ParaSLRF: A High Performance Rational Filter Method for Solving Large Scale Eigenvalue Problems

Biyi Wang[a,c], Karl Meerbergen[b], Raf Vandebril[b], Hengbin An[c,d], Zeyao Mo[c,d]

[a] *Graduate School of China Academy of Engineering Physics, Beijing, 100088, China*
[b] *Department of Computer Science, KU Leuven, Celestijnenlaan 200A, Leuven, 3001, Belgium*
[c] *Institute of Applied Physics and Computational Mathematics, Beijing, 100094, China*
[d] *CAEP Software Center for High Performance Numerical Simulation, Beijing, 100088, China*

Abstract

In *Wang et al., A Shifted Laplace Rational Filter for Large-Scale Eigenvalue Problems*, the SLRF method was proposed to compute all eigenvalues of a symmetric definite generalized eigenvalue problem lying in an interval on the real positive axis. The current paper discusses a parallel implementation of this method, abbreviated as ParaSLRF. The parallelization consists of two levels: (1) on the highest level, the application of the rational filter to the various vectors is partitioned among groups of processors; (2) within each group, every linear system is solved in parallel.

In ParaSLRF, the linear systems are solved by iterative methods instead of direct ones, in contrast to other rational filter methods, such as, PFEAST. Because of the specific selection of poles in ParaSLRF, the computational cost of solving the associated linear systems for each pole, is almost the same. This intrinsically leads to a better load balance between each group of resources, and reduces waiting times of processes.

We show numerical experiments from finite element models of mechanical vibrations, and show a detailed parallel performance analysis. ParaSLRF shows the best parallel efficiency, compared to other rational filter methods based on quadrature rules for contour integration. To further improve performance, the converged eigenpairs are locked, and a good initial guess of iterative linear solver is proposed. These enhancements of ParaSLRF show good two-level strong scalability and excellent load balance in our experiments.

Keywords: Generalized eigenvalue problem, Rational filtering, Subspace iteration, Shifted Laplace, Parallel computing
2000 MSC: 65F15, 65N25, 65H17

1. Introduction

Consider the following symmetric definite generalized eigenvalue problem (GEP)

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{B}\mathbf{x}, \tag{1}$$

Email address: wangbiyi20@gscaep.ac.cn (Biyi Wang)

where $\mathbf{A} \in \mathbb{R}^{n\times n}$ and $\mathbf{B} \in \mathbb{R}^{n\times n}$ are large, sparse and symmetric positive definite matrices. The GEP (1) arises from, e.g., vibration analysis, quantum mechanics, electronic structure calculations, etc. [1]. For example, in vibration analysis, the matrix $\mathbf{A}$ is the stiffness matrix and $\mathbf{B}$ is the mass matrix. The tuple $(\lambda, \mathbf{x})$ is called an eigenpair of the GEP. Because of the positive definiteness of the matrices A and B, the eigenvalues are real and positive. We are interested in computing all eigenvalues in a specified interval $(0, \gamma]$ and their associated eigenvectors.

To achieve this goal, there is a class of methods based on contour integration, utilizing the resolvent operator along a closed contour, which encloses all the desired eigenvalues. This class of methods was proposed in 2003 by Sakurai and Sugiura [2, 3]. The approximation of the contour integral by a quadrature rule is a *rational filter function*. There are several quadrature rules that can be used, such as midpoint [4], Gauss-Legendre [5], Gauss-Chebyshev [4], and Zolotarev [6] quadrature rules. The resulted rational filter functions can amplify eigenvalues within the contour and suppress eigenvalues outside the contour. In addition to the rational filter functions obtained by contour integrals, Xi and Saad [4] and Wang, Meerbergen, Vandebril, An, and Mo [7] determine the rational filter function differently; they utilize complex conjugate poles and put the poles on lines through the origin with a particular slope.

The application of the rational filter function requires the solution of ℓ systems with multiple right-hand sides of the form

$$(\mathbf{A} - \sigma_j \mathbf{B})\mathbf{Y} = \mathbf{BV} \quad \text{for} \quad j = 1, \ldots, \ell, \tag{2}$$

where σ_j, $j = 1, \ldots, \ell$, are the poles. The choice of the poles is not only important for the quality of the filter function, but also has significant impact on the performance of the linear system solver. The computational cost of solving (2) with a direct method is almost independent of σ_j. When the problem is large, iterative methods may become preferable. In this case, the difficulty and cost for solving (2) strongly depends on the value of σ_j. This paper only considers the case where the linear systems are solved by iterative methods.

Matrices (1) originating from mechanical vibration problems have properties similar to matrices in the Helmholtz equation. In this paper, instead of using the poles based on various quadrature rules, we select the poles with the form of the shifted Laplace preconditioner for the Helmholtz equation [8, 9, 10], which was inspired by earlier work [11, 12, 13]. The main idea is to pick complex conjugate poles on the lines $\{y = x(1 \pm \alpha\imath), \alpha > 0, x \in \mathbb{R}^+\}$. We call the corresponding rational filter a shifted Laplace rational filter (SLRF) [7]. It makes the linear system complex, but easier to precondition and solve. A detailed analysis and discussion is given by Wang et al. [7].

To the best of our knowledge, no existing study has thoroughly investigated the parallel performance of rational filter methods when the corresponding linear systems are solved using iterative linear solvers. In this paper, we experimentally show and analyze the impact of different strategies on the performance of various parallel rational filter methods when computing many eigenpairs. Our results show that there is a severe work-load imbalance and huge communication waiting overhead for the rational filters based on the classical quadrature rules. The parallel shifted Laplace rational filter (ParaSLRF) performs best in

terms of both CPU time and computation efficiency in our experiments. To further reduce the computational cost, we use a locking strategy and reduce the number of vectors required to be filtered, when convergence is reached. In addition, we propose to use a scaled Ritz vector as initial guess to accelerate the convergence of the linear system (2). To reduce the computation cost further and achieve a better load-balancing, we fix the maximum number of linear iterations.

The outline of this paper is as follows. In Section 2, we review parallel rational filter methods for eigenvalue computations and briefly discuss our strategy of selecting the poles and weights for the rational filter. We compare the computational cost between ParaSLRF and the rational filters based on various quadrature rules. Additionally, we illustrate the data distribution patterns and the computation flow within the parallel implementation framework. In Section 3, we present the numerical results of the ParaSLRF and compare its computational efficiency with that of the aforementioned filters based on quadrature rules. In Section 4, we adopt two strategies to improve the performance further and represent strong scalability of the enhanced version. We end this paper with the main conclusions in Section 5.

The following notation is used throughout this paper: $\mathbf{A} \in \mathbb{R}^{n\times n}$ and $\mathbf{B} \in \mathbb{R}^{n\times n}$ represent $n \times n$ large, sparse and symmetric positive definite matrices. The $\mathbf{B}$-inner product of two vectors $\mathbf{u}$, $\mathbf{v} \in \mathbb{R}^n$ is defined by $\langle \mathbf{u}, \mathbf{v}\rangle_{\mathbf{B}} = \mathbf{v}^\top \mathbf{B}\mathbf{u}$. The induced norm is $\|\mathbf{v}\|_{\mathbf{B}} = \sqrt{\langle \mathbf{v}, \mathbf{v}\rangle_{\mathbf{B}}}$. We define N as the total number of poles σ (in the upper plane), with corresponding weights denoted by w.

2. Parallel rational filter method

The core of the rational filter method is the rational function

$$\Phi(\lambda) = \sum_{j=1}^{\ell} \frac{w_j}{\lambda - \sigma_j}, \tag{3}$$

where σ_j is the j-th pole, and w_j is the corresponding weight, for $j = 1, 2, \ldots, \ell$. For a closed contour $\Gamma \subset \mathbb{C}$ which encloses all the desired eigenvalues, the rational function (3) can amplify the function value at the desired eigenvalues λ_i inside the contour Γ, and decrease the function values at the eigenvalues outside the contour Γ. The name *rational filter* arises from the filter properties.

To apply the filter it is necessary to solve ℓ linear systems of the form (2). Since $\mathbf{A}$ and $\mathbf{B}$ are real symmetric and the domain of interest is an interval on the real axis, we choose the ℓ nodes as complex conjugate points in the complex plane. Let $\ell = 2N$, then, with a proper ordering, we have $\sigma_{N+j} = \overline{\sigma_j}$ and $w_{N+j} = \overline{w_j}$, $j = 1, \ldots, N$. As a result, N complex valued linear systems have to be solved when the filter is applied to a real vector. The computation can be done in parallel.

The main idea is to distribute the required linear systems to multiple computing resources and solve them in parallel, then, use summation and reduction operations. We refer to each rational filtering step as an outer iteration, and the linear solves within each filtering step

as inner iterations. For brevity, we assume the np processes and N poles partitioned in n_{part} groups. We create parallel sub-communicators $\{G_j\}_{j=1}^{n_{part}}$ with $N_{sub} = N/n_{part}$ assigned poles and $np_{sub} = np/n_{part}$ processes. A framework of the parallel rational filter method for computing eigenpairs of the matrix pencil (1) is given in Algorithm 1, with details below. The algorithm aims to compute NEV eigenvalues in $(0, \gamma]$ and associated eigenvectors.

Algorithm 1 Parallel rational filter method for GEP (1)

Input: $\mathbf{A} \in \mathbb{R}^{n\times n}$, $\mathbf{B} \in \mathbb{R}^{n\times n}$, interval $(0, \gamma]$, NEV wanted eigenvalues, L columns of the starting vectors $\mathbf{V}_1 \in \mathbb{R}^{n\times L}$ where $L \geq$ NEV, poles $\sigma_1, \ldots, \sigma_N \in \mathbb{C}$ and corresponding weights $w_1, \ldots, w_N \in \mathbb{C}$, np number of processes, n_{part} number of partitions.

Output: converged eigenpairs $[\Theta, \mathbf{X}]$.

1: /* Initialization phase. */
2: Load matrix pencil $\mathbf{A}, \mathbf{B}$ and generate random starting vectors $\mathbf{V}_1$ in a global communicator G_0.
3: Split the global communicator G_0 into n_{part} sub-communicators $G_j, j = 1, 2, \ldots, n_{part}$, where the number of assigned processes and poles are $np_{sub} = np/n_{part}$ and $N_{sub} = N/n_{part}$, respectively. Scatter and redistribute $\mathbf{A}, \mathbf{B}$ and $\mathbf{V}_1$ from the global communicator to each sub-communicator.
4: **for** $k = 1, 2, \ldots$ until all of the wanted eigenpairs converged **do**
5: /* Parallel over the n_{part} partitions. */
6: **for** $j = 1, 2, \ldots, n_{part}$ **in parallel do**
7: **for** $i = 1, 2, \ldots, N_{sub}$ **do**
8: Set $\sigma_{j,i} = \sigma_{(j-1)N_{sub}+i}$ and $w_{j,i} = w_{(j-1)N_{sub}+i}$.
9: Solve $\mathbf{Y}_{j,i} = (\mathbf{A} - \sigma_{j,i}\mathbf{B})^{-1}\mathbf{B}\mathbf{V}_k$ in sub-communicator G_j **in parallel**.
10: **end for**
11: **end for**
12: Concatenate $\mathbf{Y}_j = [\mathbf{Y}_{j,1}, \mathbf{Y}_{j,2}, \ldots, \mathbf{Y}_{j,N_{sub}}]$ in sub-communicator G_j, **for** $j = 1, \ldots, n_{part}$ **in parallel**.
13: Scatter $\mathbf{Y}_j$ from each sub-communicator to the global communicator and then perform a sum-reduce operation $\mathbf{U} = \sum_{j=1}^{n_{part}} \sum_{i=1}^{N_{sub}} 2\,\mathrm{Re}(w_{j,i}\mathbf{Y}_{j,i})$.
14: Solve the projected eigenvalue problem $(\mathbf{U}^\top\mathbf{A}\mathbf{U})\mathbf{s}_m = \theta_m(\mathbf{U}^\top\mathbf{B}\mathbf{U})\mathbf{s}_m$ and let $\mathbf{v}_m = \mathbf{U}\mathbf{s}_m$, $m = 1, 2, \ldots, L$, with θ_m sorted in ascending order.
15: Check the convergence of the approximated eigenpairs $(\theta_m, \mathbf{v}_m), m = 1, 2, \ldots, L$. If converged, append θ_m to Θ, and duplicate $\mathbf{v}_m$ to $\mathbf{X}$, respectively.
16: Let $\mathbf{V}_{k+1} = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_L]$. Scatter $\mathbf{V}_{k+1}$ to each sub-communicator again.
17: **end for**

On Lines 2 to 3, the initialization is executed. The matrices $\mathbf{A}, \mathbf{B}, \mathbf{V}_1$ are duplicated and redistributed across each sub-communicator G_j. On Lines 6 to 11, the computing tasks are divided into n_{part} parts and the computation is performed in parallel, in multiple sub-communicators. On Line 13, the matrix $\mathbf{U}$ calculated by the sum-reduce operation represents the vectors obtained by applying the rational filtering on vectors $\mathbf{V}_k$. On Line 14, the projected pencil consisting of matrices $\hat{\mathbf{A}} = \mathbf{U}^\top\mathbf{A}\mathbf{U}$ and $\hat{\mathbf{B}} = \mathbf{U}^\top\mathbf{B}\mathbf{U}$ is formed. The

approximated eigenpairs $(\theta_m, \mathbf{v}_m)$ are extracted from the subspace projection on $\mathbf{A}$ and $\mathbf{B}$, by using the Rayleigh-Ritz method. On Line 15, the convergence of the approximated pairs $(\theta_m, \mathbf{v}_m)$ is examined. On Line 16, the matrix $\mathbf{V}_{k+1}$ is assembled and then scattered to each sub-communicator for the next rational filtering iteration. As described in Algorithm 1, we can use a two-level parallelism:

- Level-1: solve the linear systems with multiple right-hand terms independently, over n_{part} sub-communicators G_j, $j = 1, \ldots, n_{part}$;
- Level-2: solve each linear system in parallel within sub-communicator G_j.

2.1. Data distribution and computation flow

To achieve two-level parallelism , we need to manage the data properly and design a suitable computation flow. Assume we have np processes, N poles in total. We partition the np processes into n_{part} groups. (For the ease of presentation, we assume here that np is a multiple of n_{part}.) By default, np processes are organized as a 1D vector to represent the global communicator `MPI_COMM_WORLD`, denoted by G_0. The np processes are reorganized into a 2D grid of size $n_{part} \times (np/n_{part})$. Each column $P(:, j)$, where $j = 1, \ldots, n_{part}$, represents a group of resources. We associate it with a sub-communicator G_j consisting of $np_{sub} = (np/n_{part})$ processes, see Figure 1. For each column j, we assign $N_{sub} = N/n_{part}$ poles.

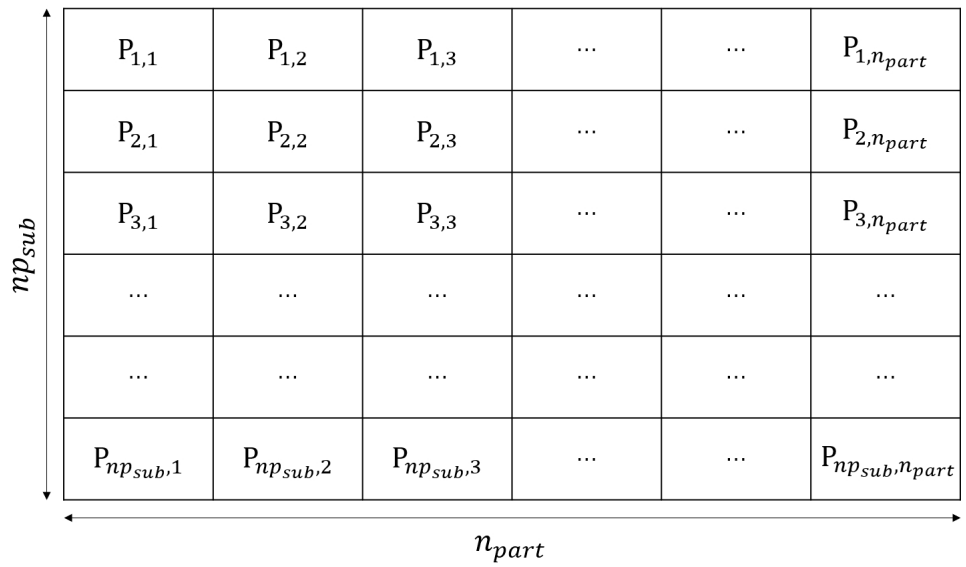

Figure 1: The 2D representation of np processes.

First we load the matrix pencil to the global communicator G_0, and each process stores some consecutive matrix rows. In G_0, we generate L random vectors $\mathbf{V}_1$ with the same row distribution as the matrix pencil. Then we scatter the matrix pencil and vectors forward to each sub-communicator G_j. We follow the same data storage and manipulation as in

SLEPc [14]. To illustrate Algorithm 1 more clearly, Figure 2 shows the data distribution and detailed operations at iteration step k.

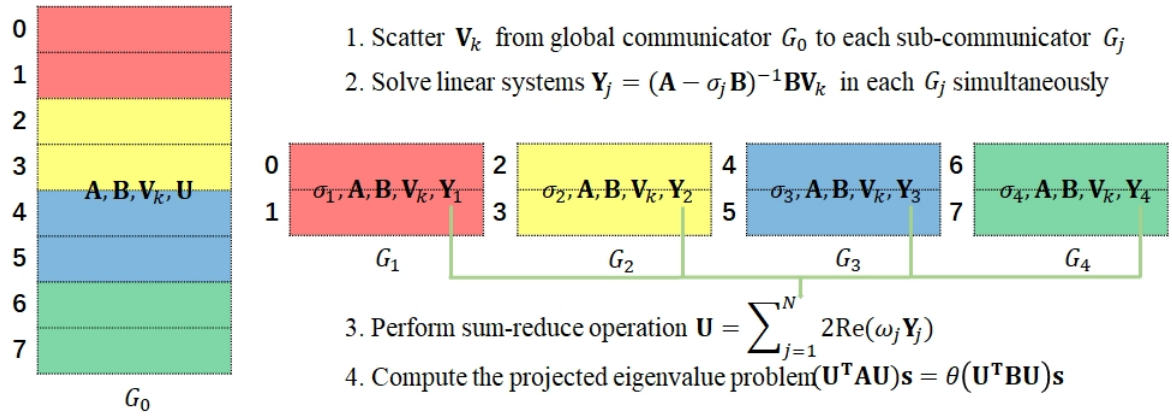

Figure 2: Computational flow of the parallel rational filter method at iteration step k, for $np = 8, N = n_{part} = 4$.

In this example $np = 8$ and $N = 4$, and the global communicator G_0 is split into 4 parts, i.e., each sub-communicator G_j manages 2 processes. For each G_j, we store the full $\mathbf{A}, \mathbf{B}$, and assign one pole σ_j. The right-hand-side vectors $\mathbf{V}_k$ maintain the same row-split distribution as the matrix pencil, consistently preserved across both the global communicator and all sub-communicators. After solving the required linear systems in each G_j, the solution vectors $\mathbf{Y}_j$ are scattered backward to the global communicator, where the sum-reduce operation is performed to compute the rational filtered vectors $\mathbf{U}$. Then, we form the projected pencil $(\mathbf{U}^\top\mathbf{AU}, \mathbf{U}^\top\mathbf{BU})$, where $\mathbf{U}, \mathbf{A}$ and $\mathbf{B}$ have the same row-split distribution in the global communicator. As the projected problem is small, we store the matrix product in the root process of G_0. Finally, we solve the eigenpairs of the small projected problem, which are used for updating the approximated vectors $\mathbf{V}_{k+1}$ for next iteration step.

As we can see, there is no explicit orthogonalization operation, so there is no communication amongst the G_j's, i.e., groups of processes corresponding to different poles. However, the sum-reduce operation requires all the linear systems to be solved in each sub-communicator. When the linear systems assigned in each sub-communicator have different difficulty levels, there might be waiting time in the MPI communication between the sub-communicators.

2.2. The choice of poles

The main difference between the various rational filter methods is the selection of poles σ_j, and corresponding weights w_j. Apart from the classical quadrature rules, there are other interesting ways to select the poles and weights [4, 6, 15, 16].

To solve the associated linear systems for each pole, iterative solvers are preferable for large-scale problems. However, we have to take the position of the poles into account carefully. Given a pole σ we can represent it as

$$\sigma = \mathrm{Re}(\sigma)(1 + \alpha\imath),$$

where α equals $\mathrm{Re}(\sigma)/\mathrm{Im}(\sigma)$ and indicates the degree of deviation from the real axis. The effect of α on solving the associated linear systems is analysed by Wang, et al. [7]. On the one hand, for example, part of the poles generated from Gauss-Chebyshev quadrature, are very close to the real axis (i.e., small α) usually result in ill-conditioned linear systems, which are more difficult to solve by iterative solvers. On the other hand, as pointed out in [17, 16, 4, 7], moving poles away from the spectrum (or real axis for Hermitian problem) can reduce the difficulty of solving the associated linear systems by using an iterative solver. In this paper, the poles and weights from the shifted Laplace rational filter (SLRF) are determined with parameters $\alpha = 1$ and $\beta = 0.01$ which is a relaxation parameter [7].

3. Numerical experiments

3.1. Finite element model of vibration

We consider the following vibration equation

$$\begin{cases} \boldsymbol{\sigma} \cdot \nabla - \rho \ddot{\mathbf{u}} &= 0, \quad \text{in } \Omega, \\ \boldsymbol{\sigma} \cdot \mathbf{n} = \bar{\mathbf{t}} &= 0, \quad \text{on } \Gamma_N, \\ \mathbf{u} &= 0, \quad \text{on } \Gamma_D, \end{cases} \tag{4}$$

where $\mathbf{u}$ is the displacement, $\boldsymbol{\sigma}$ is the stress and ρ is the density of the material. The domain is denoted by Ω, Γ_D denotes the Dirichlet boundary condition and Γ_N denotes the Neumann boundary condition, $\mathbf{n}$ is the outer normal vector of the boundary of the domain $\partial\Omega$. The derivation of the weak formulation of (4) for a finite element discretization is discussed by Wang, An, Xie, and Mo [18].

In the following, we introduce three mechanical models for numerical tests. These models have the same material parameters: the density of the material is $1.0\mathrm{kg/m}^2$, the Poisson's ratio is 0.29, the Young's modulus is $21.5\mathrm{N/m}^2$.

3.1.1. Pyramid model

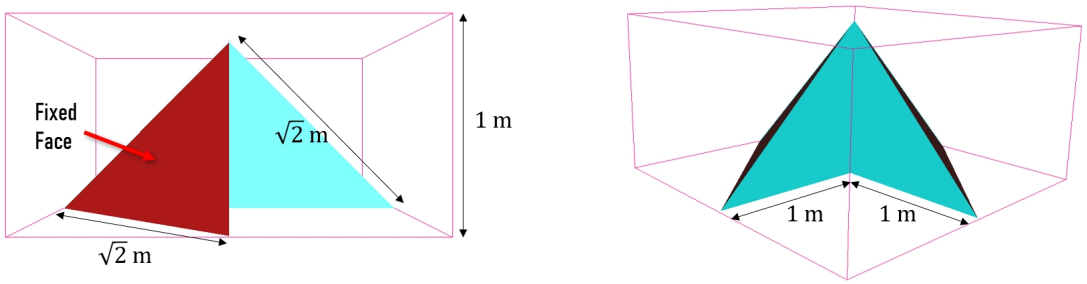

Figure 3: The shape of pyramid model.

The first model is a three dimensional mechanical model which looks like a pyramid. The foundation is fixed on the ground and two surfaces in red are fixed, see Figure 3. The domain is partitioned by tetrahedral elements and the equations are discretized by first order finite elements. Two scales of this problem will be tested:

- Pyramid S1: $n = 36,339$ with $12,113$ grid nodes;
- Pyramid S2: $n = 268,353$ with $89,451$ grid nodes;

where n is the number of the DOFs.

For Pyramid S1, the first 20 smallest eigenvalues are located in [350.6497, 1530.2328], and the first 100 smallest eigenvalues are located in [350.6497, 4272.0205]. For Pyramid S2, the first 20 smallest eigenvalues are located in [344.4414,1477.9992], and the first 100 smallest eigenvalues are located in [344.4414,3907.2456].

3.1.2. Hollow platform model and fish-like model

The hollow platform model originates from cutting out a cylinder from a platform. The surface of the cylinder is fixed. The dimension of the obtained GEP is 37161. The fish-like model is generated by rotating a given two dimensional region along the x-axis. The obtained body looks like a fish and the lower surface is fixed. The dimension of the obtained GEP is 55248. For more details about this two models, we refer to Wang, et al. §4 in [7].

3.2. Setting

The computations were carried out on a Linux compute server with two 28-cores Intel Xeon Gold 6330N processors, and 1 TB of RAM. All methods are written in C, built on top of PETSc [19] and SLEPc [20], and compiled with the `-O3` optimization flag. For a fair comparison, the same matrix-vector products, inner products interfaces integrated in PETSc and SLEPc are called for all methods. In the following, we define $\mathbf{r}_k = \frac{(\mathbf{A}-\theta_k\mathbf{B})\mathbf{x}_k}{\theta_k\|\mathbf{x}_k\|_\mathbf{B}}$ as the relative residual vector where θ_k is Rayleigh quotient associated with the approximate eigenvector $\mathbf{x}_k$. As stopping criteria, we use $\|\mathbf{r}_k\|_2 < 10^{-8}$.

We have to point out that the incomplete LU factorization method (ILU) and Boomer-AMG method integrated in HYPRE [21] lack robust support for complex linear system computations. Unless otherwise stated, the linear solver is GCR (generalized conjugate residual) method with the GAMG preconditioner in PETSc. We reuse the preconditioners for each pole to avoid assembling the associated matrices repeatedly during the construction phase of GAMG. For the linear system $\mathbf{Mu} = \mathbf{f}$, the stopping criteria for the linear solver requests the relative residual norm $\|\mathbf{f} - \mathbf{M}\mathbf{u}_k\|_2/\|\mathbf{f}\|_2$ to be less than 10^{-10} or the number of iterations has reached the maximum iteration number, set to 10000.

For the parallel shifted Laplace rational filter (ParaSLRF) the values of poles and corresponding weights are determined by the parameters $\alpha = 1$ and $\beta = 0.01$. The dimension of the search space L should be larger than the number of desired eigenvalues (NEV). Therefore, we choose $L = 1.2 \times \text{NEV}$.

3.3. Results and efficiency analysis

The first 20 and 100 smallest eigenpairs of the aforementioned mechanical models are computed. For each computation task we use $np = 16$ processes. For each rational filter method we use $N = 4$ poles (in the upper plane). The processes are split into $n_{part} = 4$ parts, i.e., each sub-communicator is bound to one pole and manages $np_{sub} = np/n_{part} = 4$ processes for solving the associated linear systems.

In Table 1, the reported results include the number of outer iterations **Iter** and the CPU time $\mathbf{Time_{Total}}$. Moreover, we record the maximum time, denoted by $\mathbf{Time^{Max}_{LinearSolve}}$ for solving the required linear systems across all sub-communicators and we document its ratio, denoted by **Ratio**, relative to the minimum time of linear solves in the whole procedure.

Table 1: Numerical results.

Model	Interval	Filter	Iter	$\text{Time}_{\text{Total}}$	$\text{Time}^{\text{Max}}_{\text{LinearSolve}}$	Ratio
Pyramid S1	NEV=20 (0, 1531]	Midpoint	15	2.0292E+03	2.0279E+03	4.9
		Gauss-Legendre	5	1.2217E+03	1.2210E+03	9.0
		Gauss-Chebyshev	9	4.0055E+03	4.0045E+03	16.3
		ParaSLRF	10	3.2080E+02	3.1960E+02	1.0
	NEV=100 (0, 4273]	Midpoint	10	8.2540E+03	8.2484E+03	7.2
		Gauss-Legendre	5	7.8351E+03	7.8322E+03	13.5
		Gauss-Chebyshev	9	2.7157E+04	2.7152E+04	26.0
		ParaSLRF	9	1.1535E+03	1.1484E+03	1.0
Hollow platform	NEV=20 (0, 215]	Midpoint	4	4.2453E+03	4.2434E+03	4.7
		Gauss-Legendre	4	7.7003E+03	7.6985E+03	8.6
		Gauss-Chebyshev	4	1.4001E+04	1.3999E+04	15.9
		ParaSLRF	6	1.5403E+03	1.5382E+03	1.0
	NEV=100 (0, 1507]	Midpoint	7	3.5463E+04	3.5456E+04	6.4
		Gauss-Legendre	4	4.0816E+04	4.0812E+04	12.8
		Gauss-Chebyshev	5	1.0494E+05	1.0493E+05	25.8
		ParaSLRF	7	5.4263E+03	5.4121E+03	1.0
Fish-like	NEV=20 (0, 1858]	Midpoint	6	6.3236E+03	6.3226E+03	4.1
		Gauss-Legendre	5	9.8096E+03	9.8088E+03	7.8
		Gauss-Chebyshev	6	2.1733E+04	2.1733E+04	14.3
		ParaSLRF	7	1.6070E+03	1.6061E+03	1.1
	NEV=100 (0, 10239]	Midpoint	11	3.9331E+04	3.9324E+04	6.2
		Gauss-Legendre	6	4.0080E+04	4.0076E+04	12.5
		Gauss-Chebyshev	9	1.1449E+05	1.1449E+05	24.2
		ParaSLRF	7	3.8386E+03	3.8346E+03	1.0

From Table 1, we observe that ParaSLRF achieves the best performance in our experiments, it successfully completes all tasks with the shortest execution time. Instead, the rational filters based on the quadrature rules do not work well. For instance, the Gauss-Chebyshev filter demonstrates the highest computational cost, requiring up to 29.8 times more CPU time compared to the ParaSLRF method, for solving the first 100 eigenpairs of the fish-like model. From the penultimate column, we see that solving the associated linear systems is the most expensive part in the whole procedure. However, we have to point out that the allocated processes are not fully utilized and many processes are idle most of the

time, for the rational filters based on quadrature rules. From the last column of Table 1, we can observe that the **Ratio** value can be really high which means severe imbalance between the sub-communicators. The reason for this phenomenon is that the time for solving the linear systems in each sub-communicator is quite different. The reduction of vectors $\mathbf{U}$ requires synchronization, forcing faster sub-communicators to wait for the slowest one to complete the linear solve. On the contrary, ParaSLRF maintains a consistent **Ratio** value approximately equal to 1, indicating that (i) the computational cost for solving the associated linear systems is almost constant across all sub-communicators; (ii) the utilization of resources is high and the MPI waiting overhead is low.

To validate our statements, we record the additional MPI waiting time (i.e. the maximum idle time across all sub-communicators), denoted by $\mathbf{Time}^{\mathbf{Max}}_{\mathbf{MPIWait}}$, during the stage of the sum-reduce operation and report its proportion to the total CPU time, denoted by **Prop**, for each task. The results are represented in Table 2.

From Table 2, we can observe that the rational filter methods based on quadrature rules have high MPI waiting time. The most evident example is the Gauss-Chebyshev rational filter with a maximum MPI waiting time of more than 92% of the total time, that is, at least a quarter of the processes are idle more than 92% of the time. ParaSLRF on the other hand has a MPI waiting time of around 4% of the total CPU time. All computing resources are fully utilized as much as possible, because of the good work-load balancing.

4. Enhanced ParaSLRF

As shown in Table 1, solving the linear systems dominates the computational cost of the algorithm. To further reduce the computational cost and improve the performance of ParaSLRF we adopt two useful strategies.

The first improvement consists of locking the approximate eigenpairs that satisfy the stopping criteria and reduce the number of vectors required to be filtered. Next, in the subsequent iterations, we make the filtered vectors and the converged eigenvectors orthogonal with respect to the mass matrix $\mathbf{B}$.

The second improvement lies in the fact that $1/(\theta-\sigma)\mathbf{v}$, as the initial guess is a good choice to accelerate the linear solve, under the assumption that $(\theta, \mathbf{v})$ is a good approximation of one of the eigenpairs $(\lambda, \mathbf{x})$. There are two arguments to see this. First, employing an initial approximation close to the exact solution $1/(\lambda-\sigma)\mathbf{x}$, seems an obvious strategy. Second, it is known that a fixed number of iterations for the linear system with the Cayley transformation usually leads to an eigenvector approximation with a lower residual norm [17, 22, 23]; using the proposed initial guess for shift-and-invert transformation is actually applying the Cayley transformation with initial zero guess, as the following proposition shows.

Proposition 4.1. *Given the pair* $(\theta, \mathbf{v}) \in \mathbb{R} \times (\mathbb{R}^n \setminus \{0\})$. *Let* $\mathcal{K}_m^{T^C}(\mathbf{A} - \sigma\mathbf{B}, \mathbf{res}_C^{(0)})$ *be the (preconditioned) Krylov subspace formed for solving the "Cayley transformation" linear system*

$$T^C : (\mathbf{A} - \sigma\mathbf{B})\mathbf{u} = (\mathbf{A} - \theta\mathbf{B})\mathbf{v} \quad \textit{with initial guess } \mathbf{u}^{(0)} = 0, \tag{5}$$

Table 2: Extra MPI waiting time and its proportion to total solution time.

Model	Interval	Filter	$\text{Time}^{\text{Max}}_{\text{MPIWait}}$	Prop (%)
Pyramid S1	NEV=20 (0, 1531]	Midpoint	1.6182E+03	79.75
		Gauss-Legendre	1.0849E+03	88.80
		Gauss-Chebyshev	3.7596E+03	93.86
		ParaSLRF	8.1755E+00	2.52
	NEV=100 (0, 4273]	Midpoint	7.1097E+03	86.14
		Gauss-Legendre	7.2531E+03	92.57
		Gauss-Chebyshev	2.6109E+04	96.14
		ParaSLRF	1.0824E+01	0.94
Hollow platform	NEV=20 (0, 215]	Midpoint	3.3365E+03	78.59
		Gauss-Legendre	6.8059E+03	88.38
		Gauss-Chebyshev	1.3119E+04	93.70
		ParaSLRF	3.4959E+01	2.27
	NEV=100 (0, 1507]	Midpoint	2.9926E+04	84.39
		Gauss-Legendre	3.7624E+04	92.18
		Gauss-Chebyshev	1.0086E+05	96.11
		ParaSLRF	1.2405E+02	2.29
Fish-like	NEV=20 (0, 1858]	Midpoint	4.7763E+03	75.53
		Gauss-Legendre	8.5520E+03	87.18
		Gauss-Chebyshev	2.0209E+04	92.99
		ParaSLRF	8.8895E+01	5.53
	NEV=100 (0, 10239]	Midpoint	3.3013E+04	83.94
		Gauss-Legendre	3.6881E+04	92.02
		Gauss-Chebyshev	1.0976E+05	95.87
		ParaSLRF	1.4755E+02	3.84

and let $\mathcal{K}_m^{T^{SI}}(\mathbf{A}-\sigma\mathbf{B}, \mathbf{res}_{SI}^{(0)})$ *be the Krylov space of dimension* m *for solving the shift-and-invert transformation system*

$$T^{SI} : (\mathbf{A}-\sigma\mathbf{B})\mathbf{y} = \mathbf{B}\mathbf{v} \quad \textit{with initial guess } \mathbf{y}^{(0)} = \frac{1}{\theta-\sigma}\mathbf{v}, \tag{6}$$

then

(i.) the initial residuals satisfy $\mathbf{res}_{SI}^{(0)} = \frac{1}{\sigma-\theta}\mathbf{res}_{C}^{(0)}$,

(ii.) the resulting Krylov subspaces $\mathcal{K}_m^{T^C}(\mathbf{A}-\sigma\mathbf{B}, \mathbf{res}_{C}^{(0)})$ *and* $\mathcal{K}_m^{T^{SI}}(\mathbf{A}-\sigma\mathbf{B}, \mathbf{res}_{SI}^{(0)})$ *are identical.*

Proof. Split the right-hand side of (5),

$$(\mathbf{A}-\sigma\mathbf{B})\mathbf{u} = (\mathbf{A}-\theta\mathbf{B})\mathbf{v} = (\mathbf{A}-\sigma\mathbf{B})\mathbf{v} + (\sigma-\theta)\mathbf{B}\mathbf{v}.$$

thus,

$$\begin{aligned}(\mathbf{A}-\sigma\mathbf{B})(\mathbf{u}-\mathbf{v}) &= (\sigma-\theta)\mathbf{B}\mathbf{v},\\ (\mathbf{A}-\sigma\mathbf{B})\frac{\mathbf{u}-\mathbf{v}}{\sigma-\theta} &= \mathbf{B}\mathbf{v}.\end{aligned}$$

Let $\mathbf{y} = \frac{\mathbf{u}-\mathbf{v}}{\sigma-\theta}$, we know that $\mathbf{u}^{(0)} = 0$ is equivalent to $\mathbf{y}^{(0)} = \frac{1}{\theta-\sigma}\mathbf{v}$. Let P^{-1} denote the preconditioner. For system (5) the initial (preconditioned) residual is

$$\mathbf{res}_{C}^{(0)} = P^{-1}(\mathbf{A}-\theta\mathbf{B})\mathbf{v},$$

and for system (6) the initial residual is

$$\mathbf{res}_{SI}^{(0)} = P^{-1}\mathbf{B}\mathbf{v} - P^{-1}(\mathbf{A}-\sigma\mathbf{B})\frac{\mathbf{v}}{\theta-\sigma} = \frac{1}{\sigma-\theta}P^{-1}(\mathbf{A}-\theta\mathbf{B})\mathbf{v} = \frac{1}{\sigma-\theta}\mathbf{res}_{C}^{(0)}.$$

The residuals are the starting vectors of the two Krylov spaces. Since these vectors lie in the same direction and the (preconditioned) matrix is the same for both linear system, the two Krylov spaces must be the same. □

The connection with the Cayley transform, shows that the scaled approximate eigenvector $\mathbf{y}^{(0)} = \frac{1}{\theta-\sigma}\mathbf{v}$ is a good initial guess of the iterative solver for the shift-and-invert linear system. One can fix the maximum number of linear iterations $\mathrm{It}_{\text{max-linear}}$ of the linear solver, which is beneficial to reduce the cost of the linear solve and achieve a better load balance across all sub-communicators.

The enhanced ParaSLRF is presented in Algorithm 2. On Line 10, the associated linear systems are solved by an iterative solver with the fixed maximum iteration number $\mathrm{It}_{\text{max-linear}}$. On Lines 15 to 17, the filtered vectors $\mathbf{U}$ which contain the information of the desired eigenvectors, are orthogonalized with the converged eigenvectors $\mathbf{X}$, in order to avoid the converged eigenpairs from reappearing in the subsequent iterations. On Lines 19 to 20, the converged eigenvectors are removed from $\mathbf{V}_{k+1}$ so that only n_{act} columns remain, where n_{act}

Algorithm 2 Enhanced ParaSLRF

Input: $\mathbf{A} \in \mathbb{R}^{n\times n}$, $\mathbf{B} \in \mathbb{R}^{n\times n}$, interval $(0, \gamma]$, NEV the number of the wanted eigenvalues, L number of columns of the starting vectors where $L \geq$ NEV, poles $\sigma_1, \ldots, \sigma_N \in \mathbb{C}$ and corresponding weights $w_1, \ldots, w_N \in \mathbb{C}$, np number of processes, n_{part} number of partitions, maximum linear iteration $\mathrm{It}_{\text{max-linear}}$.

Output: converged eigenpairs $[\Theta, \mathbf{X}]$.

/* Initialization phase. */

Load matrix pencil $\mathbf{A}, \mathbf{B}$ and generate random starting vectors $\mathbf{V}_1 \in \mathbb{R}^{n\times L}$ in global communicator G_0.

Split the global communicator into n_{part} sub-communicators where the number of assigned processes are $np_{sub} = np/n_{part}$ and $N_{sub} = N/n_{part}$. Scatter and redistribute $\mathbf{A}, \mathbf{B}$ and $\mathbf{V}_1$ from the global communicator to each sub-communicator G_j.

Set $n_{act} = L$ (n_{act} counts the dimension of the active subspace), $n_{conv} = 0$ (n_{conv} counts the number of converged eigenpairs).

for $k = 1, 2, \ldots$ until all of the wanted eigenpairs have converged **do**

  /* Parallel over the n_{part} partitions. */

  for $j = 1, 2, \ldots, n_{part}$ **do**

    for $i = 1, 2, \ldots, N_{sub}$ **do**

      Set $\sigma_{j,i} = \sigma_{(j-1)N_{sub}+i}$ and $w_{j,i} = w_{(j-1)N_{sub}+i}$.

      Solve $(\mathbf{A} - \sigma_{j,i}\mathbf{B})\mathbf{Y}_{j,i} = \mathbf{B}\mathbf{V}_k$ by using the distributed linear iterative solver with fixed maximum linear iterations $\mathrm{It}_{\text{max-linear}}$ in each sub-communicator G_j at the same time.

    end for

  end for

  Concat $\mathbf{Y}_j = [\mathbf{Y}_{j,1}, \mathbf{Y}_{j,2}, \ldots, \mathbf{Y}_{j,N_{sub}}]$ in each sub-communicator.

  Scatter each $\mathbf{Y}_j$ from sub-communicators to the global communicator where a sum-reduce operation $\mathbf{U} = \sum_{j=1}^{n_{part}} \sum_{i=1}^{N_{sub}} 2\,\mathrm{Re}(w_{j,i}\mathbf{Y}_{j,i})$ is performed.

  if $n_{conv} > 0$ **then**

    Make the filtered vectors $\mathbf{U}$ and converged eigenvectors $\mathbf{X}$ orthogonal with respect to the mass matrix $\mathbf{B}$.

  end if

  Solve the projected eigenvalue problem $(\mathbf{U}^\top\mathbf{A}\mathbf{U})\mathbf{s}_m = \theta_m(\mathbf{U}^\top\mathbf{B}\mathbf{U})\mathbf{s}_m$ and let $\mathbf{v}_m = \mathbf{U}\mathbf{s}_m$, $m = 1, 2, \ldots, n_{act}$.

  Examine the convergence of each pair $(\theta_m, \mathbf{v}_m)$. If converged, append θ_m to Θ and duplicate $\mathbf{v}_m$ to $\mathbf{X}$. Update the numbers n_{conv}, and $n_{act} = L - n_{conv}$.

  Remove the converged vectors in $\mathbf{V}_k$ and resize $\mathbf{V}_{k+1} = [\mathbf{v}_1, \ldots, \mathbf{v}_{n_{act}}]$. Scatter $\mathbf{V}_{k+1}$ to each sub-communicator again.

  Let $\mathbf{Y}_{j,i}^{init} = [\frac{1}{\theta_1 - \sigma_{j,i}}\mathbf{v}_1, \ldots, \frac{1}{\theta_{n_{act}} - \sigma_{j,i}}\mathbf{v}_{n_{act}}]$, $j = 1, 2, \ldots, n_{part}$, and $i = 1, 2, \ldots, N_{sub}$, be the initial guess of the iterative linear solver for the next iteration.

end for

is the dimension of the active subspace for the next rational filtering iteration. On Line 21, the initial guess $\mathbf{Y}_{j,i}^{init}$ is assembled by the scaled approximate eigenvectors to accelerate the next linear solve.

Although the new algorithm adds overhead due to orthogonalization, it significantly reduces the cost of the linear solves, compared to Algorithm 1. It is not easy to set a default value for the maximum number of inner linear iterations $\mathrm{It}_{\text{max-linear}}$. In order to investigate the effect of $\mathrm{It}_{\text{max-linear}}$ on the convergence and performance of the enhanced ParaSLRF, we computed the first 100 eigenpairs of aforementioned models, for different values of $\mathrm{It}_{\text{max-linear}}$. The CPU time and the number of outer iterations for each task can be found in the Figures 4—6, respectively.

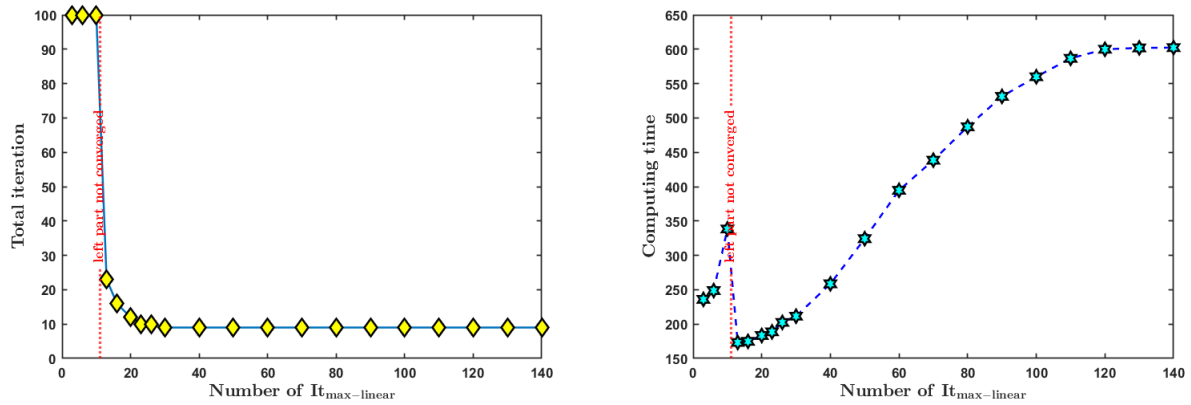

Figure 4: The outer iterations (left) and CPU time (right) of the enhanced ParaSLRF with different $\mathrm{It}_{\text{max-linear}}$ for the pyramid S1 model ($N = 36339$).

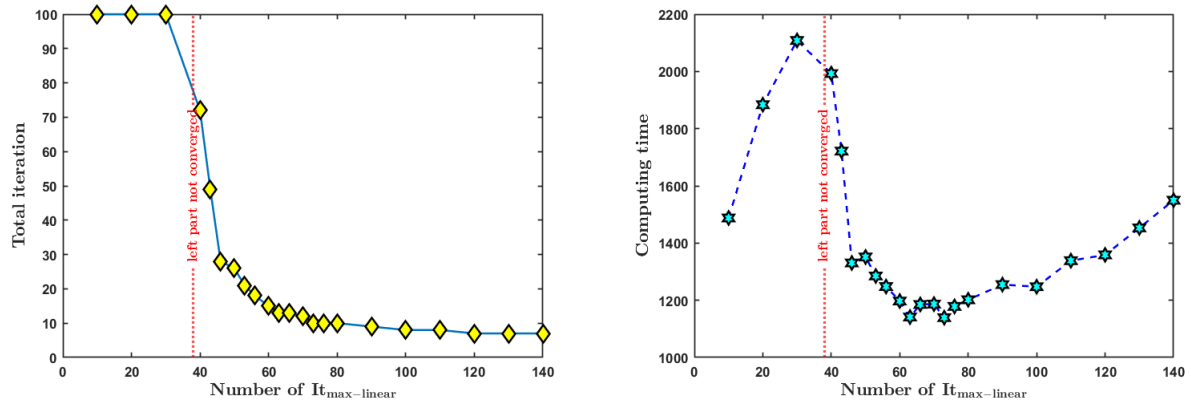

Figure 5: The outer iterations (left) and CPU time (right) of the enhanced ParaSLRF with different $\mathrm{It}_{\text{max-linear}}$ for hollow platform model ($N = 37161$).

In Figures 4—6 the red dashed line shows the minimal value of $\mathrm{It}_{\text{max-linear}}$ to obtain convergence of the wanted eigenpairs in 100 outer iterations. It is not surprising that the best value of $\mathrm{It}_{\text{max-linear}}$ for achieving the lowest CPU time is problem dependent. Although

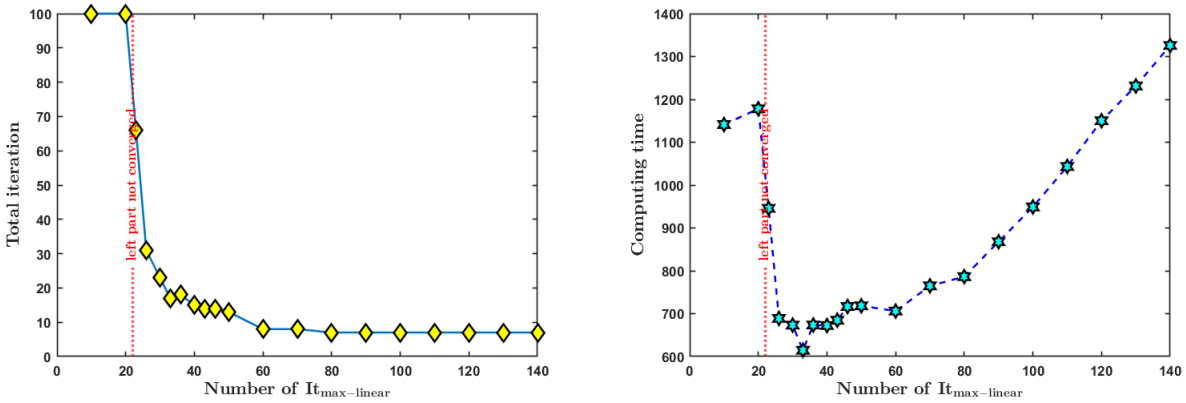

Figure 6: The outer iterations (left) and CPU time (right) of the enhanced ParaSLRF with different $\mathrm{It}_{\text{max-linear}}$ for fish-like model ($N = 55248$).

a high $\mathrm{It}_{\text{max-linear}}$ can reduce the number of outer iterations this may also lead to excessive CPU time. Therefore, a careful trade-off is required when selecting this parameter. Through experimental validation, the optimal $\mathrm{It}_{\text{max-linear}}$ was set to 13, 63, and 33 for the pyramid S1, hollow platform, and fish-like models, respectively.

To demonstrate the improved performance, we compute the first 20 and 100 smallest eigenpairs of aforementioned mechanical models with corresponding best value of $\mathrm{It}_{\text{max-linear}}$. The computational environment and the notation used for reporting the results are the same as previously described in Section 3.3. The results are shown in Table 3.

Table 3: Performance improvement.

Model	Interval	Method	Iter	$\mathrm{Time}_{\mathrm{Total}}$	$\mathrm{Time}^{\mathrm{Max}}_{\mathrm{MPIWait}}$	Prop (%)
Pyramid S1	NEV=20 (0, 1531]	ParaSLRF	10	320.852	8.175	2.50
		Enhanced ParaSLRF	20	35.247	0.514	1.46
	NEV=100 (0, 4273]	ParaSLRF	9	1153.512	10.824	0.93
		Enhanced ParaSLRF	23	173.227	1.347	0.78
Hollow platform	NEV=20 (0, 215]	ParaSLRF	6	1540.356	34.959	2.26
		Enhanced ParaSLRF	15	244.410	3.359	1.37
	NEV=100 (0, 1507]	ParaSLRF	7	5426.306	124.050	2.28
		Enhanced ParaSLRF	13	1147.207	16.263	1.42
Fish-like	NEV=20 (0, 1858]	ParaSLRF	7	1607.061	88.895	5.53
		Enhanced ParaSLRF	50	258.127	4.319	1.67
	NEV=100 (0, 10239]	ParaSLRF	7	3838.613	147.550	3.84
		Enhanced ParaSLRF	17	613.464	8.967	1.46

From Table 3, we can observe that the computational cost and CPU time consumption are significantly reduced although the two applied strategies slightly increase the number

of outer iterations. In addition, the enhanced version is at least 4.7 times faster than the original one for all tasks and can be up to 9.1 times faster when computing 20 eigenpairs of pyramid S1 model.

We also find that, for the enhanced ParaSLRF, the extra MPI waiting time reduces significantly; the proportion of the extra MPI waiting time to the total CPU time also reduces further, which represents better utilization of resources and load-balancing.

4.1. Strong scalability

To investigate the strong scalability of the enhanced ParaSLRF, we use the pyramid S2 model represented in §3.1.1 with refined mesh, as test example. The discretized system has 268,353 degrees of freedom. We set $\mathrm{It}_{\text{max-linear}} = 60$ and intend to compute the first 100 smallest eigenpairs of this system, with different computing resources configuration. The computations were carried out on a cluster where each computational node has 2 Intel Xeon Gold 6240 CPUs@2.6 GHz (18 cores per CPU), and 192 GiB RAM.

For the Level-1 scalability test, we use $N = 16$ poles and split the poles into $n_{part} = 1, 2, 4, 8, 16$ parts, respectively. We allocate $np_{sub} = 18$ processes bound to one socket for each part. The total number of processes np varies as 18,36,72,144,288. We record the CPU time and compute the corresponding speed-up. The results are shown in Figure 7.

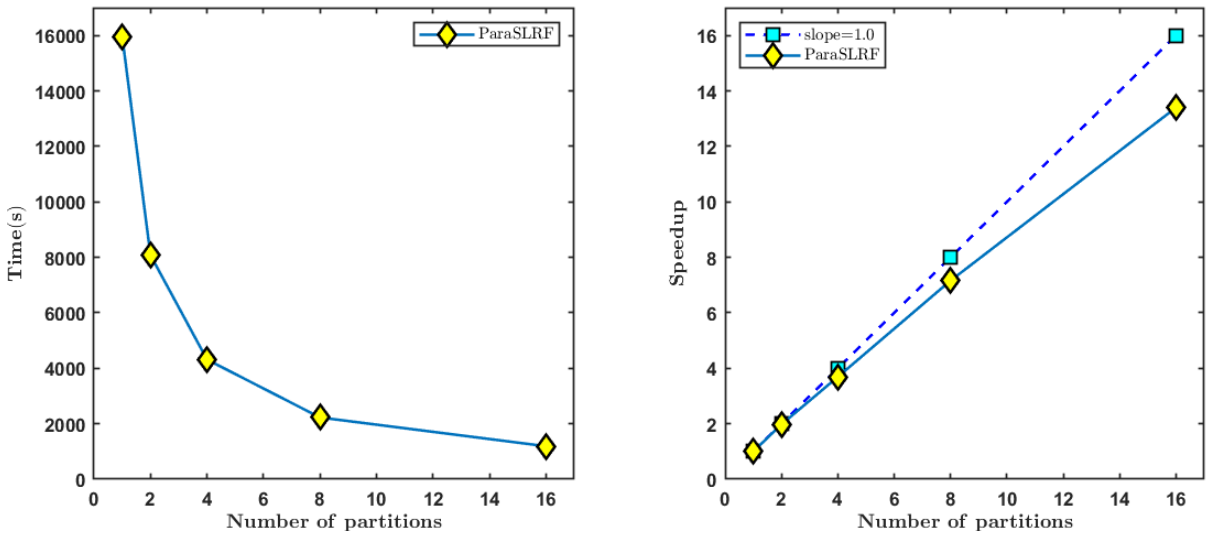

Figure 7: Strong scalability of the Level-1 parallelism.

From Figure 7 we observe that the enhanced ParaSLRF performs well on the Level-1 strong scalability, which is close to linear. These results indicate that prioritizing resources allocation at the Level-1 is critical for achieving a good parallel performance.

As for the Level-2 scalability test, it is actually the strong scalability test of the linear iterative solver. Thus, we do not need to use a large number of poles for this test. Here, we only use $N = 4$ poles and split all processes into $n_{part} = 4$ parts. The number of allocated processes np_{sub} starts from $9, 18, 36, 72, 144$ for each sub-communicator (with a

total of 576 cores). The CPU time and corresponding speed-up for each computation task are represented in Figure 8.

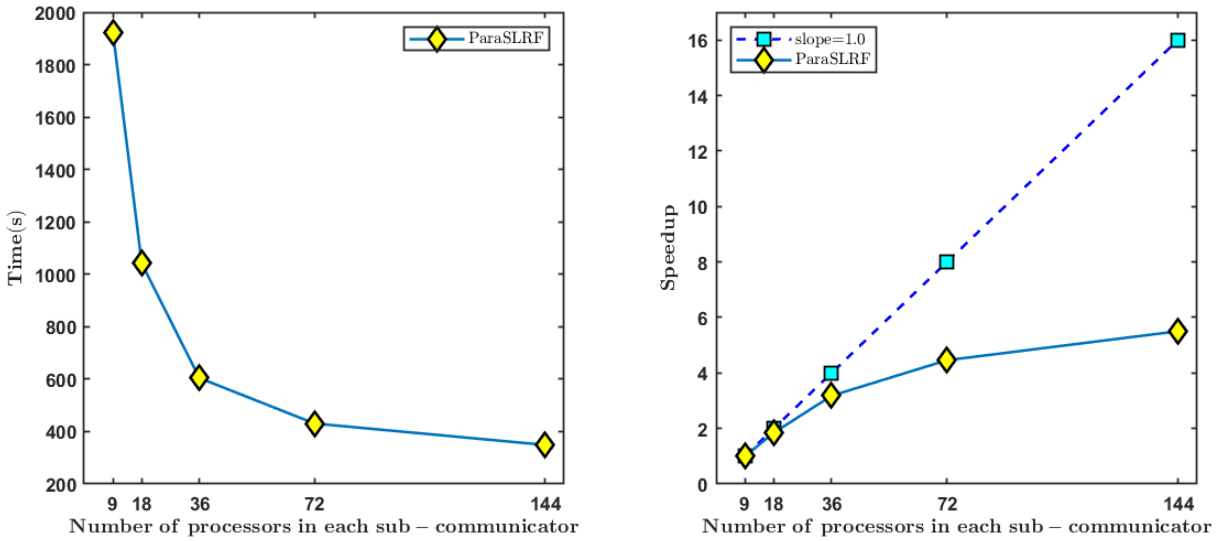

Figure 8: Strong scalability of the Level-2 parallelism.

From Figure 8 we observe that the Level-2 strong scalability of the enhanced ParaSLRF is not so good. On the one hand, solving sparse linear systems is memory-bound, which means that it is hard to achieve the ideal scalability for any linear solver; and on the other hand, as the number of processes grows rapidly the number of degrees of freedom handled by each process is smaller, which can pull down the performance of GAMG. In general, this is a normal scalability performance of the GCR iterative method preconditioned by GAMG.

4.2. Fixed computing resources

Because of the two levels of parallelism within the enhanced ParaSLRF, it is valuable to explore strategies for making the most efficient use of fixed computing resources (e.g., the limited number of processes), to achieve the best parallel performance. We have to consider how to allocate computing resources in each parallel level. If 4 computational nodes (with a total of 144 cores) are available and 16 poles are used for the computations, there are five ways to allocate the resources across the two-level parallelism. It is reasonable to set $n_{part} = 1, 2, 4, 8, 16$ for the number of sub-communicators in Level-1 and utilize $np_{sub} = 144, 72, 36, 18, 9$ processes in each sub-communicator, respectively.

Table 4 shows the CPU time for solving the first 100 eigenpairs of the pyramid S2 model, with different distributions of computing resources across the Level-1 and Level-2. It is no surprise that it is optimal to place as many resources as possible at the Level-1 and the best performance is achieved when the required linear systems only involve one pole in each sub-communicator.

Table 4: The CPU time for computing 100 eigenpairs of the Pyramid S2 model with different distributions of computing resources. Increasing the number of MPI processes at Level-2 (i.e., np_{sub}) reduces the memory required by each process (i.e., the number of rows of the vectors and matrices assigned to each process).

n_{part}	np_{sub}	rows	Time(s)
1	144	1864	3420.508
2	72	3727	2817.676
4	36	7454	2369.271
8	18	14909	2199.769
16	9	29817	2128.285

5. Conclusion

In this paper, we developed a parallel shifted Laplace rational filter method (ParaSLRF) for computing all eigenvalues located in the interval $(0, \gamma]$, and corresponding eigenvectors, for generalized eigenvalue problems arising from mechanical vibrations.

Numerical results indicate that, compared to the rational filters based on quadrature rules, ParaSLRF performs much better in terms of computational cost and CPU time. Moreover, we observe that there is less communication waiting overhead between sub-communicators for ParaSLRF, i.e., it offers excellent load-balancing capability which makes ParaSLRF highly suitable for parallel computing.

In addition, we introduced two improvements to enhance the performance of ParaSLRF. First, converged eigenpairs are locked by making the filtered vectors $\mathbf{B}$-orthogonal to the converged vectors. A significant reduction of the computational cost is obtained with only a small additional overhead. Second, we selected a good initial guess for the linear iterative solver and fixed the maximum linear iteration number to reduce the computational cost further. This led to a better load-balancing. Scalability tests show that the enhanced version has very good two-level parallel scalability. Moreover, we observe that distributing as much computing resources as possible at Level-1 is the optimal distribution with limited resources.

References

[1] Y. Saad, Numerical Methods for Large Eigenvalue Problems: revised edition, SIAM, 2011.

[2] T. Sakurai, H. Sugiura, A projection method for generalized eigenvalue problems using numerical integration, J. Comput. Appl. Math. 159 (1) (2003) 119–128.

[3] T. Ikegami, T. Sakurai, U. Nagashima, A filter diagonalization for generalized eigenvalue problems based on the Sakurai–Sugiura projection method, J. Comput. Appl. Math. 233 (8) (2010) 1927–1936.

[4] Y. Xi, Y. Saad, Computing partial spectra with least-squares rational filters, SIAM J. Sci. Comput. 38 (5) (2016) A3020–A3045.

[5] E. Polizzi, Density-matrix-based algorithm for solving eigenvalue problems, Phys. Rev. B 79 (11) (2009) 115112.

[6] S. Güttel, E. Polizzi, P. T. P. Tang, G. Viaud, Zolotarev quadrature rules and load balancing for the FEAST eigensolver, SIAM J. Sci. Comput. 37 (4) (2015) A2100–A2122.

[7] B. Wang, K. Meerbergen, R. Vandebril, H. An, Z. Mo, A shifted Laplace rational filter for large-scale eigenvalue problems, arXiv preprint arXiv:2503.21618 (2025).
[8] Y. A. Erlangga, C. Vuik, C. W. Oosterlee, On a class of preconditioners for solving the Helmholtz equation, Appl. Numer. Math. 50 (3-4) (2004) 409–425.
[9] Y. A. Erlangga, C. W. Oosterlee, C. Vuik, A novel multigrid based preconditioner for heterogeneous Helmholtz problems, SIAM J. Sci. Comput. 27 (4) (2006) 1471–1492.
[10] M. B. Van Gijzen, Y. A. Erlangga, C. Vuik, Spectral analysis of the discrete Helmholtz operator preconditioned with a shifted Laplacian, SIAM J. Sci. Comput. 29 (5) (2007) 1942–1958.
[11] M. M. M. Magolu, R. Beauwens, G. Warzée, Preconditioning of discrete Helmholtz operators perturbed by a diagonal complex matrix, Commun. Numer. Methods Eng. 16 (11) (2000) 801–817.
[12] M. M. M. Magolu, Incomplete factorization-based preconditionings for solving the Helmholtz equation, Int. J. Numer. Methods Eng. 50 (5) (2001) 1077–1101.
[13] M. M. M. Magolu, Performance of parallel incomplete LDL^t factorizations for solving acoustic wave propagation problems from industry, Numer. Linear Algebra Appl. 11 (8-9) (2004) 813–830.
[14] Y. Maeda, T. Sakurai, J. Roman, Contour integral spectrum slicing method in SLEPc, Tech. rep., Technical Report SLEPc STR-11, Universidad Politecnica de Valencia, Spain. (2016).
[15] A. P. Austin, L. N. Trefethen, Computing eigenvalues of real symmetric matrices with rational filters in real arithmetic, SIAM J. Sci. Comput. 37 (3) (2015) A1365–A1387.
[16] H. Ohno, Y. Kuramashi, T. Sakurai, H. Tadano, A quadrature-based eigensolver with a Krylov subspace method for shifted linear systems for Hermitian eigenproblems in lattice QCD, JSIAM Lett. 2 (2010) 115–118.
[17] K. Meerbergen, D. Roose, The restarted Arnoldi method applied to iterative linear system solvers for the computation of rightmost eigenvalues, SIAM J. Matrix Anal. Appl. 18 (1) (1997) 1–20.
[18] B. Wang, H. An, H. Xie, Z. Mo, A new subspace iteration algorithm for solving generalized eigenvalue problems in vibration analysis, J. Comput. Appl. Math. (2025) 116622.
[19] S. Balay, S. Abhyankar, M. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. Gropp, et al., PETSc users manual, Argonne National Laboratory (2019).
[20] J. E. Roman, C. Campos, E. Romero, A. Tomás, SLEPc users manual, D. Sistemes Informàtics i Computació Universitat Politècnica de Valencia, Spain, Report No. DSIC-II/24/02 (2015).
[21] R. D. Falgout, U. M. Yang, hypre: A library of high performance preconditioners, in: Int. Conf. Comput. Sci., Springer, 2002, pp. 632–641.
[22] R. B. Lehoucq, K. Meerbergen, Using generalized Cayley transformations within an inexact rational Krylov sequence method, SIAM J. Matrix Anal. Appl. 20 (1) (1998) 131–148.
[23] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, H. van der Vorst, Templates for the solution of algebraic eigenvalue problems: a practical guide, SIAM, 2000.