Impact of Reordering on the LU Factorization Performance of Bordered Block-Diagonal Sparse Matrix

Haya Monawwar, Ahmad Ali, Hantao Cui*
School of Electrical and Computer Engineering
Oklahoma State University
Stillwater, OK 74074
haya.monawwar, ahmad.ali@okstate.edu, hcui@ieee.org

Jonathan Maack, Min Xiong

National Renewable Energy Laboratory
Golden, CO 80401 USA
jonathan.maack, min.xiong@nrel.gov

***Abstract*—Power engineers rely on computer-based simulation tools to assess grid performance and ensure security. At the core of these tools are solvers for sparse linear equations. When transformed into a bordered block-diagonal (BBD) structure, part of the sparse linear equation solving can be parallelized. This work focuses on using the Schur-complement-based method for LU factorization on BBD matrices, specifically, Jacobian matrices from large-scale systems. Our findings show that the natural ordering method outperforms the default ordering method in computational performance for each block of the BBD matrix. This observation is validated using synthetic 25k-bus and 70k-bus cases, showing a speed-up of up to 38% when using natural ordering without permutation. Additionally, the impact of the number of partitions is studied, and the result shows that computational performance improves with more, smaller partitions in the BBD matrices.**

***Index Terms*—power system dynamics, BBD matrix, equation solving, power system simulation, power flow**

I. Introduction

With the rapid development of the electric grid, power networks have grown significantly in their size. Power network needs to operate securely and reliably under the significant growing electricity demands. To understand the static and dynamic performance of the grid, power engineers heavily depend on computer-based simulation tools. It is crucial for these tools to perform computations with both accuracy and computational efficiency.

Numerical solvers of linear equations are ubiquitous in various power system simulation tools. One example of dynamic analysis of power systems is real-time or electromagnetic transient (EMT) simulation. Such analyses can be performed using various simulation tools, such as PSCAD [1]. These tools are computationally intense due to solving a large number of equations (called nodal equations), in the order of hundreds of thousands for step sizes of microseconds [2]. Moreover, in a large interconnected system, the simulation models cover a wide range of time scales and undergo numerous discrete transitions [3]. Additionally, solving nodal equations can take up 80% to 97% of the computation time in large-scale EMT simulations [4]. As the underlying numerical solvers from the scientific computing community are extremely optimized, the focus is placed on the structure of sparse matrices to improve the computational performance of power system problems.

The matrices involved in power system simulations are derived from the grid model. Given that power system matrices are typically sparse, this characteristic has been widely exploited to reduce computational time. Among the various sparse structures, the bordered block diagonal (BBD) structure has been gaining traction in recent research. A BBD structured matrix is defined by its sparse composition, featuring only block matrices on the main diagonal, a right border, and a bottom border of block matrices. The standard linear equation $\mathbf{Ax} = \mathbf{b}$ using a BBD matrix for $\mathbf{A}$ is given as

$$\begin{bmatrix} A_{11} & 0 & 0 & 0 & A_{1n} \\ 0 & A_{22} & 0 & 0 & A_{2n} \\ 0 & 0 & \ddots & 0 & \vdots \\ 0 & 0 & 0 & A_{mm} & A_{mn} \\ A_{n1} & A_{n2} & A_{n3} & A_{nm} & A_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \\ b_n \end{bmatrix} \quad (1)$$

where $\mathbf{A}$ is a square sparse matrix, $\mathbf{x}$ and $\mathbf{b}$ are dense vectors, n and m are the indices of row and column blocks, and $n = m+1$ [5]. Such a matrix can be obtained by permutating a sparse matrix, such as the conductance matrix or, more generally, a Jacobian matrix. Using BBD matrices, the linear equations can be solved in a decomposed manner while processing the numerous small diagonal blocks and their Schur complement [6] separately.

To transform a square matrix into a BBD-type, multiple methods are available, such as graph partitioning using the METIS package [7]. Once the BBD structure is established, Schur-complement-based LU-factorization followed by forward and backward reduction can be used for solving the linear equations. Such an approach using BBD matrices sees applications wherever sparse linear equations are solved, including power flow calculation [8], time domain simulation [9], [10], and EMT simulation.

This work aims to understand the most efficient approach for solving sparse linear equations where $\mathbf{A}$ is a BBD matrix.

This work leverages the BBD matrix conversion algorithm implemented in ParaEMT [11], a Python-based open-source tool focused on EMT simulations. More specifically, this work accelerates the Schur-complement based LU algorithm for linear equation solving after the BBD conversion. We use the power flow Jacobian matrices from large-scale systems, convert them to BBD, and employ the sparse solver routine. When we apply the solver routine to each block, namely, using the Approximate Minimum Degree Column (COLAMD) for ordering and UMFPACK [12] for solving, the total solution time is greater than solving the original, non-BBD matrix.

This paper presents a new understanding of how permutation impacts the LU factorization time when working with the BBD matrix. Our contribution is that a shorter computational time can be achieved by turning off the default AMD permutation. This is the opposite of the general notion, and an explanation of matrix structure and density is presented. The main contribution to the power system community is that the proposed improvement can boost computational efficiency over the existing BBD matrix solver, thereby enhancing performance in large-scale simulations and real-time applications. Case studies using the Synthetic 25k- and 70k-bus systems validate this observation. In addition, the impact of the number of blocks in the BBD structure is also investigated.

This paper is organized as follows: Section II explains the basics of BBD formation and presents the computational bottleneck in the existing algorithm. Section III presents the case studies and discussions, and Section IV describes the reached conclusions and recommendations for future work.

II. Methodology

A. Converting Matrices into BBD Structure

As seen from (1), BBD is overall a sparse structure. However, depending on the input matrix, the diagonal blocks in the BBD matrix are likely to have a higher *density* (represented as the percentage of non-zero elements over all elements) than the initial input matrix. When using BBD matrices, each diagonal block can be processed in parallel asynchronously. It is worth noting that A_{nn}, the corner block, is considered separately from the other diagonal blocks.

The formation of a BBD structure begins with graph partitioning using the METIS package. The user will specify the desired number of partitions as a parameter. The total number of partitions is one less than the number of blocks on the main diagonal since the corner block is separate. Choosing the correct number of partitions is crucial. METIS can only partition a graph into partitions if it meets the conditions of minimal edge-cut and approximately equal nodes in each partition [7]. If these conditions are not met, the requested partitioning and BBD formation will fail, since the main diagonal cannot contain zero blocks. Generally, larger matrices are more likely to be separated into a modest number of partitions than smaller cases.

After graph partitioning, nodes are categorized into common nodes (those with edges to multiple node clusters) and uncommon nodes (with edges only to nearby nodes). Next, common and uncommon nodes are separated to create subgraphs for each partition. Each subgraph undergoes nested dissection and permutation ordering. The original input matrix is reordered based on these permutations. Finally, a BBD object is created and populated with diagonal, corner, and border matrices. During this step, the nodes of each block are identified using the previously created subgraphs. The whole process is visualized in Fig. 1.

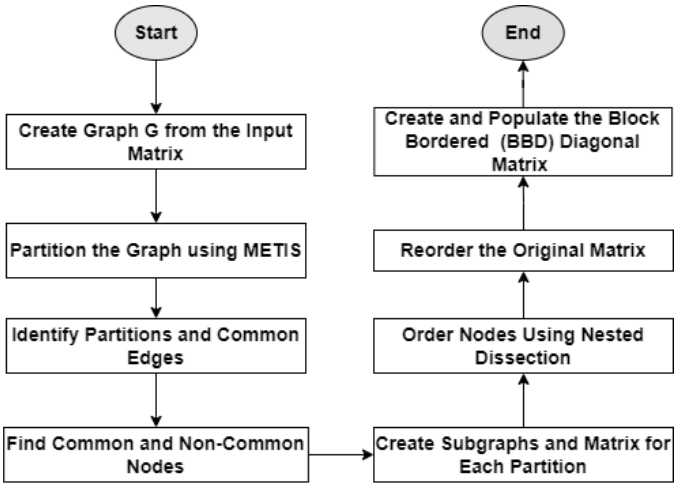

Fig. 1. The BBD matrix formation process as in the BBD solver [1]

A practical issue with this algorithm is that the resulting BBD matrix may not always be easily identifiable as a clear-cut BBD structure. This issue largely stems from the characteristics of the input given to the algorithm. Smaller matrices are less likely to produce visually identifiable and parallel processing-friendly BBD matrices compared to the BBD forms of larger matrices.

B. Bottleneck in the Existing BBD Solver

The foundation for solving a BBD sparse linear system is the LU factorization based on the Schur complement. That is, the matrix $\mathbf{A}$ is permutated by $\mathbf{P}$ and decomposed into $\mathbf{L}$ and $\mathbf{U}$, namely, $\mathbf{PA} = \mathbf{LU}$. Different procedures are followed to obtain the LU decomposition of the internal, border, and corner submatrices. Specifically, the decomposition of the corner block relies on the summation of $\mathbf{L}$ and $\mathbf{U}$ factors from the internal submatrices. Calculating the $\mathbf{L}$ and $\mathbf{U}$ matrices for the corner block is inherently a serial task, necessitating the insertion of a synchronization point [11]. Once $\mathbf{LU}$ factors have been obtained, the equation $\mathbf{Ly} = \mathbf{b}$ is solved for $\mathbf{y}$. During this stage, another synchronization point is required because the calculation of $\mathbf{y_n}$ for the corner matrix depends on the solution $\mathbf{y}$ from the preceding internal submatrices. Finally, backward substitution is performed to obtain $\mathbf{x}$ using the equation $Ux = y$.

To optimize an algorithm, it is important to time different parts of the algorithm and find out the parts taking up the most computation time. It is found that the percentage of the total computational time between the forward and backward

substitution stages is an average of 80% and 20%, respectively, for the Synthetic 70k-, 25k-, and 10k-bus systems [13]. Upon timing the forward reduction method, the block-by-block solution of the internal submatrices to solve $Ly = b$ took up most of the computational time for this part. Since the main advantage of using a BBD structure is to be able to process smaller-sized blocks in parallel, block-by-block computation can not be modified. However, the permutation technique employed by the sparse solver can be adjusted.

By default, the sparse linear solver in SciPy [14] uses COLAMD with UMFPACK. The BBD solver calls the SciPy solver each sparse linear block. COLAMD sets a density threshold of 50% for rows and columns to be eliminated before the reordering process begins. Columns that do not satisfy the criteria are later placed last in the output column ordering. The goal of this reordering strategy is to minimize fill-ins during factorization by processing the sparsest columns first and positioning denser columns at the end of the matrix to be solved last [15], [16]. Theoretically, this technique enhances equation-solving efficiency. Moreover, in works like [17], a custom pre-ordering strategy is devised to convert a matrix into BBD form. It uses the approximate minimum degree (AMD) ordering to reduce fill-ins in matrix $\mathbf{A}$ during LU factorization. The equation $\mathbf{Ax} = \mathbf{b}$ is then solved by treating the BBD matrix as one unit. However, this limits the use of parallel computing.

In our BBD solver, when solving for each diagonal block or internal submatrix, it is observed that the default permutation strategy requires significantly more computational time than without any reordering. In fact, the partitioning can be seen as an ordering strategy in place of AMD. Hence, this work compares the default AMD-based permutation with a no permutation (also known as "natural ordering") on the coefficient matrix $\mathbf{A}$ as an improvement to the existing BBD solver.

III. Results and Discussion

For this section, tests are performed on the 25,000-bus Synthetic US Northeast/Mid-Atlantic model and the 70,000-bus Synthetic Eastern US models from MATPOWER [13]. The differences in computational time are explored, and the impact of the number of partitions on the speed-up is investigated. The input matrices for $\mathbf{A}$ are the Jacobian matrix of the Newton power flow obtained from ANDES [10].

A. Case Study I: Comparison of COLAMD and Natural Ordering

This section presents the computational time, with and without matrix reordering, for a) solving the entire BBD matrix as a single unit, and b) solving each diagonal (internal) block within the BBD matrix.

When formulating the BBD matrix, the number of partitions is arbitrarily set to five. Consequently, the BBD matrix for each test case comprises five internal blocks to solve. The main diagonal thus contains five diagonal blocks in addition to one corner block, which is solved independently from the other diagonal/internal blocks. This structure is illustrated in Fig. 2 and Fig. 3 for the 25,000 bus and 70,000 bus cases, respectively.

Initially, the performance of the sparse linear solver is evaluated on the entire BBD matrix as one unit, comparing default permutation and natural ordering (no permutation) scenarios. It is observed that the solver with COLAMD permutation required less than half the computation time than without permutation. This finding aligns with theoretical expectations as discussed in [15], [16]. However, when solving each internal block individually, using natural ordering reduced the overall computation time for forward and backward substitution by 15.6% for the 25,000 bus system and 24.9% for the 70,000 bus system. These results are summarized in Table I.

TABLE I
Computational Time (ms) of the BBD Solver and the Sparse Linear Solver with and without Permutation for BBD Matrices with 5 Partitions

Case	BBD Solver		SciPy solver	
	COLAMD	Natural	COLAMD	Natural
25,000	271	103	122	363
70,000	1,980	5,74	764	2,380

To explore this contradiction, the matrix structure is studied to understand how each internal block differs from the entire BBD structure. Given that the vector $\mathbf{b}$ is a simple 1-D vector of ones, its impact on computation time is negligible. However, the density of rows in the coefficient matrix can result in a high number of non-zeros (NNZ) due to fill-ins from factorization, and its dense columns can lead to increased computation time if the default permutation style is used [15], [16].

Significant differences are identified in the percentage of non-zeros. The overall percentage of non-zeros in the entire BBD matrix is at least one order of magnitude lower than that in the internal blocks. Consequently, the percentage of dense rows and columns is also higher by the same order of magnitude in every block compared to the entire BBD matrix. This observation can be confirmed in the statistics shown in Tables II, III, and IV. The matrices are visualized in Fig. 2 and Fig. 3, which highlight the denser rows and columns in the internal BBD blocks. Although only visuals for one block each are provided due to space constraints, this finding is consistent across all internal blocks.

The dense rows in these internal blocks lead to a higher overall percentage of non-zeros, including fill-ins, during factorization, leading to increased computation time. Therefore, avoiding further reordering for the internal blocks of a BBD matrix can help reduce the computational time.

A question about how density is defined may arise here. In [15], [16], the row or column density threshold is set to 50% of the row or column filled. The ordering algorithm will treat the dense rows and columns separately to improve efficiency. In the synthetic cases considered in this paper, both the initial input and the BBD output have 0.012% and 0.004% non-zeros in the 25k and the 70k systems, respectively. However, this threshold appears high for our applications; some rows and

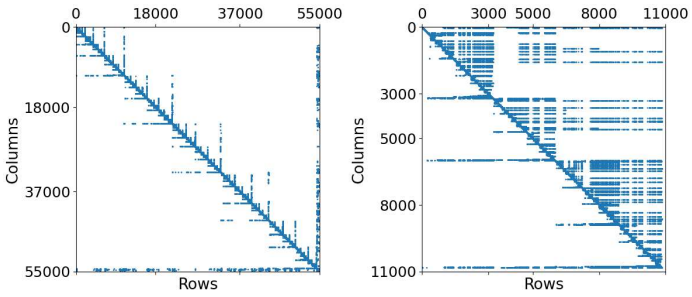

Fig. 2. BBD Matrix of the 25,000 bus system with 5 partitions, and its first block.

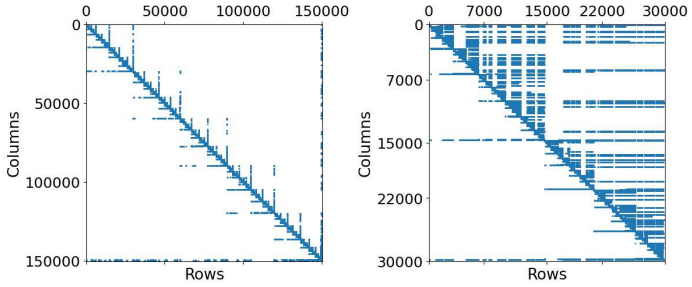

Fig. 3. BBD Matrix of the 70,000 bus system with 5 partitions, and its first block.

columns are below this threshold but should have been skipped for reordering to avoid creating fill-ins.

To determine the density of rows and columns in the BBD matrices, a ratio-based density metric is used. First, the maximum NNZ in any row or column is identified. Then, the number of rows or columns containing at least half of this maximum NNZ is counted. This count is divided by the total number of rows or columns, yielding the percentage of rows or columns considered *dense*.

Table III and Table IV show the NNZ count and density percentage of **LU** factors for the entire BBD unit and its blocks, with and without permutation. The data in Tables II, III, and IV demonstrate that more dense rows and columns in the internal blocks of the BBD matrix can increase overall computation time when applying the default reordering strategy to matrix **A**.

TABLE II
NNZ COMPARISON WITH PERMUTATION AND NATURAL ORDERING FOR THE BBD MATRIX WITH 5 PARTITIONS

Case	Initial % of Non-Zeros	% of Dense Rows / Cols	L + U NNZ (COLAMD)	L + U NNZ (Natural)
25,000	0.012	0.221 / 0.221	1,578,607	2,625,751
70,000	0.004	0.299 / 0.299	5,312,783	10,746,064

B. Case Study II: Investigating the Effect of Number of Partitions on the Proposed Improvement

In this section, we investigate the impact of the number of partitions in the BBD matrix on the computational speed-up achieved by our proposed strategy. In the previous case study, the number of partitions for generating the BBD matrix is arbitrarily set to five. To further explore this effect, we have increased the number of partitions to fifteen, also chosen arbitrarily. This analysis aims to determine whether the observed speed-up in computation time is consistent across different partition counts, thereby validating the robustness of our strategy.

TABLE III
NNZ COMPARISON OF L AND U FACTORS OF EACH BLOCK IN THE 25K BUS CASE WITH 5 PARTITIONS

Block #	Initial % of Non-Zeros	% of Dense Rows / Cols	L + U NNZ (COLAMD)	L + U NNZ (Natural)
Block 1	0.127	6.259 / 0.387	328,229	160,273
Block 2	0.147	12.619 / 0.385	321,050	187,057
Block 3	0.117	10.589 / 0.137	285,446	150,539
Block 4	0.165	9.227 / 0.638	409,316	203,383
Block 5	0.185	6.617 / 0.268	470,553	227,725

TABLE IV
NNZ COMPARISON OF L AND U FACTORS OF EACH BLOCK IN THE 70K BUS CASE WITH 5 PARTITIONS

Block #	Initial % of Non-Zeros	% of Dense Rows / Cols	L + U NNZ (COLAMD)	L + U NNZ (Natural)
Block 1	0.088	11.172 / 0.198	2,120,757	1,442,187
Block 2	0.080	10.345 / 0.151	1,501,449	841,769
Block 3	0.076	6.536 / 0.161	2,330,490	709,112
Block 4	0.094	12.625 / 0.233	1,874,839	882,441
Block 5	0.077	10.237 / 0.359	1,844,415	715,202

A larger number of partitions results in smaller internal blocks within the BBD matrix. Consequently, the impact of no permutation may be negligible for these smaller matrices. However, when BBD generation is applied to the 25k and 70k bus systems with an increased number of partitions, the speed-up of our strategy is maximized. To further validate this, we repeated the tests similar to those presented in Table I, comparing the performance of the current BBD solver and the sparse linear solver with both, the default permutation and no permutation. The results showed a maximized speed-up when using a BBD matrix with a higher number of partitions - 38% for the 70k bus system and 16.3% for the 25k bus system. Detailed computational times for each strategy are provided in Table V.

To verify our findings from Case Study I, matrix structures are studied once more and the percentage of dense rows and columns is found, similar to Table II. The percentage of dense rows and dense columns and the NNZ with permutation and without permutation in the **LU** factors for both the cases with 15 partitions are summarized in Table VI.

TABLE V
COMPUTATIONAL TIME (MS) OF THE BBD SOLVER AND THE SPARSE LINEAR SOLVER WITH AND WITHOUT PERMUTATION FOR BBD MATRICES WITH 15 PARTITIONS

Case	BBD Solver		SciPy solver	
	Permutation	No Permutation	Permutation	No Permutation
25,000	156	108	937	129
70,000	670	443	715	4770

TABLE VI
NNZ COMPARISON WITH PERMUTATION AND NATURAL ORDERING FOR THE BBD MATRIX WITH 15 PARTITIONS

Case	Initial % of Non-Zeros	% of Dense Rows / Cols	L + U NNZ (COLAMD)	L + U NNZ (Natural)
25,000	0.0118	0.221 / 0.221	1,593,212	3,276,857
70,000	0.004	0.299 / 0.299	5,432,764	11,970,868

Tables VII and VIII show a reduction in NNZ in the **LU** factors compared to cases with five partitions. However, the high percentage of dense rows and columns within these partitions leads to increased fill-ins when applying the default reordering strategy during factorization, resulting in higher overall computation time. Since there are more partitions in this case, findings in Tables VII and VIII are presented only for the first 5 blocks, though verified for all 15 blocks in both cases.

Additionally, this phenomenon can be visually corroborated. Fig. 4 and Fig. 5 depict the BBD matrices of the 25k and 70k bus systems with 15 partitions. These figures include a detailed view of block 5 from each BBD matrix, illustrating the presence of dense rows and columns. The visual representation supports the numerical data, highlighting the impact of the default permutation strategy in the traditional solver on dense matrix structures with respect to computational efficiency.

The discussion above demonstrates that rows and columns with less than the defined density can negate the benefits of using the COLAMD permutation. By the placement of these non-zeros, it is not necessary that a row or column must be greater than or equal to the defined density criteria for the algorithm to identify it. Therefore, densities lower than the threshold must also be addressed to minimize fill-ins during factorization when using this permutation. To achieve this, the previously defined density metric can be utilized. This adaptive density threshold can be tailored to the specific case under consideration, as most power system matrices are significantly less dense than 50%. This approach differs from the heuristic density threshold used in [15], [16], indicating that the threshold may need to be adjusted for each specific case when applying COLAMD.

TABLE VII
NNZ COMPARISON OF L AND U FACTORS OF FIRST 5 BLOCKS IN THE 25K BUS SYSTEM WITH 15 PARTITIONS

Block #	Initial % of Non-Zeros	% of Dense Rows / Cols	L + U NNZ (COLAMD)	L + U NNZ (Natural)
Block 1	0.324	5.519 / 0.781	70,269	45,287
Block 2	0.382	12.619 / 0.899	74,265	51,851
Block 3	0.319	7.710 / 0.822	84,475	46,595
Block 4	0.237	9.293 / 0.696	47,129	34,250
Block 5	0.244	9.983 / 0.583	50,510	35,177

TABLE VIII
NNZ COMPARISON OF L AND U FACTORS OF FIRST 5 BLOCKS IN THE 70K BUS SYSTEM WITH 15 PARTITIONS

Block #	Initial % of Non-Zeros	% of Dense Rows / Cols	L + U NNZ (COLAMD)	L + U NNZ (Natural)
Block 1	0.123	7.912 / 0.385	238,991	130,545
Block 2	0.134	9.899 / 0.357	243,405	144,523
Block 3	0.129	9.181 / 0.365	244,351	135,163
Block 4	0.157	10.399 / 0.756	248,176	160,246
Block 5	0.182	11.078 / 0.384	343,697	187,717

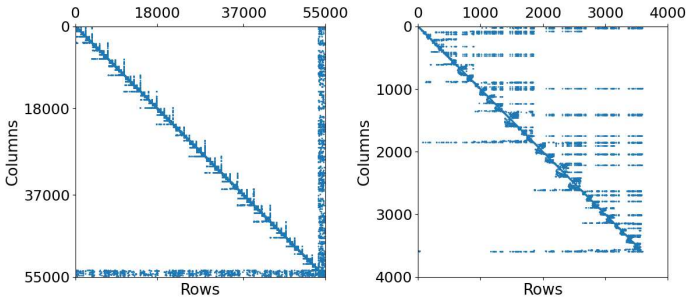

Fig. 4. BBD Matrix of the 25,000 bus system with 15 partitions, and its fifth block.

IV. CONCLUSION

This paper investigates the impact of reordering on the LU factorization performance of BBD matrices, which has great potential for accelerating power system simulations. We presented the workflow to convert a power system sparse matrix into the BBD structure, followed by the Schur complement-based LU decomposition to solve the equation $\mathbf{Ax} = \mathbf{b}$.

It is found that, for each block of a BBD matrix, applying the default COLAMD ordering before LU factorization may increase the number of fill-ins and thus reduce the computation efficiency. Instead, using the natural ordering for each block is more efficient by eliminating the reordering time and saving the factorization time. This approach is validated through two case studies on large-scale systems. Furthermore, reducing the partition size can further improve the computational speed.

Further research could involve testing larger systems with an increased number of partitions in the BBD matrix, which may further maximize computational efficiency. Moreover, a similar hypothesis can be tested with other permutation strategies.

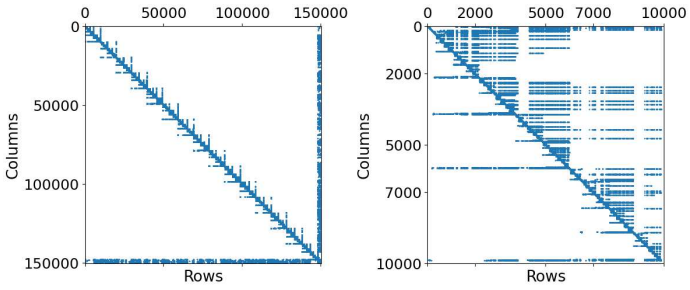

Fig. 5. BBD Matrix of the 70,000 bus system with 15 partitions, and its fifth block.

ACKNOWLEDGEMENT

The authors would like to acknowledge Dr. Jin Tan, Dr. Andy Hoke, and Dr. Deepthi Vaidhynathan at NREL for the inputs related to ParaEMT.

This work is sponsored by the National Science Foundation, Award ECCS-2226826.

REFERENCES

[1] "Home | PSCAD," https://www.pscad.com/.

[2] X. Xiong, Y. Feng, and L. Zhao, "Power system security assessment based on real-time electromagnetic-electromechanical transient hybrid simulation," in *2021 International Conference on Advanced Electrical Equipment and Reliable Operation (AEERO)*, Oct. 2021, pp. 1–5.

[3] D. Fabozzi, A. S. Chieh, B. Haut, and T. Van Cutsem, "Accelerated and Localized Newton Schemes for Faster Dynamic Simulation of Large Power Systems," *IEEE Transactions on Power Systems*, vol. 28, no. 4, pp. 4936–4947, Nov. 2013.

[4] L. Zhang, B. Wang, X. Zheng, W. Shi, P. R. Kumar, and L. Xie, "A Hierarchical Low-Rank Approximation Based Network Solver for EMT Simulation," *IEEE Transactions on Power Delivery*, vol. 36, no. 1, pp. 280–288, Feb. 2021.

[5] S. Fan, H. Ding, A. Kariyawasam, and A. M. Gole, "Parallel Electromagnetic Transients Simulation with Shared Memory Architecture Computers," *IEEE Transactions on Power Delivery*, vol. 33, no. 1, pp. 239–247, Feb. 2018.

[6] F. Zhang, Ed., *The Schur Complement and Its Applications*, ser. Numerical Methods and Algorithms. New York: Springer-Verlag, 2005, vol. 4.

[7] G. Karypis and V. Kumar, "METIS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices," 1997.

[8] X. Wang, S. G. Ziavras, C. Nwankpa, J. Johnson, and P. Nagvajara, "Parallel solution of Newton's power flow equations on configurable chips," *International Journal of Electrical Power & Energy Systems*, vol. 29, no. 5, pp. 422–431, Jun. 2007.

[9] H. Cui and F. Li, "ANDES: A Python-Based Cyber-Physical Power System Simulation Tool," in *2018 North American Power Symposium (NAPS)*, Sep. 2018, pp. 1–6.

[10] H. Cui, F. Li, and K. Tomsovic, "Hybrid symbolic-numeric framework for power system modeling and analysis," *IEEE Transactions on Power Systems*, vol. 36, no. 2, pp. 1373–1384, 2021.

[11] M. Xiong, B. Wang, D. Vaidhynathan, J. Maack, M. J. Reynolds, A. Hoke, K. Sun, and J. Tan, "ParaEMT: An Open Source, Parallelizable, and HPC-Compatible EMT Simulator for Large-Scale IBR-Rich Power Grids," *IEEE Transactions on Power Delivery*, vol. 39, no. 2, pp. 911–921, Apr. 2024.

[12] T. A. Davis, "Algorithm 832: UMFPACK V4.3—an unsymmetric-pattern multifrontal method," *ACM Trans. Math. Softw.*, vol. 30, no. 2, pp. 196–199, Jun. 2004.

[13] R. D. Zimmerman, C. E. Murillo-Sánchez, and R. J. Thomas, "MATPOWER: Steady-State Operations, Planning, and Analysis Tools for Power Systems Research and Education," *IEEE Transactions on Power Systems*, vol. 26, no. 1, pp. 12–19, Feb. 2011.

[14] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, and P. van Mulbregt, "SciPy 1.0: Fundamental algorithms for scientific computing in Python," *Nature Methods*, vol. 17, no. 3, pp. 261–272, Mar. 2020.

[15] T. A. Davis, J. R. Gilbert, S. I. Larimore, and E. G. Ng, "A column approximate minimum degree ordering algorithm," *ACM Trans. Math. Softw.*, vol. 30, no. 3, pp. 353–376, Sep. 2004.

[16] ——, "Algorithm 836: Colamd, a column approximate minimum degree ordering algorithm," *ACM Trans. Math. Softw.*, vol. 30, no. 3, p. 377–380, sep 2004. [Online]. Available: https://doi.org/10.1145/1024074.1024080

[17] L. Qian, D. Zhou, X. Zeng, F. Yang, and S. Wang, "A parallel sparse linear system solver for large-scale circuit simulation based on Schur Complement," in *2013 IEEE 10th International Conference on ASIC*, Oct. 2013, pp. 1–4.