Domain decomposition techniques for contour integration eigenvalue solvers

Vassilis Kalantzis and Yousef Saad, University of Minnesota, Twin Cities
James Kestyn and Eric Polizzi, University of Massachusetts, Amherst

Contribution

Our focus lies on improving the parallel performance of the FEAST eigensolver by using domain decomposition approaches to solve the underlying linear systems.
Our implementation features:

1. A hybrid domain decomposition linear system solver which can be tuned to perform either as a direct solver or as a preconditioned iterative solver.
2. Up to three different levels of parallelism exploiting the MPI and OpenMP parallel paradigms.

We focus on the standard symmetric eigenvalue problem and show that domain decomposition can lead to faster, and often more scalable, computations within the FEAST framework.

The FEAST eigensolver

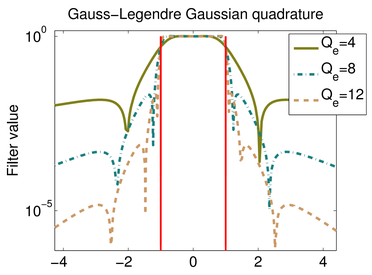
Let Γ be a smooth, counter-clockwise oriented curve, that encloses the eigenvalues of interest, e.g., those lying inside some real interval $[\alpha, \beta]$. Then, the spectral projector $\mathcal{P}$ associated with the eigenpairs of interest can be approximated as:

$$\mathcal{P} = \frac{1}{2i\pi}\int_\Gamma (\zeta I - A)^{-1} d\zeta \approx \sum_{j=1}^{Q_e} \omega_j (\zeta_j I - A)^{-1},$$

where $\{\zeta_j, \omega_j\}_{1\le j\le Q_e}$ are the Q_e complex pole-weight pairs resulted by using a quadrature rule, e.g. Gauss-Legendre. Each eigenvalue λ of A is transformed to an eigenvalue $\rho(\lambda)$ of $\rho(A) = \sum_{j=1}^{Q_e} \omega_j(\zeta_j I - A)^{-1}$, where

$$\rho(\zeta) = \sum_{j=1}^{Q_e} \frac{\omega_j}{\zeta_j - \zeta},$$

is a rational "filter" function which can be seen as an approximation of the step function in $[\alpha, \beta]$.



FEAST: Filtered Subspace iteration The FEAST eigensolver applies subspace iteration on the rational matrix $\rho(A) \rightarrow$ linear system solutions. Convergence depends on: a) the number of quadrature nodes Q_e, and b) the dimension $\hat{r}$ of the subspace.

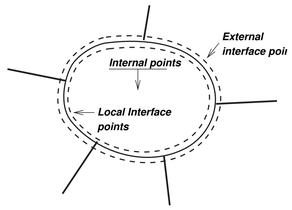
References

[1] J. Kestyn, V. Kalantzis, Eric Polizzi, Yousef Saad. PFEAST: A High Performance Sparse Eigenvalue Solver Using Distributed-Memory Linear Solvers, Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2016.

[2] V. Kalantzis, James Kestyn, Eric Polizzi, Yousef Saad. Domain decomposition-based contour integration approaches for the solution of symmetric eigenvalue problems. Submitted, 2016.

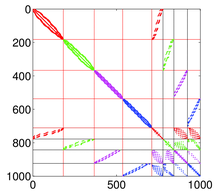
Domain decomposition

Graph partitioning The discretized domain is partitioned in P (non-overlapping) subdomains. Three types of unknowns appear: a) interior, b) local-interface, c) external-interface.



Matrix reordering Reorder equations-unknowns so that interior variables across all subdomains appear first:

$$A = \begin{pmatrix} B_1 & & & E_1 \\ & \ddots & & \vdots \\ & & B_P & E_P \\ E_1^T & \dots & E_P^T & C \end{pmatrix}$$



Block inverse representation $(A-\zeta I)^{-1}$ can be written as:

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

where $F(\zeta) = (B-\zeta I)^{-1}E$ and

$$S(\zeta) = \begin{pmatrix} S_1(\zeta) & E_{12} & \dots & E_{1P} \\ E_{21} & S_2(\zeta) & \dots & E_{2P} \\ \vdots & \vdots & \ddots & \vdots \\ E_{P1} & E_{P2} & \dots & S_P(\zeta) \end{pmatrix}$$

- $S(\zeta)$ is distributed by blocks of rows among subdomains $i = 1, \dots, P$
- $E_{ik} \neq 0 \leftrightarrow$ subdomains i and k are adjacent
- $S_i(\zeta)$ is dense $\rightarrow$ Memory requirements...

- Apply $(B-\zeta I)^{-1} \rightarrow$ Embarrassingly parallel. Implemented by the multi-threaded version of the Pardiso library 5.0.0
- Apply $S(\zeta)^{-1} \rightarrow$ Hybrid preconditioned iterative solver

Parallel formation of the preconditioner for $S(\zeta)$ Approximate $\hat{S}(\zeta) \approx S(\zeta)$ by forming an approximation of the on-diagonal blocks of $S(\zeta)$, $\hat{S}_i(\zeta) \approx S_i(\zeta) \leftrightarrow$ memory vs. robustness

Inputs: Given $\zeta \in \mathbb{C}$, `drop-B`, `drop-S`

For $i = 1, \dots, P$**: (Embarrassingly parallel)**

1. Obtain a factorization $[\hat{L}_i, \hat{U}_i] = B_i - \zeta I$ using a drop tolerance `drop-B`
2. Form $\hat{S}_i(\zeta) = C_i - \zeta I - (\hat{U}_i^{-T}E_i)^T(\hat{L}_i^{-1}E_i)$ and drop any entry smaller than `drop-S`

End

Final step: Factorize $\hat{S}(\zeta)$ by a distributed sparse solver (MUMPS, SUPERLU_DIST)

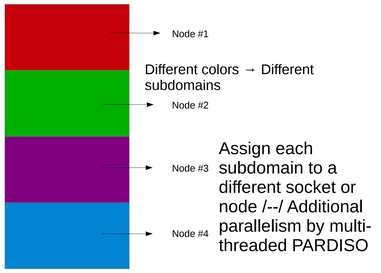
Acknowledgments

We are grateful to the MSI for providing us with computational resources. This work was supported jointly by NSF under award CCF-1505970, CCF-1510010, and the U.S. Department of Energy under award number DE-SC0008877.

Parallel implementation

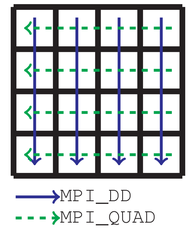
The DD framework can be implemented using a 1-D grid of processors, by assigning each subdomain to a different processor of the 1-D grid and communicating by MPI (left column). Each processor adds an extra level of OpenMP parallelism by the multi-threaded PARDISO library. FEAST offers different levels of parallelism which can be further exploited by replicating the 1-D grid (2-D grid of processors), thus reducing the computational complexity in each 1-D grid (middle and right columns).

Domain Decomposition parallelism – Basic 1-D grid of processors



Parallelize the quadrature node solutions

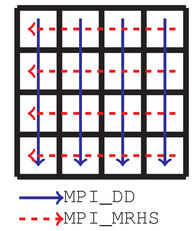
- Each 1-D grid of processors solves $(A-\zeta_j I)X_j = V$ for one or a few $\zeta_j,\ j = 1, \dots, Q_e$.



- Nearly optimal scaling if $S(\zeta)$ is formed and factorized exactly.

Parallelize the right-hand side solutions

- Each 1-D grid of processors writes $V = [V_1, \dots, V_\gamma]$ and solves $(A-\zeta_j I)X_j^{(\gamma)} = V_\gamma,\ j = 1, \dots, Q_e$.



- Better load-balancing if linear systems with $S(\zeta)$ are solved by an iterative solver.

Performance comparisons

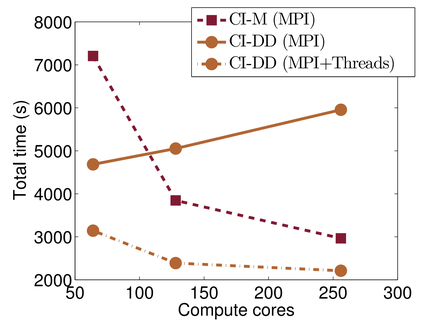
Computational system The experiments performed at the `Mesabi` Linux cluster at Minnesota Supercomputing Institute. Each MPI process was launched on a separate socket of each Intel Haswell E5-2680v3 processor. P denotes both the number of MPI processes and number of subdomains.

Table Computing all eigenpairs of a 2D discretized Laplacian of size $n = 1500^2$ inside the interval $[\alpha, \beta] = [(\lambda_{1000} + \lambda_{1001})/2, (\lambda_{1200} + \lambda_{1201})/2]$. Results are shown for a 1-D grid of processors.

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

- CI-M: FEAST+MUMPS, CI-DD: FEAST+DD
- The exact $S(\zeta_j),\ j = 1, \dots, Q_e$ was used for preconditioning.
- The domain decomposition solver is twice as fast as the standard approach.
- We repeated the above experiment exploiting a 2-D grid of processors with two and four columns of processors (512 and 1024 MPI processes, respectively). Wall-clock timings were reduced by a factor of 1.98 and 3.81, respectively.

Top figure Computing all eigenpairs of a 3D discretized Laplacian of size $n = 150^3$ inside the interval $[\alpha, \beta] = [(\lambda_{100} + \lambda_{101})/2, (\lambda_{200} + \lambda_{201})/2]$ ($N_c = 1$). Linear systems with $S(\zeta_j),\ j = 1, \dots, Nc$ were solved by the preconditioned GMRES (`drop-B`=10^{-4}, `drop-S`=10^{-2}). For the MPI+Threads version of CI-DD we used a fixed number of $P = 32$.



Bottom figure Same experiment repeated for a 2-D grid of processors using $2-4$ columns of processors, each with 256 compute cores.