SAND2022-0441R

Randomized Cholesky Preconditioning for Graph Partitioning Applications

ORISE NNSA-MSIIP Report

Heliezer JD Espinoza[*]

January 28, 2022[†]

1 Introduction/Motivation

A graph is a mathematical representation of a network; we say it consists of a set of vertices, which are connected by edges. Graphs have numerous applications in various fields, as they can model all sorts of connections, processes, or relations. For example, graphs can model intricate transit systems or the human nervous system. However, graphs that are large or complicated become difficult to analyze. This is why there is an increased interest in the area of graph partitioning, reducing the size of the graph into multiple partitions. For example, partitions of a graph representing a social network might help identify clusters of friends or colleagues. Graph partitioning is also a widely used approach to load balancing in parallel computing. The partitioning of a graph is extremely useful to decompose the graph into smaller parts and allow for easier analysis.[1]

There are different ways to solve graph partitioning problems. For this work, we focus on a spectral partitioning method which forms a partition based upon the eigenvectors of the graph Laplacian (details presented in Acer, et. al.).[2] This method uses the LOBPCG algorithm to compute these eigenvectors. LOBPCG can be accelerated by an operator called a preconditioner. For this internship, we evaluate a randomized Cholesky (rchol) preconditioner for its effectiveness on graph partitioning problems with LOBPCG. We compare it with two standard preconditioners: Jacobi and Incomplete Cholesky (ichol). This research was conducted from August to December 2021 in conjunction with Sandia National Laboratories.

2 Mathematical Background

For any given graph, the graph Laplacian, L is an $n \times n$ matrix representation of a graph, which is needed to perform computational runs on a graph. A graph Laplacian is defined as

$$L = D - A,$$

[†]Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

[*]Special thanks to my project mentors, Dr. Boman & Dr. Loe, without whom this project would not be possible.

where D is known as the degree matrix and A is the adjacency matrix. The degree matrix, D is a diagonal matrix, where each diagonal value corresponds to the number of edge connections to each vertex. The adjacency matrix (also known as the connection matrix), A, denotes an edge between vertices of that respective row and column number with a nonzero entry. A simple graph's nonzero entry will equal one as the edges are not weighted. Note that L is symmetric, positive semi-definite, and has n non-negative, real-valued eigenvalues

$$0 = \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n. \tag{1}$$

Given a square matrix W, the equation $Wx = \lambda x$ is known as the eigenvalue-eigenvector problem. In the equation, λ is the eigenvalue while x is the eigenvector. This research project utilizes the Locally Optimal Block Preconditioned Conjugate Gradient method (LOBPCG) to compute eigenvalues and eigenvectors of graph Laplacians.[3][4] LOBPCG forms a *Krylov subspace*,

$$K_y(L, z) = \text{span}\{z, Lz, L^2z, \ldots, L^{y-1}z\},$$

then uses a projection to extract approximate eigenvector solutions x_i from this subspace.[5] A corresponding approximate eigenvalue q_i can be obtained using the Rayleigh quotient:

$$q_i = \frac{||{x_i}^T L x_i||}{||{x_i}^T x_i||} \approx \lambda_i \tag{2}$$

The *relative residual*

$$r_i = ||q_i x_i - L x_i|| \text{ (where } \|x_i\| = 1) \tag{3}$$

gives a measure of the accuracy of the computed eigenvalue and eigenvector solutions.

To accelerate the convergence of LOBPCG, we apply a preconditioner to the internal solve of a linear system, which we denote as $Wx = b$. A *preconditioner* allows us to replace the original problem $Wx = b$ with an equivalent system. When W is symmetric, which L is as previously stated, we use split preconditioning to ensure that equivalent system remains symmetric. In this case, the preconditioner $M = HH^T$ is used as follows:

$$(H^{-1}WH^{-T})H^T x = (H^{-1}b).$$

For this study, three different preconditioners were applied to LOBPCG. Jacobi, a standard preconditioner, is a diagonal matrix whose entries are found by taking the reciprocals of the diagonal of W. The second preconditioner of interest, Incomplete Cholesky Factorization (ichol), is found through Cholesky Factorization of W such that $W = U^T U$, where U is an upper triangular matrix with real and positive diagonal entries, and U^T denotes the conjugate transpose of U. However, ichol differs from standard Cholesky Factorization by having "no fill", meaning that if there is a zero entry in the original matrix W ($w_{ij} = 0$), then the corresponding entry will also be zero in U ($u_{ij} = 0$). This process significantly increases the speed finding the preconditioner compared to utilizing just the standard Cholesky Factorization. The final preconditioner of interest is Randomized Cholesky Factorization (rchol). rchol approximates the Cholesky factorization for a graph Laplacian, L using a randomized algorithm.[6] As rchol is a newly developed preconditioner, we compare it to Jacobi and ichol to see how it fares in reducing LOBPCG's solve time. However it should be noted that while similar in concept, standard ichol and rchol differ in the way they fill entries in U. As previously mentioned, standard ichol has "no fill", while rchol's fill is randomized. There are ways to adjust ichol's fill by manipulating some of the algoritm's parameters, which will be briefly discussed further in Section 5, to more closely match any given rchol run's fill, but this was not done in this project.

3 Software

MATLAB was the main software tool used for this project. To complicate matters, the use of MATLAB through remote command-line-only access of Sandia workstations was necessary for many components of this project, such as properly accessing the installed rchol code. As such, in addition to MATLAB training, Linux command line knowledge was needed to securely connect to respective Sandia workstations. Lastly, understanding of Git, both through the Linux command line and through the GitHub/GitLab websites, was needed to properly collaborate with the Sandia program mentors. Learning to use these software tools comprised a significant portion of the work of this internship.

4 Results

As mentioned previously, three different types of preconditioners of interest were applied to LOBPCG to solve the eigenvalue-eigenvector problem on different test matrices W. In addition to the preconditioned systems for LOBPCG, a non-preconditioned LOBPCG was run for speed comparison's sake. The test matrices are a subset of those tested in Acer, et. al, limited to problems where the number of nonzeros is less than 250,000.[2] Most of these matrices are from Texas A&M's SuiteSparse Library with the exception of the "Brick 3D" Matrices.[7] These matrices, "Brick 3D 100" and "Brick 3D 200," were created with the Trilinos package Galeri via a uniform mesh on a 27-point stencil discretization of the standard Laplacian differential operator, where the number of elements in each mesh direction is 100 and 200, respectively.

These test matrices fall into two categories: structured matrices and unstructured matrices. The entries of structured sparse matrices have a pattern-like/formulaic relationship. For example, a finite difference matrix is a structured sparse matrix. By contrast, an unstructured matrix, such as from a social network graph, has no such relationship between its entries. As such, there can be a lot of unpredictable clustering for these matrices.

Prior to computing the graph Laplacian of our test problems, we performed preprocessing calculations by following the procedure laid out in Acer, et. al.[2] This was done by symmetrizing the matrix $(W + W^T + I)$, setting all nonzero entries equal to 1 (assuming that all edge weights are one), and finding the largest connected component. Prior to each run of LOBPCG, the graph Laplacian L of the respective test matrix W was computed. Then the respective preconditioner of each graph Lapacian was found. Note that for rchol, to avoid having a numerically singular preconditioner, we computed the randomized Cholesky factorization of a shifted matrix $L + \alpha I$, where $\alpha = .001$. Finally, L and the respective preconditioner of choice were input into LOBPCG.

The results of the LOBPCG runs on the test matrices are listed in the table below (Table 1). The variables for the table are as follows:

- *tf*: time taken to compute the factorization of the graph Laplacian L
- *tt*: the total time, including *tf*, LOBPCG took to complete a run, either by reaching the maximum number of iterations (2000) or by converging to the preset tolerance (1e$-$3).
- *nit*: number of iterations LOBPCG took to converge.
- *relres*: final relative residual norm of λ_4.

The bold entries for each matrix highlight the preconditioning option with the fastest LOBPCG solve time. Reported solve times and corresponding iteration counts are taken from the median solve time of three runs.

Test Matrix	No Prec			Jacobi				ichol				rchol			
Name	tt	nit	relres	tf	tt	nit	relres	tf	tt	nit	relres	tf	tt	nit	relres
ecology1	287	385	9.83E-04	0.03	282	313	9.79E-04	0.03	94	99	9.30E-04	**1.23**	**17**	**27**	**9.98E-04**
dielFilterV2real	388	490	9.90E-04	0.19	241	293	9.61E-04	1.47	154	144	9.67E-04	**15**	**30**	**11**	**6.88E-04**
thermal2	343	367	9.44E-04	0.05	351	400	9.62E-04	0.16	172	174	9.45E-04	**2.11**	**39**	**52**	**1.00E-03**
Bump_2911	2,525	992	1.00E-03	0.28	1,213	396	9.93E-04	1.86	467	122	9.84E-04	**74**	**164**	**13**	**6.84E-04**
100ˆ3	182	268	9.69E-04	0.07	187	276	9.31E-04	0.24	74	83	9.98E-04	**12**	**28**	**9**	**4.06E-04**
200ˆ3	3,326	899	9.54E-04	0.55	3,153	590	9.10E-04	1.90	1,197	176	8.61E-04	**289**	**489**	**9**	**7.80E-04**
hollywood	2,751	2000	1.18E-01	0.14	219	192	9.17E-04	**28.4**	**85**	**26**	**5.00E-04**	58	261	54	7.44E-04
cit-Patents	7,160	2000	6.06E-03	0.16	620	156	7.46E-04	**3.77**	**156**	**23**	**8.10E-04**	33	441	121	7.17E-04
wb-edu	17,698	2000	1.51E-01	0.57	4,945	655	7.00E-04	**4.74**	**444**	**66**	**7.63E-04**	23	7258	785	1.00E-03

Table 1: LOBPCG results for varying test matrices with no, Jacobi, ichol, and rchol preconditioning. Observed parameters are the time of the graph Laplacian computation *tf*, the total LOBPCG solve time *tt*, number of iterations LOBPCG took to converge *nit*, and final relative residual norm of λ_4 *relres*. Note that the maximum number of iterations set for all LOBPCG runs was 2000 and the preset tolerance given for the system convergence was 1e$-$3.

For many of our test matrices, rchol preconditioning gives the fastest solve time. However, rchol is not the best preconditioner for every matrix. Interestingly, ichol is the fastest LOBPCG preconditioner for the last 3 test matrices (hollywood, cit-Patents, and wb-edu). While this result is seemingly arbitrarily, the fastest LOBPCG applied preconditioner seems to correspond to whether a matrix is structured or not. The first set of test matrices are all structured matrices, while the final three are unstructured matrices. As such, these results seem to suggest that rchol is an outstanding LOBPCG preconditioner for structured matrices, while ichol is better for unstructured matrices. However, these are results are only on a relatively small matrices, in terms of the number of nonzero entries. More test matrices, especially larger unstructured matrices need to be tested through LOBPCG to get a better picture of this trend.

As previously mentioned, LOBPCG is an eigenvalue-eigenvector solver algorithm. It does this by solving for the smallest eigenvalues first. In our runs, LOBPCG was set to solve for the smallest 4 eigenvalues. In the following figures, the relative residuals for 3 out of the smallest 4 eigenvalues were plotted to illustrate how fast each preconditioned solve converges to the desired tolerance. We ignore the smallest eigenvalue λ_1 as it tends to zero ($\lambda_1 \to 0$) and does not provide any information for spectral partitioning purposes.

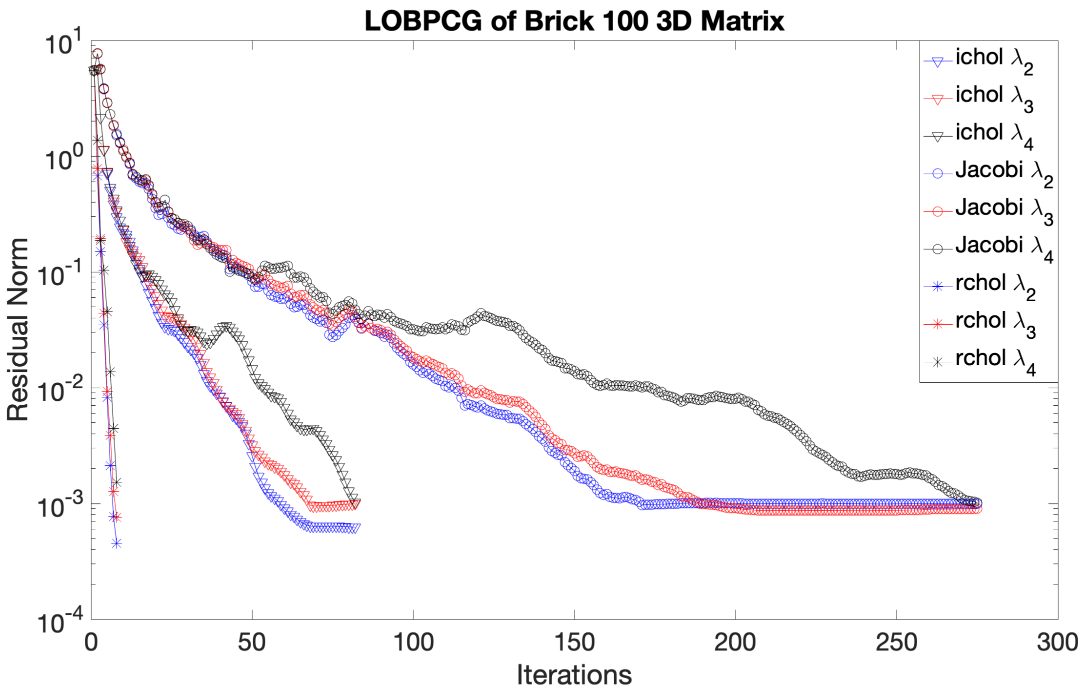

Figure 1: 100 3D Brick Matrix LOBPCG for 3 eigenvalues with varying preconditioners

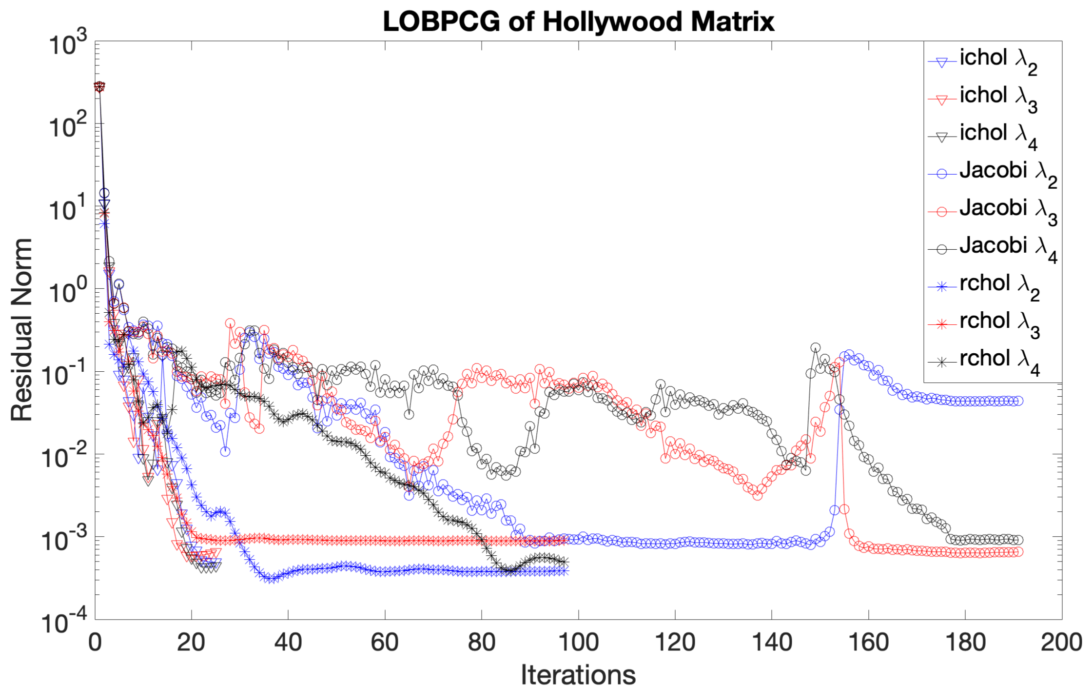

Figure 2: Hollywood LOBPCG for 3 eigenvalues with varying preconditioners

A final note on these results: while Chen, et. al. suggest permuting a matrix before performing rchol factorization, said permutations significantly slowed performance in this study.[6] More specifically, the solver would take longer to solve for the eigenvalues with a rchol for a Minimum Degree (AMD) reordered test matrix W than with no preconditioner. More often than not, it would also end up taking even more iterations. This, overall, is why rchol on an unpermuted matrix seems the most desirable, based on our results.

5 Conclusion

There were some test matrices with less than 250,000 nonzeros listed in Acer, et. al. that do not appear in Table 1.[2] Unfortunately, rchol preconditioned LOBPCG attempts to test these matrices failed due to a problem in the rchol code. Future work on this project could delve deeper the code/method to see if there is a way to correct the error being caused. In addition to uncovering the issue with rchol, there are two immediate ways to expand on this specific experiment as is:

The first is to showcase a better comparison between ichol and rchol. As presented in Chen, et. al and briefly mentioned in Section 2, it is difficult (and perhaps a bit unfair) to directly compare the ichol and rchol preconditioned systems as is. The issue here is that standard ichol has zero fill while rchol has a randomized fill. Table 4 from Chen suggests that one way to better compare these two preconditioned systems is to perform incomplete Cholesky (ichol) with threshold dropping. In threshold dropping, there is a drop tolerance that can be adjusted to increase the fill of ichol. The drop tolerances continues to be altered in this version of ichol until the ratio of fill over matrix size is approximately the ratio in an rchol run.[6]

Another area to immediately explore this solving method is to see if there is a way to allow MATLAB to run in parallel. As is, MATLAB seems to be using only one CPU core to run these iterative linear solvers. This, along with memory limitations, influenced the decision to only test matrices with a limited number of nonzeros. As such, figuring out how to utilize MATLAB parallel computing capabilities to our advantage would allow for more in depth analysis into larger and larger matrices.

While the avenues mentioned above are great examples of ways to continue this project, implementing a software interface from the rchol code to Trilinos is an immediate area of interest for future work.[8] Once this interface is created, we can verify these results with the test matrices using Sphynx graph partitioning software and Anasazi package for LOBPCG in addition to testing on larger matrices. rchol can also be tested as a preconditioner for other linear system solvers/interfaces such as for the Conjugate Gradient method in Belos.[9] Furthermore, we can use the parallelism inherent in Trilinos to test larger problems at larger scale.

On a final note, the work I was able do to through this ORISE internship has benefited me in so may ways. It not only exposed me to real-world mathematical research, especially in a national lab environment, but through this opportunity, I will be able to use this research towards my on my Master's Thesis. This opportunity has allowed me to grow tremendously in mathematics, and I now see that pursuing a career in research is a great, viable option for me. I hope to continue my work with Sandia National Laboratories as I continue to make progress towards completing my Master's Thesis, and as I continue in my academic journey.

References

[1] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz. *Recent Advances in Graph Partitioning*, volume 9220, chapter 4. Springer International Publishing, 11 2016.

[2] Seher Acer, Erik G. Boman, Christian A. Glusa, and Sivasankaran Rajamanickam. Sphynx: A parallel multi-gpu graph partitioner for distributed-memory systems. *Parallel Computing*, 106:102769, 2021.

[3] Yousef Saad. *Iterative Methods for Sparse Linear Systems.* Society for Industrial and Applied Mathematics (SIAM), January 2003.

[4] Andrew V. Knyazev. Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method. *SIAM Journal on Scientific Computing*, 23(2):517–541, 2001.

[5] Martin Gutknecht. *A Brief Introduction to Krylov Space Methods for Solving Linear Systems*, pages 53–62. Springer Berlin Heidelberg, 01 2007.

[6] Chao Chen, Tianyu Liang, and George Biros. Rchol: Randomized cholesky factorization for solving sdd linear systems. *arXiv:2011.07769v4*, 11 2020.

[7] Timothy A. Davis and Yifan Hu. The University of Florida sparse matrix collection. *ACM Trans. Math. Software*, 38(1):Art. 1, 25, 2011.

[8] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley. An overview of the trilinos project. *ACM Trans. Math. Softw.*, 31(3):397–423, September 2005.

[9] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. Nat. Bur. Standards*, 49:409–436, 1952.