in a distributed memory implementation, i.e., the global-to-local restriction operator R_i^δ, which collects data from neighboring subdomains, and the local-to-global interpolation operator $(R_i^\delta)^T$, which sends partial solutions to neighboring subdomains.

Using the *restriction* and *interpolation* matrices, we write the one-level restricted additive Schwarz preconditioner (RAS) [9] in matrix form as

$$B^{-1} = \sum_{i=1}^{N_s} (R_i^0)^T B_i^{-1} R_i^\delta, \tag{12}$$

where B_i^{-1} is the subspace inverse of B_i and $B_i = R_i^\delta \mathcal{A}(\theta)(R_i^\delta)^T$. Note that when $\delta = 0$, the RAS preconditioner in (12) is reduced to the block Jacobi preconditioner.

Although the theoretical analysis for the convergence of RAS is still incomplete, much numerical evidence suggests that using RAS to solve a variety of linear systems can not only reduce by half the communication time needed when the classical additive Schwarz preconditioner is used, but can also accelerate the convergence of iterative methods.

3.2. Outer loop

In the outer loop, k desired eigenpairs are computed successively. To compute the $(j+1)$th eigenpair, we apply the following locking technique with suitably chosen orthonormal searching space V. Such technique is incorporated in Algorithm 1 as follows:

- In line 13, we take a Ritz pair (θ_j, f_j), where $f_j = V s_j$, with small residual $\|\mathcal{A}(\theta_j)f_j\|_2$ as the jth approximate eigenpair and append θ_j into Λ.
- In line 14, the approximate eigenvector f_j is normalized and then appended into $V_{\mathbf{F}}$. Consequently, the columns of $V_{\mathbf{F}}$ form an orthonormal basis of the eigenspace spanned by $f_1, \ldots, f_j$.
- In line 15, we choose an orthonormal matrix V_{ini} and set $V = [V_{\mathbf{F}}, V_{ini}]$ such that $V^TV = I$.

In so doing, the first desired j eigenpairs will be included (or *locked*) in the eigenpairs corresponding to the projected eigenvalue problem in line 3 of Algorithm 1. On the other hand, these j eigenvalues will not be chosen in line 6, as we search the $(j+1)$th eigenvector over the space that is orthogonal to the V defined in line 15 of the jth iteration.

4. Detailed parallel implementations using PETSc and SLEPc

In this section, we describe the parallel implementation of the ASPJD algorithm based on two powerful scientific software libraries, namely the PETSc [2] and the SLEPc [17]. As shown in Fig. 2, the design of PETSc adopts the principle of *software layering*. As an application code of PETSc, the major component in our ASPJD eigensolver, the JD object, is built on top of the KSP, a Linear Equation Solver. All PETSc libraries are based on Message Passing Interface (MPI) and two modules of linear algebra libraries: Basic Linear Algebra Subproblems (BLAS) and Linear Algebra Packages (LAPACK) library. The vector (Vec) and matrix (Mat) are two basic objects in PETSc. The eigenvector x_l and other working vectors are created as parallel vectors in the Vec object. The column vectors of V and W_i in line 3 and 11 of Algorithm 1 are stored as an array of parallel vectors. The

coefficient matrices A_i and the matrix $\mathcal{A}(\theta)$ are created in a parallel sparse matrix format. We do explicitly form $\mathcal{A}(\theta)$ using parallel matrix–matrix addition and it is used in the construction of a preconditioner.

We discuss parallel implementations of the ASPJD algorithm in detail, as follows, by classifying the algorithm into three main parts: (1) linearized projected eigenvalue solve, (2) correction equation solve, and (3) a sequence of basic linear algebraic operations.

- *Redundant linearized projected eigenvalue solve*. As mentioned in the previous section, the linearized projected eigenvalue problem (5) is typically quite small. Solving such small problems in parallel, especially in cases where large numbers of processors are used, may result in large computational cost in data communications compared to floating-point operations. Therefore, we adopts another approach. On each processor, the sequential QZ routine, called ZGGEVX in LAPACK, is employed to redundantly solve the same linearized projected eigenvalue problem, $M_A z = \theta M_B z$, through an interface provided by SLEPc [17]. Here, the matrices M_A and M_B, as well as M_i, are stored in the sequential dense matrix format and their sizes increase as ASPJD iterates. Additional blocks $V^* w_i$ (or $v^* W_i$) and $v^* w_i$ to be inserted into new M_i are computed in parallel; the results are then distributed to each processor via the vector multiple and single inner product, VecMDot and VecDot, respectively.
- *Parallel correction equation solve*. We use a Krylov subspace method (e.g., GMRES) in conjunction with preconditioner (10) and the RAS preconditioner B^{-1} in (12) to approximately solve the correction Eq. (8). We implement the parallel correction equation solve mainly by using the following three PETSc objects. First, a Krylov subspace type method, such as parallel GMRES, can be chosen from the KSP object at runtime. Second, the preconditioning operation defined in (10) is implemented by a PETSc user-defined preconditioner named PCSHELL, as described in Algorithm 2. Third, the RAS preconditioner, B^{-1}, as described in (12), is a built-in PC object of PETSc named PCASM.

We remark that the multiplication of B^{-1} with a given vector, $v = B^{-1} w$, consists of three steps. On each subdomain, (i) collects the data from the subdomain and the neighboring subdomains, $w_i = R_i^{\delta} w$; (ii) solves the subdomain problem,

$$B_i v_i = w_i; \tag{13}$$

(iii) computes the sum, $v = \sum_{i=1}^{Ns} (R_i^0)^T v_i$ by ignoring the computed value in the overlapping region. Furthermore, to save computational cost and memory use in practice, local subdomain problems (13) are solved by an inexact solver incomplete LU (ILU). The ILU solver performs an incomplete LU decomposition and then performs a forward and backward substitution using the incomplete factors.

Algorithm 2. Preconditioning operation: $B_d z = y$

Input: $y, z_2 = B^{-1} p$, and $\mu_2 = u^* z_2$
Output: z
1 Apply the preconditioning: $z_1 = B^{-1} y$
2 Compute the vector dot product: $\mu_1 = u^* z_1$
3 Compute $\eta = -\mu_1/\mu_2$
4 Compute $z = z_1 + \eta z_2$

- *Parallel basic linear algebraic operations*. PETSc provides a numerous of Vec and Mat commends for performing basic vector and matrix operations in parallel. Some of these are used in the implementation of the ASPJD eigensolver, as follows: (i) Matrix-by-vector product (*MatMult*): $W_i = A_i V$ (line 3), $r = \mathcal{A}(\theta) u$ (line 7), and $w_i = A_i v$ (line 10) in Algorithm 1. (ii) Dot product (*VecDot* or *VecMDot* for multiple vectors cases): $M_i = V^* W_i$ (line 3), $v^* W_i$, $V^* w_i$, and $v^* w_i$ (line 10) in Algorithm 1; $\mu_1 = u^* z_1$ (line 2 of Algorithm 2). (iii) Vector updates (*VecWAXPY* or *VecMAXPY* for the multiple vectors cases): $u = Vs$ and $p = \mathcal{A}'(\theta) u$ after performing the matrix-vector products $A_i u$ in line 7 of Algorithm 1; $z = z_1 + \eta z_2$ (line 4 of Algorithm 2). In addition, *IPOrthogonalize* is used in SLEPc to perform the process of orthogonalization in line 9 of Algorithm 1.

5. Numerical results

To validate our parallel ASPJD eigensolver and to evaluate the parallel performance of our proposed algorithm, we consider the test cases listed in Table 2. These cases arise in the discretization of two QD models described in Section 2 by using different grid sizes. Physical parameters used in the numerical simulations are summarized in Table 3.

The ASPJD eigensolver has been implemented by using C language and several components in PETSc and SLEPc. The Intel C compiler version 9.1 with `-O2` optimization flag is used to compile the codes. The numerical experiments were performed on the Vger cluster at the National Central University in Taiwan.

The Vger consists of 108 computing nodes; each node has two Intel Xeon 3.0 GHz Dual-Core processors with 4 GB of memory. The computing nodes are interconnected by InfiniBand switches of 2 GB/s bandwidth with a peak performance rate of 5184 Gflop/s. All computations were done in double precision complex arithmetic. The execution timings are reported in seconds and exclude the time spent on coefficient matrix loading and construction of the working vectors.

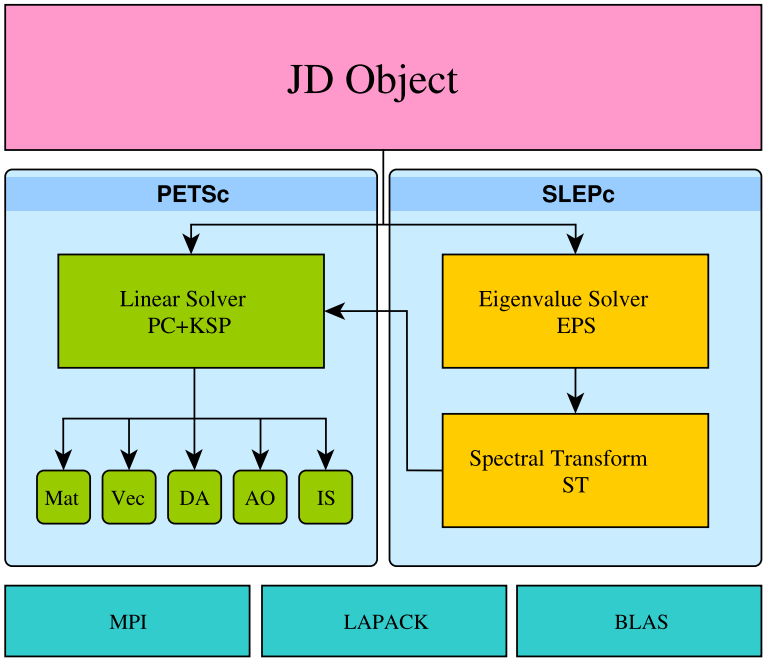


Fig. 2. The organization of PETSc, SLEPc, and the ASPJD Eigensolver.

We declare the *JD iterations*, i.e. the **while**-loop starting from line 4 of Algorithm 1, converge to an eigenpair, if absolute or relative residual of Eq. (1) is less than 10^{-10}. $V_{ini} = (1, 1, \ldots, 1)^T$ is set to be in the initial search space. The maximum number of ASPJD restarts is set at 30.

A *left* restricted additive Schwarz preconditioned GMRES with a zero initial guess is used for solving the correction Eq. (8). The *GMRES iterations* are referred to the iterations used by GMRES to solve the correction equation. The subdomain problems (13) are solved by an incomplete LU decomposition. For simplicity of implementation, the actual partitioning and overlapping size are determined internally by PETSc. Except when explicitly stated otherwise, the target eigenvalue is the smallest positive eigenvalue.

5.1. Parallel code validation

To check the correctness of our parallel implementation, we compute the target positive eigenvalues of Problem Q2 and Problem C2 using 64 processors. In these cases, the target eigenvalues for the pyramidal QD and cylindrical QD model are within the confinement potential intervals $\mathcal{I}_1 \equiv [0, 0.77]$ and $\mathcal{I}_2 \equiv [0, 0.35]$, respectively. For both cases, we use a 10 fixed-step block Jacobi preconditioned GMRES and ILU with zero level fill-ins is selected to be a subdomain solver. As presented in Table 4, our computed results are compared with the reference results obtained by a sequential Fortran JD code [19,49]. The table shows that the relative differences in 2-norm are all less than 10^{-10}. Note that the sixth positive eigenvalue of Problem Q2 and the fifth positive eigenvalue of Problem C2 listed in the table are beyond the intervals $\mathcal{I}_1$ and $\mathcal{I}_2$.

Table 2
Test problems and their statistics. The notations *L*,*M*, and *N* represent for the number of grid points in the *x*-, *y*-, and *z*-coordinates, respectively. The notations ρ and ζ represent the number of grid points in the radial and height coordinates, respectively.

Problem	PEP type	QD model	Grid	$(L,M,N)/(\rho,s\zeta)$	Matrix size
Q1	Quintic	Pyramidal	Uniform	(64,64,48)	186,543
Q2	Quintic	Pyramidal	Uniform	(128,128,96)	1,532,255
Q3	Quintic	Pyramidal	Uniform	(352,352,264)	32,401,863
C1	Cubic	Cylindrical	Non-uniform	(535,190)	101,650
C2	Cubic	Cylindrical	Non-uniform	(1720,585)	1,006,200
C3	Cubic	Cylindrical	Uniform	(1742,579)	1,008,618

Table 3
Physical parameters used the for non-parabolic effective mass models described in (3). These parameters are taken from [19,49].

QD model	Hetero-structure	c_ℓ	g_ℓ	δ_ℓ	P_ℓ
Pyramidal	Dot ($\ell = 1$)	0.000	0.420	0.48	0.8503
	Matrix ($\ell = 2$)	0.770	1.520	0.34	0.8878
Cylindrical	Dot ($\ell = 1$)	0.000	0.235	0.81	0.2875
	Matrix ($\ell = 2$)	0.350	1.590	0.80	0.1993

Table 4
Comparison of a few computed and reference eigenvalues of Problems Q2 and C2. Note that only the real parts of the eigenvalues are shown in this table and that the imaginary parts of the eigenvalues are all less than 1.0e−12.

		Computed eigenvalues	Reference values	Relative differences
Problem Q2	λ_1	0.416470647279179	0.416470647281298	2.11880e−12
	λ_2	0.599267979646491	0.599267979636767	9.72411e−12
	λ_3	0.599267979645261	0.599267979636393	8.86801e−12
	λ_4	0.717907643881655	0.717907643900873	1.92171e−11
	λ_5	0.729699797944921	0.729699797946620	1.69830e−12
	λ_6	0.792492416777499	0.792492416781907	4.40780e−12
Problem C2	λ_1	0.087344809377190	0.087344809345140	3.20501e−11
	λ_2	0.150294727564833	0.150294727561288	3.54500e−12
	λ_3	0.245994432693207	0.245994432699191	5.98435e−12
	λ_4	0.330502438790559	0.330502438793776	3.21687e−12
	λ_5	0.350930026609811	0.350930026592148	1.76629e−11

Table 5
JD iterations and execution time in seconds (shown within parentheses) for Problems Q2 and C2. Various combinations of RAS(δ), ILU(μ), and GMRES(s) are applied. The number of processors is 64. The mark * indicates the average number of GMRES iterations while setting $gtol$=10^{-3}. The mark † indicates that the eigensolver converges to an undesired eigenpair.

GMRES(s)	ILU(μ)	RAS(0)	RAS(1)	RAS(2)
(a) *Problem Q2*				
	ILU(0)	12 (31.9); 87*	12 (45.3); 86*	13 (63.9); 80*
GMRES with	ILU(1)	12 (34.3); 86*	14 (62.3); 84*	12 (67.3); 74*
$gtol = 10^{-3}$	ILU(2)	11 (35.8); 88*	11 (54.6); 76*	19 (142.6); 77*
	ILU(3)	12 (42.1); 86*	12 (77.5); 72*	15 (159.3); 76*
GMRES(10)	ILU(0)	21 (7.6)	25 (16.9)	28 (26.6)
	ILU(1)	20 (7.8)	28 (21.8)	16 (16.7)
	ILU(2)	19 (8.4)	19 (17.5)	13 (17.5)
	ILU(3)	19 (9.9)	18 (23.4)	13 (25.8)
GMRES(15)	ILU(0)	22 (11.7)	13 (11.1)	13 (15.3)
	ILU(1)	26 (14.7)	12 (11.3)	14 (19.4)
	ILU(2)	21 (12.6)	12 (14.4)	16 (28.6)
	ILU(3)	24 (18.0)	16 (26.7)	17 (44.9)
GMRES(20)	ILU(0)	19 (12.3)	15 (17.0)	13 (19.2)
	ILU(1)	15 (10.2)	14 (16.8)	13† (22.7)
	ILU(2)	15 (11.9)	12 (18.2)	15 (33.9)
	ILU(3)	14 (12.6)	12 (24.9)	14 (46.1)
(b) *Problem C2*				
	ILU(0)	11 (15.0); 100*	11 (17.3); 100*	11 (18.9); 100*
GMRES with	ILU(1)	12 (17.4); 100*	12 (20.6); 100*	16 (29.8); 100*
$gtol = 10^{-3}$	ILU(2)	11 (18.2); 100*	14 (27.7); 100*	12 (26.0); 100*
	ILU(3)	11 (18.6); 100*	15 (30.4); 100*	15 (36.6); 100*
GMRES(10)	ILU(0)	67 (11.1)	58 (12.7)	57 (18.2)
	ILU(1)	53 (9.3)	41 (9.1)	38 (9.4)
	ILU(2)	48 (9.0)	36 (8.6)	30 (8.1)
	ILU(3)	44 (9.1)	29 (7.9)	27 (8.0)
GMRES(15)	ILU(0)	44 (9.6)	41 (11.5)	41 (12.5)
	ILU(1)	33 (8.0)	27 (8.1)	26 (8.5)
	ILU(2)	30 (7.8)	23 (7.2)	22 (7.6)
	ILU(3)	28 (8.0)	22 (7.7)	19 (7.4)
GMRES(20)	ILU(0)	31 (9.9)	27 (9.9)	27 (10.7)
	ILU(1)	25 (7.9)	21 (8.0)	20 (8.3)
	ILU(2)	24 (8.0)	19 (7.7)	18 (8.0)
	ILU(3)	25 (9.3)	17 (7.6)	27 (13.9)

The mark * indicates average number of GMRES iterations while setting $gtol = 10^{-3}$.
The mark † indicates that the eigensolver converges to an undesired eigenpair.

Table 6
Effects of GMRES(s) on JD iterations and timing (shown within parentheses) for Problem Q1 and Q2. The shortest execution time for each number of processors (np) is shown in boldface. Note that Problems Q1 (186,543) is too small to be solved efficiently using more than 64 processors. Problem Q2 (1,532,255) is too large to be solved by using 4 processors due to memory constraints.

	Problem Q1				Problem Q2			
np	$s = 5$	$s = 10$	$s = 15$	$s = 20$	$s = 5$	$s = 10$	$s = 15$	$s = 20$
4	27 (13.9)	**16** (**10.2**)	16 (13.9)	13 (14.7)	–	–	–	–
8	28 (6.9)	**16** (**4.6**)	21 (8.6)	17 (8.9)	28 (89.9)	23 (96.5)	18 (98.4)	**11** (**75.5**)
16	27 (3.3)	**16** (**2.0**)	13 (2.2)	14 (3.1)	30 (38.1)	**22** (**35.7**)	19 (40.5)	14 (37.6)
32	22 (1.4)	**17** (**1.1**)	15 (1.1)	21 (5.4)	33 (19.9)	**21** (**16.4**)	22 (23.4)	13 (17.1)
64	24 (1.1)	22 (0.9)	**17** (**0.7**)	21 (1.2)	38 (11.2)	**21** (**7.6**)	22 (11.7)	19 (12.3)
128	–	–	–	–	41 (5.9)	**21** (**3.8**)	20 (4.5)	17 (6.5)
256	–	–	–	–	41 (3.6)	**21** (**2.1**)	20 (2.5)	17 (2.7)
320	–	–	–	–	41 (3.2)	**21** (**1.7**)	21 (2.2)	17 (2.0)

5.2. Algorithmic parameter tuning for the parallel ASPJD solver

We study how the following factors affect the number of JD iterations and the overall execution time:

1. The overlapping size: Let RAS(δ) denote the restricted additive Schwarz preconditioner (12) with overlapping size $\delta = 0, 1, 2, 3$.
2. The quality of subdomain solve: Let μ be the level of fill-ins in ILU(μ), which is used to solve the subdomain problems (13). We choose $\mu = 0, 1, 2$, or 3.
3. The stopping strategies for GMRES: Let s be the iteration number performed by GMRES(s) while solving the correction equation in line 8 of Algorithm 1. We choose $s = 10, 15$, or 20.

Numerical results of JD iterations and execution time are summarized in terms of RAS(δ), ILU(μ), and GMRES(s) in Table 5. Here, we compute only the first positive eigenpair. In the tables, we also include results obtained by setting the GMRES iteration tolerance for relative residual reduction $gtol = 10^{-3}$. Additionally, results of JD iterations and execution time in terms of GMRES(s) with respect to the number of processors are summarized in Tables 6 and 7. We highlight the observations regarding parameter tuning as follows:

5.2.1. To avoid over-solving in the correction equations

One advantage of the ASPJD algorithm is that the convergence of the algorithm can be achieved by solving the correction equations with modest accuracy and usually without downgrading overall performance. Such behavior has been observed for linear eigenvalue problems [35]. We present similar observations for high degree PEPs here.

Table 5 suggests that the correction equations are solved to a precision beyond what is needed for GMRES with $gtol = 10^{-3}$, which is commonly used in an inner–outer iterative algorithm such as an inexact Newton method. Although in this case the ASPJD algorithm takes fewer outer iterations in general, the corresponding GMRES iterations are excessively large. Consequently, all timing results are much slower than those observed using fixed-step GMRES.

Note that the actual relative residual reductions corresponding to GMRES(10), GMRES(15), and GMRES(20) for Problem Q2 (C2) are roughly equal to 0.2870, 0.2620, and 0.1912 (0.3254, 0.2715, and 0.2306), respectively.

Table 7
Effects of GMRES(s) on JD iterations and timing (shown within parentheses) for Problems C1 and C2. The shortest execution time for each number of processors (np) is shown in boldface. Note that Problems C1 (101,650) is too small to be solved efficiently by using more than 64 processors. Problem C2 (1,006,200) is too large to be solved by using 4 processors due to memory constraints.

	Problem C1				Problem C2			
np	$s = 5$	$s = 10$	$s = 15$	$s = 20$	$s = 5$	$s = 10$	$s = 15$	$s = 20$
4	**21** (**4.9**)	16 (5.3)	19 (8.9)	15 (9.1)	–	–	–	–
8	35 (4.0)	**17** (**2.4**)	18 (3.6)	19 (5.1)	50 (79.2)	25 (54.5)	**19** (**54.5**)	21 (79.3)
16	37 (2.0)	22 (1.5)	**15** (**1.3**)	18 (2.0)	51 (37.3)	**24** (**24.3**)	19 (25.7)	20 (35.4)
32	40 (1.3)	23 (0.8)	**17** (**0.6**)	22 (1.1)	55 (19.6)	29 (14.6)	**21** (**13.8**)	18 (15.2)
64	47 (1.0)	22 (0.5)	21 (0.5)	**17** (**0.4**)	74 (14.2)	36 (8.6)	**23** (**7.2**)	19 (7.7)
128	–	–	–	–	102 (10.2)	48 (6.2)	29 (5.0)	**24** (**4.9**)
192	–	–	–	–	119 (9.8)	52 (5.3)	32 (4.0)	**25** (**3.7**)

5.2.2. Recommended parameter combinations of RAS(δ) and ILU(μ)

For Problem Q2, as shown in part(a) of Table 5, we find that there is no obvious benefit from increasing overlapping size for RAS or using more levels of fill-ins in ILU.

The situation for Problem C2 as shown in part (b) of Table 5 is slightly different. An overlapping RAS with a more accurate inexact subdomain solve enables us to reduce both the number of JD iterations and execution time.

Based on the numerical results, we used the following parameters for the rest of the experiments: (i) RAS(0) and ILU(0) for quintic pyramidal QD problems, and (ii) RAS(1) and ILU(2) for cubic cylindrical QD problems.

5.2.3. Effects of GMRES(s) on JD iterations and timing

As shown in Table 6 for Problem Q1 and Q2, we observed that the choice of GMRES(10) results in the best timing results in almost all cases. Two exceptions are $np = 64$ for Problem Q1 and $np = 8$ for Problem Q2. Furthermore, the number of JD iterations for each np corresponding to the least execution time is nearly *independent* of np and is mildly dependant of problem size.

For Problems C1 and C2, Table 7 shows different convergence behaviors. There is no single s achieves the best JD iterations and timing results overall. However, roughly speaking, more GMRES iterations are needed as np increase to achieve the best timing results. Unlike Problems Q1 and Q2, the number of JD iterations for Problem C1 and C2 corresponding to the least execution time are more sensitive to np. Learnt from numerical experiences for solving linear systems, the RAS preconditioner for these two problems may need to add a coarse space when the number of processors increases. Without a coarse space, the increase of overlap is required to improve global communications, and smaller tolerance for the GMRES and more exact local problems are necessary.

5.3. Parallel performance

To evaluate the parallel performance of the eigensolvers, we consider *speedup* and *parallel efficiency*. Speedup and parallel efficiency are defined as

$$\frac{T_{np_1}}{T_{np_2}} \text{ and } \frac{np_1 \times T_{np_1}}{np_2 \times T_{np_2}},$$

respectively. Speedup indicates how much faster a parallel program with np_2 is than a corresponding parallel program with np_1. Parallel efficiency shows how much we can gain from parallelization and the percentage of the execution time spent on communication and synchronization. Fig. 3 presents the speedup and parallel efficiency plots for Problem Q2 and C2. We choose the results obtained by using $np_1 = 16$ ($np_1 = 4$) for Problem Q2 (C2) as the reference timings. All data presented in the figures is based on the best timing for each np excerpted from Tables 6 and 7.

Observing from Fig. 3, we highlight the following.

- For Problem Q2, the ASPJD eigensolver exhibits very impressive parallel performance: superlinear speedup and over 100% parallel efficiency up to 320 processors.
- For Problem C2, the eigensolver achieves nearly linear speedup for $np_2 < 64$. The performance of the eigensolver is degraded somewhat, but it still maintains 70% parallel efficiency up to $np_2 = 192$.

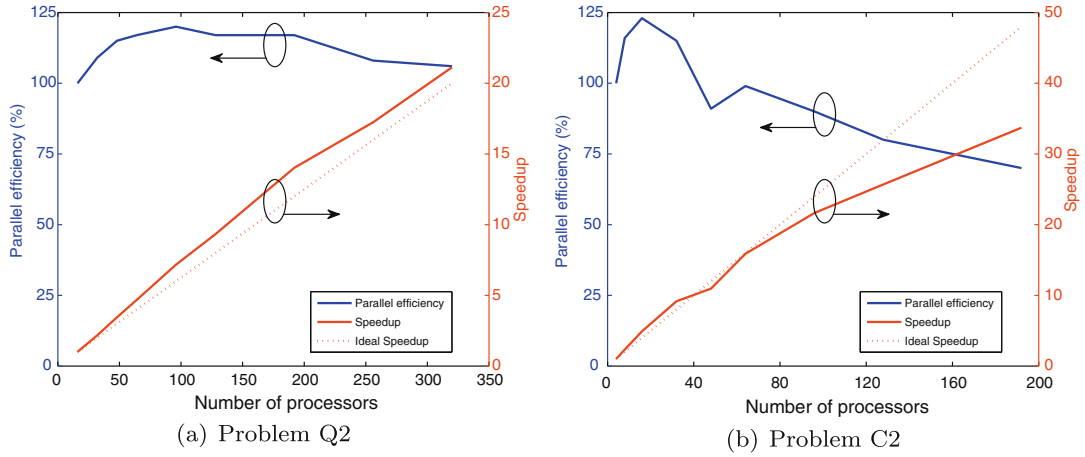


Fig. 3. Parallel efficiency and speedup for Problem Q2 and C2. Algorithmic parameters used: (a) RAS(0) and ILU(0), (b) RAS(1) and ILU(2).

Table 8
The robustness and scalability test of the ASPJD algorithm for finding a variety of target eigenvalues. Problem Q2 is considered. The notations μ and λ stand for target value and computed eigenvalue, respectively.

μ	λ	$np = 8$ JD ites (Time)	$np = 256$ JD ites (Time)	Parallel efficiency (%)
−0.1	−0.4199999999844743	54 (250.6)	56 (7.7)	101
0.1	0.4164706473388473	30 (136.9)	28 (3.7)	117
0.4	0.4164706473196381	23 (96.0)	21 (2.2)	138
0.6	0.5992679796328200	31 (138.1)	42 (5.1)	84
0.7	0.7179076438971341	47 (205.5)	52 (6.7)	95
0.75	0.7296997979519239	54 (226.6)	77 (9.5)	75
0.8	0.8064207692754761	72 (320.6)	99 (13.2)	76

Additionally, to verify the robustness and scalability of the ASPJD algorithm numerically, we also include the results of the ASPJD eigensolver for finding eigenvalues other than the smallest positive eigenvalues. As shown in Table 8, the ASPJD algorithm is quite robust and scalable. In the numerical experiments, we employ GMRES(10) with RAS(0), where the subdomain problem is solved by ILU(0) for $np = 8$ and $np = 256$. Problem Q2 is considered. The ASPJD eigensolver successfully converges to the eigenvalues close to the given target values, μ and retains good parallel efficiency that range from 75% to 138%. Note that the ASPJD eigensolver take longer time (greater 200 and 6 s for $np = 8$ and $np = 256$, respectively) to solve the eigenvalue problems for $\mu = -0.1, 0.7, 0.75$, and 0.8. It is because the eigenvalues are clustered and the corresponding eigenvectors are more oscillatory for these μ's. Actually, these particular μ's are close or beyond the boundary of the confinement potential interval $[c_1, c_2] = [0, 0.77]$. The mathematical QD model is invalid if the energy levels (i.e. eigenvalues) are out of the interval.

5.4. Performance analysis

We further analyze parallel performance of the ASPJD eigensolver (Fig. 3) by answering the following questions: Why does the ASPJD eigensolver perform remarkably for Problem Q2? Why does the parallel efficiency of Problem C2 degrade as the number of processors increases? We believe such phenomena are caused by a mixture effect due to the preconditioning strategy, and the characteristics of the QD eigenvalue problem and the structure of its corresponding the coefficient matrices, as discussed below.

- The no-communication preconditioner with RAS(0) leads to superlinear scale-up for correction equation solve and overall execution time.

We begin the study by presenting a scaling analysis for each key component in the ASPJD algorithm in Table 9. Part (a) of the table shows that all components scale very well for Problem Q2, with the exception of a relatively inexpensive linearized projected eigenvalue problem solve. In particular, ASPJD achieves a superlinear time scaling behavior for the most expensive

Table 9
Timing and the corresponding percentage (shown within parentheses) breakdown of each key component in the ASPJD algorithm for Problems Q2 and C2. The components include (a) LPEP solve: linearized projected eigenvalue problem solve; (b) *CorrEq solve*: correction equation solve; (c) *Form* M_i: M_i calculation; (d) *Compute u,p,r*: the vectors *u,p*, and *r* calculation; and (e) *From* $\mathcal{A}(\theta)$: $\mathcal{A}(\theta)$ calculation.

Component	$np = 16$	$np = 32$	$np = 64$	$np = 128$	$np = 256$
(a) *Problem Q2*					
LPEP solve	0.3 (0.9%)	0.3 (1.7%)	0.3 (3.5%)	0.3 (7.0%)	0.3 (13.0%)
CorrEq solve	22.3 (62.5%)	10.3 (62.7%)	4.7 (60.8%)	2.1 (55.5%)	1.0 (46.6%)
Form M_i	7.1 (19.8%)	3.2 (19.2%)	1.5 (19.4%)	0.8 (21.5%)	0.5 (25.3%)
Compute u,p,r	2.3 (6.4%)	1.1 (6.4%)	0.5 (7.1%)	0.3 (8.3%)	0.2 (8.6%)
Orthogonalization	1.3 (3.5%)	0.3 (2.0%)	0.1 (1.8%)	0.1 (1.9%)	0.1 (2.5%)
Form $A(\theta)$	2.4 (6.7%)	1.3 (7.8%)	0.6 (7.3%)	0.2 (5.6%)	0.1 (3.9%)
Total time	35.7	16.5	7.7	3.8	2.2
(b) *Problem C2*					
LPEP solve	0.2 (0.6%)	0.1 (0.7%)	0.1 (1.7%)	0.1 (2.9%)	
CorrEq solve	17.7 (72.9%)	11.0 (81.4%)	5.9 (81.5%)	4.1 (84.1%)	
Form M_i	3.3 (13.5%)	1.2 (8.5%)	0.7 (9.3%)	0.4 (7.5%)	
Compute u,p,r	1.2 (4.7%)	0.4 (3.2%)	0.2 (3.0%)	0.1 (2.2%)	
Orthogonalization	0.7 (2.7%)	0.3 (2.0%)	0.1 (1.3%)	0.1 (1.6%)	
Form $A(\theta)$	1.3 (5.4%)	0.6 (4.1%)	0.2 (3.1%)	0.1 (1.7%)	
Total time	24.4	13.6	7.2	4.9	

component, i.e., correction equation solve. We believe that the cause of the superlinear speedup is interplay among the following factors: First, the preconditioner used for this particular problem, i.e., RAS(0), does not involve any data communications between processors. Second, the number of GMRES iterations is fixed to 10 and the number of JD iterations (as shown in Table 6) remains nearly the same with respect to the number of processors. Third, the CPU cache memories further accelerate the data movements.

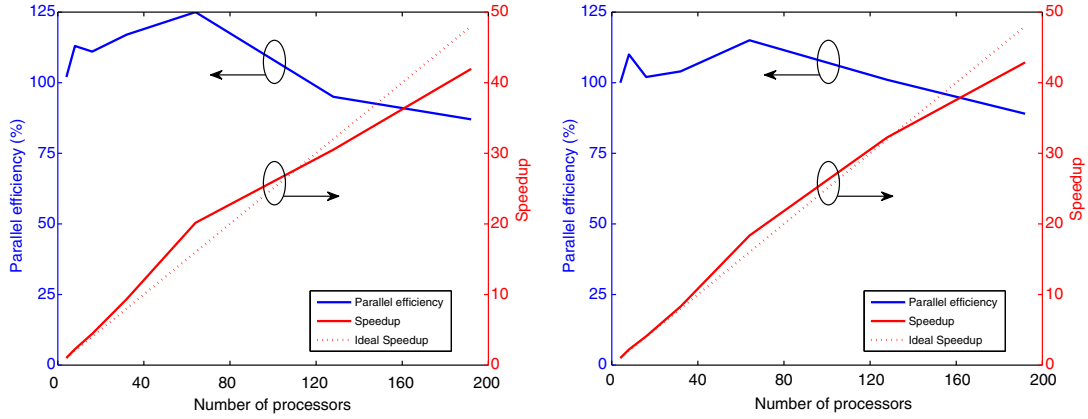
In contrast, Table 9(b) shows that ASPJD does not achieve superlinear scale-up in *CorrEq solve* for Problem C2, while scaling behaviors for all other components are similar to those for Problem Q2. Recall that, in order to keep the JD iteration under control, we use RAS(1) for preconditioning Problem C2. This is the price needed in order to use RAS with some size of overlapping, which at least minimally involves some sort of communications. Consequently, scalability for *CorrEq solve* is not as good as in Problem Q2. Overall efficiency for Problem C2 is thus degraded.

It should be noted that the QD eigenvalue problem has a special structure that the eigenvectors corresponding to the eigenvalues of interest are localized to the dot. That is, the components of the eigenvector corresponding to the matrix (outside of the QD) are mostly zero.

In our simulations, the ratio of the cuboid matrix to the pyramidal dot in the computational domain is roughly 35:1, compared to the ratio which is roughly 10:1 for the cylindrical QD problem. Consequently, we are able to decouple the original pyramidal QD eigenvalue problem into many subproblems using RAS(0) for the without penalty in terms of increased number of the JD iterations, as the cuboid matrix problem (Q2) has fewer non-zero elements relatively in the eigenvectors than the cylindrical matrix problem (C2). We believe this is at least partially responsible for the different behaviors between the Q2 and C2 problems illustrated in Table 5.

- The discretization based on non-uniform grid may lead to lower parallel efficiency.

As shown in Table 2, Problem Q2 is formulated by a (finite volumes) discretization with a *uniform* grid and Problem C2 is formulated by a (finite differences) discretization with a *non-uniform* grid. In a non-uniform grid, grid points around the hetero-structure are chosen with fine grid and other grid points are chosen with rough grid. Such non-uniform grids are essential for efficiently computing eigenpairs with higher accuracy [49]. However, the non-uniform grid also introduces more



(a) Problem C3 preconditioned by RAS(1)

(b) Problem C3 preconditioned by RAS(0)

Fig. 4. Parallel efficiency and speedup for Problem C3 formulated by a uniform grid. Preconditioning strategy used in (a) is RAS(1) and in (b) is RAS(0). Note that ILU(2) are applied in both cases.

Table 10
A list of JD iterations and time spent for computing each of the five smallest positive eigenvalues of Problem Q3 with matrix size 32,401,863.

λ_i	JD ites	Time
0.4162332868108254	27	122.6
0.5990929682408994	26	121.6
0.5990929681591901	24	112.6
0.7179672367710944	44	207.8
0.7295413375156179	27	129.7
Total JD ites and total time	148	694.3

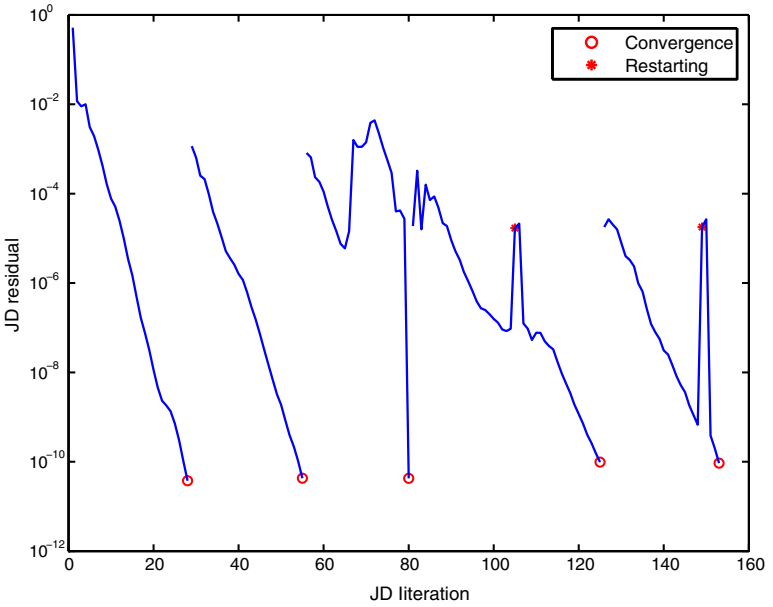


Fig. 5. JD convergence history for computing the five smallest positive eigenvalues of Problem Q3.

unbalanced entry magnitude in A_i's that can result in lower computational accuracy while solving the corresponding correction equation [8].

We conjecture that these coefficient matrices due to the non-uniform grids further affect the parallel efficiency of the ASPJD eigensolver. This conjecture is supported by the following numerical experiments.

To verify the above claims regarding how RAS(δ) and the matrix structures can affect parallel performance, we present performance plots of Problem C3 in Fig. 4. Both Problem C3 and Problem C2 are derived by modeling the same nano-structure and using the same discretization scheme. The only difference is that Problem C3 is formulated by a uniform grid, while Problem C2 is formulated by a non-uniform grid. Comparing Fig. 3(b) (associated with non-uniform grid, RAS(1), and ILU(2)) with Fig. 4(a) (associated with uniform grid, RAS(1), and ILU(2)), we can see that the parallel efficiency for Problem C3 (with uniform grid) has been improved. By further switching to the RAS(0) preconditioner, we can gain even better speedup results for Problem C3, as shown in Fig. 4(b).

5.5. A large-scale problem

To show applicability of our parallel eigensolver for large-scale PEPs, we report numerical results in Table 10 for Problem Q3 with $np = 272$. Problem Q3 is a very large problem with a dimension of more than 32 million. As we learned from previous numerical experience, an appropriate stopping strategy for GMRES depends mildly on problem size. Therefore, for this case, GMRES(20) is used for solving the correction equation and the other ASPJD parameters are set to be the same as in the previous test cases. For such large PEP, it requires only 694.3 s to compute all five target eigenpairs. The convergence history for computation of these five eigenpairs is shown in Fig. 5.

By way of reference, we replace the RAS preconditioner for the correction equation with a Jacobi preconditioner and compare them. As a result, the JD algorithm with the Jacobi preconditioner first finds λ_1 and λ_2, followed by λ_4, λ_3, and λ_5 and takes 175.8(42), 438.4(101), 1.5(1), 287.9(66), and 416.1(93) s (JD iterations) (1319.7 s and 303 JD iterations in total) to find all eigenvalues. Obviously, the ASPJD is much better than JD with the Jacobi preconditioner in terms of the total number of JD iterations and execution time. Although the eigenvector is smooth and is almost zero outside the dot, the Jacobi preconditioner seems to throw away too much information needed from neighboring girds, especially in the dot.

6. Conclusion

Aiming at the higher degree PEPs arising in nanoscale quantum dot simulations, we have proposed a new parallel Jacobi–Davidson approach. We have presented a method by which we can efficiently parallelize the GMRES-based correction equation solver by using restricted additive Schwarz preconditioning with incomplete LU as the subdomain solver. We have implemented the algorithm by using several components provided by the PETSc and SLEPc packages. The parallel codes have

been tested on a computer cluster, using up to 320 computing cores to investigate the parallel performance. The numerical experiments have demonstrated impressive speedup and parallel efficiency results for large-scale quintic and cubic PEPs.

The potential exists for further improvement of the algorithm and its implementation. First, better grid partitioning may result in faster ASPJD convergence, as suggested in [33]. Second, we believe an efficient parallel deflation scheme for successively computing eigenpairs may further enhance the parallel ASPJD eigensolver. On the other hand, we think it interesting to compare the performance of the ASPJD eigensolver with other existing parallel eigensolvers, especially for standard and generalized eigenvalue problems of the types that arise in QD simulation. It is worth pointing out that the extendability and flexibility of PETSc and SLEPc allow us to easily conduct future studies by using various preconditioning strategies, performing parallel linear system solve other than GMRES, and/or integrating new ideas into our codes.

The proposed parallel Jacobi–Davidson algorithm is not only useful for application in QD simulations, but possesses the potential to be a powerful computational tool for solving eigenvalue problems arising in other physical models with various governing equations. The efficient parallel eigensolver is also a promising candidate to fulfill the increasing computational demands due to larger computational domains and higher accuracy requirements in eigenvalue problems.

Acknowledgments

The authors appreciate the anonymous referees for their useful comments and suggestion. The authors are grateful to the Center for Scientific Computing at the National Central University for providing computing resources. This work is partially supported by the National Science Council, the Taida Institute of Mathematical Sciences, and the National Center for Theoretical Sciences in Taiwan.

References

[1] P. Arbenz, M. Bečka, R. Geus, U. Hetmaniuk, T. Mengotti, On a parallel multilevel preconditioned Maxwell eigensolver, Parallel Comput. 32 (2006) 157–165.
[2] S. Balay, K. Buschelman, W.D. Gropp, D. Kaushik, M.G. Knepley, L.C. McInnes, B.F. Smith, H. Zhang, PETSc Web page, 2009. <http://www.mcs.anl.gov/petsc>.
[3] G. Bastard, Wave Mechanics Applied to Semiconductor Heterostructures, John Wiley and Sons, New York, 1991.
[4] L. Bergamaschi, A. Martinez, G. Pini, F. Sartoretto, Eigenanalysis of finite element 3D flow models by parallel Jacobi–Davidson, Lect. Notes Comput. Sci. 2840 (2003) 565–569.
[5] L. Bergamaschi, G. Pini, F. Sartoretto, Computational experience with sequential and parallel, preconditioned Jacobi–Davidson for large, sparse symmetric matrices, J. Comput. Phys. 188 (2003) 318–331.
[6] M.M. Betcke, H. Voss, Stationary Schrödinger equations governing electronic states of quantum dots in the presence of spin–orbit splitting, Appl. Math. 52 (2007) 267–284.
[7] M.M. Betcke, H. Voss, Numerical simulation of electronic properties of coupled quantum dots on wetting layers, Nanotechnology 19 (2008) 165204.
[8] T. Betcke, Optimal scaling of generalized and polynomial eigenvalue problems, SIAM J. Matrix Anal. Appl. 30 (2008) 1320–1338.
[9] X.-C. Cai, M. Sarkis, A restricted additive Schwarz preconditioner for general sparse linear systems, SIAM J. Sci. Comput. 21 (1999) 792–797.
[10] S.L. Chuang, N. Peyghambarian, S. Koch, Physics of Optoelectronic Devices, Wiley, New York, 1996.
[11] E.A. de Andrada e Silva, G.C. La Rocca, F. Bassani, Spin–orbit splitting of electronic states in semiconductor asymmetric quantum wells, Phys. Rev. B 55 (1997) 16293.
[12] D. El-Moghraby, R.G. Johnson, P. Harrison, Calculating modes of quantum wire and dot systems using a finite differencing technique, Comput. Phys. Commun. 150 (2003) 235–246.
[13] K. Elssel, H. Voss, Reducing sparse nonlinear eigenproblems by automated multi-level substructuring, Adv. Eng. Software 39 (2008) 828–838.
[14] E. Frisch, M. Frisch, G.W. Trucks, Gaussian 03, Gaussian, Pittsburgh, PA, 2003.
[15] F. Gelbard, K.J. Malloy, Modeling quantum structures with the boundary element method, J. Comput. Phys. 172 (2001) 19–39.
[16] M.B. Van Gijzen, The parallel computation of the smallest eigenpair of an acoustic problem with damping, Int. J. Numer. Meth. Eng. 45 (1999) 765–777.
[17] V. Hernandez, J.E. Roman, V. Vidal, SLEPc: a scalable and flexible toolkit for the solution of eigenvalue problems, ACM Trans. Math. Software 31 (2005) 351–362.
[18] T.-M. Huang, W. Wang, C.-T. Lee, An efficiency study of polynomial eigenvalue problem solvers for quantum dot simulations, Technical Report, NCTS Preprints in Mathematics 2009-11-002, National Tsing Hua University, Hsinchu, Taiwan, 2009.
[19] T.-M. Hwang, W.-W. Lin, W.-C. Wang, W. Wang, Numerical simulation of three dimensional pyramid quantum dot, J. Comput. Phys. 196 (2004) 208–232.
[20] T.-M. Hwang, W. Wang, Energy states of vertically aligned quantum dot array with nonparabolic effective mass, Comput. Math. Appl. 49 (2005) 39–51.
[21] T.-M. Hwang, W.-C. Wang, W. Wang, Numerical schemes for three-dimensional irregular shape quantum dots over curvilinear coordinate systems, J. Comput. Phys. 226 (2007) 754–773.
[22] T.-M. Hwang, W.-H. Wang, W. Wang, Efficient numerical schemes for electronic states in coupled quantum dots, J. Nanosci. Nanotechnol. 8 (2008) 3695–3709.
[23] A.V. Knyazev, Toward the optimal preconditioned eigensolver: locally optimal block preconditioned conjugate gradient method, SIAM J. Sci. Comput. 23 (2001) 517–541.
[24] R.B. Lehoucq, J.A. Scott, An evaluation of software for computing eigenvalues of sparse nonsymmetric matrices, Preprint MCS- P, 1195, 1996.
[25] Y. Li, J.-L. Liu, O. Voskoboynikov, C.P. Lee, S.M. Sze, Electron energy level calculations for cylindrical narrow gap semiconductor quantum dot, Comput. Phys. Commun. 140 (2001) 399–404.
[26] Y. Li, O. Voskoboynikov, C.P. Lee, S.M. Sze, Computer simulation of electron energy levels for different shape InAs/GaAs semiconductor quantum dots, Comput. Phys. Commun. 141 (2001) 66–72.
[27] M. Markiewicz, H. Voss, Electronic states in three dimensional quantum dot/wetting layer structures, Lect. Notes Comput. Sci. 3980 (2006) 684–693.
[28] R.V.N. Melnik, K.N. Zotsenko, Finite element analysis of electronic states, Modell. Simul. Mater. Sci. Eng. 12 (2004) 465–477.
[29] M. Nool, A. van der Ploeg, Parallel Jacobi–Davidson for solving generalized eigenvalue problems, Lect. Notes Comput. Sci. 1573 (1999) 58–70.
[30] M. Nool, A. van der Ploeg, A parallel Jacobi–Davidson-type method for solving large generalized eigenvalue problems in magnetohydrodynamics, SIAM J. Sci. Comput. 22 (2000) 95–112.
[31] M. Parrinello, From silicon to RNA: the coming of age of ab initio molecular dynamics, Solid State Commun. 102 (1997) 107–120.
[32] C. Pryor, Eight-band calculations of strained InAs/GaAs quantum dots compared with one-, four-, and six-band approximations, Phys. Rev. B 57 (1998) 7190–7195.

[33] Y. Saad, Iterative Methods for Sparse Linear Systems, SIAM, Philadelphia, 2003.
[34] G.L.G. Sleijpen, A.G.L. Booten, D.R. Fokkema, H.A. van der Vorst, Jacobi–Davidson type methods for generalized eigenproblems and polynomial eigenproblems, BIT 36 (1996) 595–633.
[35] G.L.G. Sleijpen, H.A. van der Vorst, A Jacobi–Davidson iteration method for linear eigenvalue problems, SIAM Rev. 42 (2000) 267–293.
[36] B.F. Smith, P.E. Bjørstad, W. Gropp, Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations, Cambridge University Press, Cambridge, 1996.
[37] A. Stathopoulos, Nearly optimal preconditioned methods for Hermitian eigenproblems under limited memory. Part I: Seeking one eigenvalue, SIAM J. Sci. Comput. 29 (2007) 481–514.
[38] A. Stathopoulos, J.R. McCombs, Nearly optimal preconditioned methods for Hermitian eigenproblems under limited memory. Part II: Seeking many eigenvalues, SIAM J. Sci. Comput. 29 (2007) 2162–2188.
[39] F. Tisseur, Backward error and condition of polynomial eigenvalue problems, Linear Algebra Appl. 309 (2000) 339–361.
[40] S. Tomov, J. Langou, A. Canning, L.W. Wang, J. Dongarra, Comparison of nonlinear conjugate-gradient methods for computing the electronic properties of nanostructure architectures?, Lect Notes Comput. Sci. 3516 (2005) 317–325.
[41] S. Tomov, J. Langou, J. Dongarra, A. Canning, L. Wang, Conjugate-gradient eigenvalue solvers in computing electronic properties of nanostructure architectures, Int. J. Comput. Sci. Eng. 2 (2006) 205–212.
[42] A. Toselli, O.B. Widlund, Domain Decomposition Methods-Algorithms and Theory, Springer, Berlin, 2005.
[43] C. Vömel, S.Z. Tomov, O.A. Marques, A. Canning, L.W. Wang, J.J. Dongarra, State-of-the-art eigensolvers for electronic structure calculations of large scale nano-systems, J. Comput. Phys. 227 (2008) 7113–7124.
[44] C. Vömel, S.Z. Tomov, L.W. Wang, O.A. Marques, J.J. Dongarra, The use of bulk states to accelerate the band edge state calculation of a semiconductor quantum dot, J. Comput. Phys. 223 (2007) 774–782.
[45] H. Voss, Iterative projection methods for computing relevant energy states of a quantum dot, J. Comput. Phys. 217 (2006) 824–833.
[46] H. Voss, Numerical calculation of the electronic structure for three-dimensional quantum dots, Comput. Phys. Commun. 174 (2006) 441–446.
[47] H. Voss, A Jacobi–Davidson method for nonlinear and nonsymmetric eigenproblems, Comput. Struct. 85 (2007) 1284–1292.
[48] W. Wang, T.-M. Hwang, J.-C. Jang, A second-order finite volume scheme for three dimensional truncated pyramidal quantum dot, Comput. Phys. Commun. 174 (2006) 371–385.
[49] W. Wang, T.-M. Hwang, W.-W. Lin, J.-L. Liu, Numerical methods for semiconductor heterostructures with band nonparabolicity, J. Comput. Phys. 190 (2003) 141–158.