Parallel Banded Linear System Solver using Cyclic Reduction Algorithm with MPI

Yusuf Talha Şahin
Middle East Technical University
Department of Computer Engineering
Ankara, Turkey
e238088@metu.edu.tr

Abdullah Burkan Bereketoğlu
Middle East Technical University
Department of Computer Engineering
Ankara, Turkey
e235517@metu.edu.tr

Mustafa Mert Köse
Middle East Technical University
Department of Computer Engineering
Ankara, Turkey
e238170@metu.edu.tr

ABSTRACT

This work focuses on the development and evaluation of a structured special banded linear system, tridiagonal linear system. By developing a parallel tridiagonal linear system solver using Cyclic Reduction (CR) algorithm, implemented with Message Passing Interface (MPI). Banded linear systems are prevalent and in core of various scientific and engineering applications, particularly in solving partial differential equations (PDEs) and physics driven simulations. The CR algorithm, known for its efficiency in solving structured linear systems, is adapted to exploit parallelism in distributed computing environments of matrix's structural properties, making it suitable for high-performance computing (HPC) applications, in a scalable manner. In the study performance metrics such as execution time, speedup, and residual error are compared to standardized solvers, and analysis on the advantages in HPC application.

KEYWORDS

MPI, cyclic reduction, parallel computing, banded linear systems, distributed algorithms, high-performance computing

1 INTRODUCTION

The study focuses on the development, implementation and analysis of the performance of parallel cyclic reduction algorithm with respect to regular serial and parallel linear system solvers. Here the goal is to make measurements for solving tridiagonal linear systems. Banded linear systems are frequently appeared in computational physics, fluid dynamics, and structural analysis, which require efficient and scalable solvers for large-scale applications. The goal of the study is to evaluate and compare the efficiency and scalability of parallel cyclic reduction to traditional linear algebra solvers, particularly LAPACK's specialized tridiagonal solver (DGTSV) and it's general-purpose dense linear system solver (DGESV). For the analysis of the study, both sequential and parallel versions of the cyclic reduction algorithm are developed and implemented. The parallel implementation, MPI (Message Passing Interface) is used to divide the computational workload among multiple processes. Furthermore, for the test of standard and developed solvers, a custom tridiagonal matrix of varying size generation tool is built. These matrices are designed to reflect real-world scenarios, including structured matrices with specific patterns and completely random valued matrices.

Moreover, to measure the performance of CR implementation and benchmark the design, a comparison against standardized LAPACK's DGTSV and DGESV solvers held. Moreover, key metrics such as execution time and residual error to evaluate their effectiveness are tracked. Moreover, the results are reported visually to provide insights into its applicability and to show performances and speedup with parallelization. Experimentation is held by automated scripts running on Greyfurt HPC cluster using SLURM to run on parallel, with the set up provided, at large matrices the solver scales tested. The results demonstrated that the parallel CR algorithm scales effectively, especially for large systems, while maintaining accuracy.

In summary, this paper presents a parallel implementation of the CR algorithm using MPI, focusing on parallel memory environments, and its efficiency analysis on key metrics. The primary objectives are:

- To analyze the computational efficiency and scalability of the parallel CR algorithm.
- To benchmark its performance against LAPACK routines such as DGTSV and DGESV.
- To provide insights into its applicability to HPC environments.

2 BACKGROUND AND RELATED WORK

2.1 Thomas Algorithm (Tridiagonal Matrix Algorithm)

The Thomas algorithm is a simplified and efficient method for solving tridiagonal systems. It works by first reducing the system to an upper triangular form (forward elimination) and then solving for the unknowns (backward substitution).

Advantages.

- Extremely efficient for sequential execution, with a computational complexity of $O(n)$.
- Requires minimal memory, working directly on the three diagonals of the matrix.

Limitations.

- Not suitable for parallel computing, so it's less useful for large-scale systems or GPUs.
- Relies on certain mathematical conditions (like diagonal dominance) to ensure numerical stability. [1]

2.2 Cyclic Reduction (Sequential)

Cyclic reduction is a divide-and-conquer method introduced in the 1960s. It breaks a tridiagonal system into smaller subsystems, solving them recursively until reaching a manageable size. This method

shines when the matrix size is a power of two and is often used in numerical simulations, like solving partial differential equations.

Advantages.

- Has a opeartion count complexity of $O(n)$, making it faster than Gaussian elimination for large systems, since it has smaller constant factor.
- Can be adapted for parallel execution since its reduction steps are independent.

Limitations.

- The sequential version isn't as fast as the Thomas algorithm for smaller systems.
- Managing boundary conditions and intermediate steps can be tricky to ensure stability.

2.3 Multigrid Methods

Multigrid methods are advanced iterative techniques that use a hierarchy of grids to speed up convergence. They are especially effective for large sparse systems, such as those arising from discretizing partial differential equations. [4]

Advantages.

- Delivers extremely fast convergence for well-behaved systems.
- Highly scalable and well-suited for parallel computing.

Limitations.

- These methods are complex to implement and require careful tuning.
- Setting them up can be resource-intensive.

2.4 Analysis of Existing Methods: Sweet's Parallel Variant of the Cyclic Reduction Algorithm

In 1988, Sweet [9] addressed one of the biggest limitations of the Buneman [5] version of cyclic reduction. The standard algorithm is highly parallelizable, although it becomes increasingly serial in nature. Thus, it operates in one form for the first few reductions at the start, which then mandates the rest to be done serially. Such a feature significantly limits the possible parallelization of the algorithm.

The Buneman version applies to block tridiagonal systems of the form:

$$AV_j - V_{j-1} + AV_{j+1} = f_j, \quad j = 1 \to N, \quad V_0 = V_{N+1} = 0,$$

where A is a square tridiagonal matrix of order M, and $N = 2^k + 1$. The reduction step produces a set of block tridiagonal systems where each contains approximately half of the unknowns with polynomial matrices defined as:

$$A_r = (A_{r-1})^2 - 2I.$$

The substitution step takes the increasingly reduced systems a step back to generate the unknowns for the original system.

Limitations in Parallel Environments.

- **Collapsing Parallelism:** At step r of reduction, there exist $2^{k+1-r} - 1$ parallel solutions, but with additional work per solution as it reduces.
- **Serial Dependency:** Each system needs tridiagonal solves, which creates a natural problem in accomplishing this in parallel.

Sweet proposed solving these polynomial systems differently, using partial fraction expansion:

$$\frac{P(x)}{Q(x)} = \sum_{i=1}^{n} \frac{a_i}{x - \lambda_i},$$

where $P(x)$ and $Q(x)$ are relatively prime polynomials, and $Q(x)$ has n distinct roots λ_i. This approach simplifies recursive multiplication into independent addition, enabling parallel execution.

Performance of Sweet's Parallel Variant. The reduction step at level r runs $2^{k+1-r} - 1$ independent systems, with each requiring 2^{r-1} processors. This ensures that parallelization is maintained throughout, with component-wise vector summation and binary tree summation used for efficiency. Sweet's parallel variant runs in $O(\log M \cdot \log N + \log^2 N)$ using $M \cdot N$ processors, compared to the sequential Buneman version running in $O(M \cdot N \cdot \log N)$. Empirical tests on the CRAY-1A showed a 4.2x speedup for data sets where $M = N = 63$.

Implementation Challenges.

- Effective storage of the matrix using LAPACK's banded storage scheme to minimize memory usage.
- Balancing communication costs with dynamic processor assignments to ensure efficiency.
- Dependence on power-of-two dimensions and large overhead for small problems.

Sweet's approach demonstrated how mathematical restructuring could enhance parallel efficiency, making it a foundational contribution to parallel algorithms for tridiagonal systems.

Tridiagonal linear systems are a subclass of banded systems, for many numerical methods, it is critical to resolve them efficiently. Solving tridiagonal linear systems is a fundamental problem in numerical linear algebra, and several well-established methods have been developed over the years. Some of the well established methods can be defined as Gaussian elimination or LU decomposition, these methods are limited in parallel performance due to their sequential dependencies. Furthermore, these methods vary in their computational complexity, scalability, and suitability for parallelization. In the following, we discuss the most commonly used existing methods for solving tridiagonal systems, which serve as benchmarks and references for this project.

2.5 Existing Solvers (LAPACK Routines)

DGTSV: A specialized LAPACK routine designed to solve tridiagonal systems of linear equations using Gaussian elimination with partial pivoting. It is highly optimized for sequential tasks and is a go-to option for smaller or moderately large systems [2].
Advantages:

- Offers high accuracy due to partial pivoting.

- Work efficiently for small- to medium-sized problems.
- Easy to use and integrates well with numerical libraries.

Limitations:

- Does not scale well for very large systems or distributed/parallel computing setups because it's not parallelizable.
- While the computational cost grows linearly with the system size, the constant factors can make it slower for large systems.

DGESV: A general-purpose LAPACK routine for dense matrices. Despite its versatility, it incurs significant computational overhead for banded systems [2].
Advantages:

- Versatile and works for any dense system, not just tridiagonal ones.
- Reliable and well-tested for a wide range of applications.

Limitations:

- Not ideal for tridiagonal systems due to extra computations on zero elements.
- Uses more memory and is computationally expensive, especially for large problems.
- It's not optimized for parallelizing tridiagonal systems.

2.6 Cyclic Reduction for Solving Linear Systems

The Cyclic Reduction (CR) algorithm, first introduced by G. H. Golub and R. W. Hockney in the mid-1960s, is a highly efficient technique for solving structured linear systems, particularly block tridiagonal systems, and block Toeplitz matrices. These systems frequently arise in numerical simulations, such as finite difference methods for solving partial differential equations (PDEs) like the Poisson equation. By leveraging the structural properties of the underlying matrix, CR reduces computational complexity through iterative elimination of odd-indexed unknowns, halving the problem size at each step. Its parallelizability makes it useful for modern computing architectures. The continued development of CR underscores its importance in numerical linear algebra, with applications spanning scientific computing, engineering simulations, and computational physics.

The foundational work on Cyclic Reduction was first presented in Hockney's 1965 paper [8], where it was applied to tridiagonal systems with periodic boundary conditions. This algorithm, developed in collaboration with Golub, was later extended and analyzed by Buzbee, Golub, and Nielson [6], and by Buneman [5], who contributed a more numerically stable formulation. Gander and Golub [7] offered a detailed explanation of the algorithm using Schur complements to eliminate odd indices, demonstrating that the new system is reduced to half its original size while retaining its Toeplitz structure, enabling the process to be applied recursively [3]. Sweet [9] proposed an approach that improved the algorithm's applicability in parallel computing environments. The CR method has been recognized for its performance on block tridiagonal and block Toeplitz matrices, which commonly arise in discretized PDEs. CR's parallelization potential has made it an important algorithm in computational science.

3 METHODOLOGY

3.1 Parallel Cyclic Reduction

The proposed method focuses on developing a parallel version of the Cyclic Reduction (CR) algorithm to solve tridiagonal linear systems efficiently. The CR algorithm is a well-known method for solving structured linear systems, particularly tridiagonal systems, which frequently arise in scientific and engineering applications such as solving partial differential equations (PDEs) and physical simulations. The goal of this proposal is to leverage parallelism to achieve significant speedup, especially for large-scale systems, while maintaining the accuracy of the solution.

3.1.1 Overview of the Cyclic Reduction Algorithm. The Cyclic Reduction algorithm is an iterative method that reduces a tridiagonal system of equations into smaller subsystems by eliminating odd-indexed variables at each step. The process continues recursively until the system is reduced to a single equation, which is then solved directly. The solution is then propagated back through the system to solve for the remaining variables.

The algorithm consists of two main phases:

- **Reduction Phase:** In this phase, the system is recursively reduced by eliminatingodd-indexed variables. At each step, the system size is halved, and the process continues until the system is reduced to a single equation.
- **Substitution Phase:** Once the reduced system is solved, the solution is propagated back through the system to solve for the eliminated variables.

3.1.2 Parallelization Strategy. The proposed method aims to parallelize the Cyclic Reduction algorithm using the Message Passing Interface (MPI). The parallelization strategy involves distributing the computational workload across multiple processes, allowing for efficient scaling on high-performance computing (HPC) systems.

3.1.3 Performance Metrics. The performance of the parallel Cyclic Reduction algorithm will be evaluated based on the following metrics:

- **Execution Time:** The time taken to solve the tridiagonal system, measured in milliseconds.We utilize wall-clock time for this metric.
- **Speedup:** The ratio of the execution time of the sequential algorithm to the execution time of the parallel algorithm. This metric indicates the efficiency of the parallel implementation.
- **Residual Error:** The accuracy of the solution, measured as the norm of the residual vector. A smaller residual error indicates a more accurate solution.

3.2 Implementation Details

The implementation of the Parallel Cyclic Reduction (CR) algorithm is designed to solve tridiagonal linear systems efficiently in a distributed computing environment using the Message Passing Interface (MPI). The goal is to leverage parallelism to achieve significant speedup for large-scale systems while maintaining the accuracy of the solution. Below is an overview of the key components and steps involved in the implementation. Key steps include:

- **Matrix Partitioning and Initialization:** The tridiagonal matrix is partitioned among the available MPI processes. Each process is responsible for a portion of the matrix and the corresponding right-hand side vector. The matrix is stored in a compact format, with only the non-zero diagonals (main diagonal, upper diagonal, and lower diagonal) being stored. This reduces memory usage and improves cache efficiency. Each process initializes its portion of the matrix and the right-hand side vector, typically with a specific pattern (e.g., `dl[i] = -1.0, d[i] = 2.0, du[i] = -1.0`), which is common in tridiagonal systems arising from discretized PDEs.
- **Forward Reduction Phase:** The forward reduction phase involves recursively eliminating odd-indexed variables within each process's assigned portion of the matrix. This is done iteratively, with the system size being halved at each step. The elimination process requires communication between processes to exchange boundary values. Non-blocking MPI calls (`MPI_Isend` and `MPI_Irecv`) are used to overlap computation and communication, reducing the overall execution time. Each process performs local elimination and communicates with its neighbors to ensure consistency across the entire system.
- **Parallel Communication:** Parallel communication is a critical component of the implementation. During the forward reduction and backward substitution phases, processes communicate with their neighbors to exchange boundary values. The communication pattern follows a structure, where processes exchange data with their neighbors at each level of the reduction. This ensures that the elimination process is consistent across the entire system, even when the matrix is distributed across multiple processes. Non-blocking communication routines are used to minimize communication overhead and allow for overlapping computation and communication.
- **Backward Substitution Phase:** Once the system is reduced to a single equation, the solution is propagated back through the system in the backward substitution phase. Each process solves for the variables within its assigned portion of the matrix, using the boundary values received from neighboring processes. The solution is then propagated back through the system to solve for the remaining variables. This phase also involves communication between processes to ensure that the solution is consistent across the entire system.
- **Solution Aggregation:** After the backward substitution phase, the final solution is aggregated from all processes. The root process (usually process 0) gathers the solution vectors from all processes and writes the results to a file. This allows for further analysis and comparison with other solvers. The solution aggregation step ensures that the final solution is available for validation and post-processing.
- **Residual Error Calculation** The implementation includes a function to calculate the residual error, which measures the accuracy of the solution. The residual error is computed as the norm of the residual vector $\|Ax - b\|$, where A is the tridiagonal matrix, x is the solution vector, and b is the right-hand side vector. This function is used to validate the correctness of the parallel solver by comparing the residual error with a predefined tolerance (`EPSILON`).

The implementation of the parallel Cyclic Reduction algorithm is designed to efficiently solve tridiagonal linear systems in distributed computing environments. By leveraging MPI for parallelization, the implementation achieves significant speedup for large systems while maintaining the accuracy of the solution. The key features of the implementation include efficient communication, load balancing, and memory efficiency, making it suitable for high-performance computing applications.

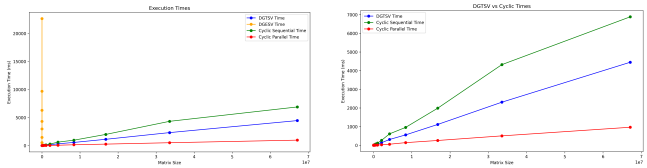
4 EXPERIMENTAL SETUP

4.1 Computing Environment

The implementation of Parallel CR and experiments were conducted on a 64-core HPC cluster managed by SLURM job queueing system (Greyfurt machine of METU Computer Engineering Department). Greyfurt platform supports up to 64 MPI processes, making it suitable for scalability testing and performance evaluation of parallel algorithms. Although not a large-scale cluster, it provides a controlled environment for benchmarking. The code was written and debugged on local devices for rapid iteration and testing. Later, it was deployed to Greyfurt for performance testing and benchmarking, leveraging SLURM for efficient job scheduling and resource management.

4.2 Key Features of the Testing Platform

- 64 cores for parallel execution.
- SLURM for job queueing and resource allocation.
- Scalability testing with up to 64 MPI processes.
- Greyfurt platform provided a reliable environment for evaluating the performance and scalability of the parallel Cyclic Reduction algorithm.



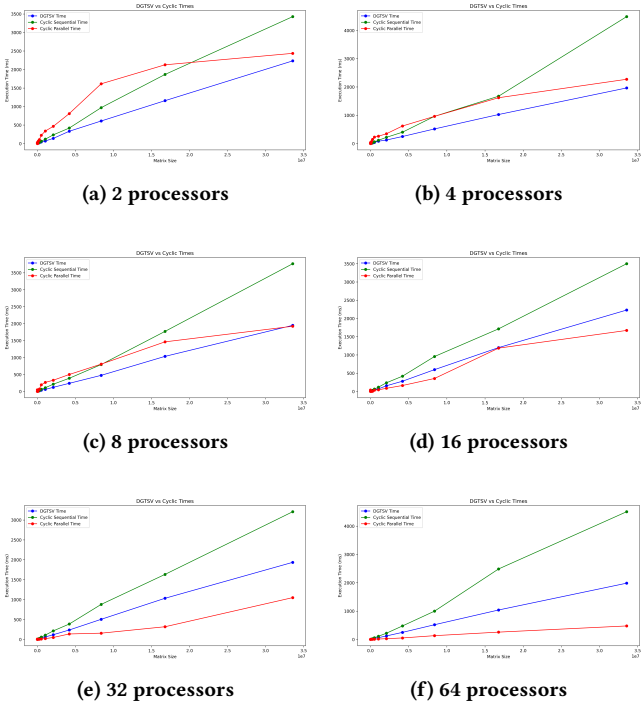
(a) Execution time comparison for DGTSV, DGESV, sequential CR, and parallel CR

(b) Execution time comparison for DGTSV, sequential CR, and parallel CR

Figure 1: Execution time comparison of standard linear solvers and Cyclic Reduction algorithm, sequential and parallel for 64 processors

5 RESULTS & DISCUSSION

The discussion begins with a comparison of our proposed solution to existing methodologies and the sequential implementation of the code. As shown in **Figure 1a**, the runtime for DGESV is the highest among all methods. Since DGESV involves the full utilization of $n \times n$ matrices, memory limitations arise for larger matrix sizes, also



(a) 2 processors (b) 4 processors

(c) 8 processors (d) 16 processors

(e) 32 processors (f) 64 processors

Figure 2: Execution time comparison for DGTSV, sequential CR, and parallel CR with varying processor counts and matrix sizes

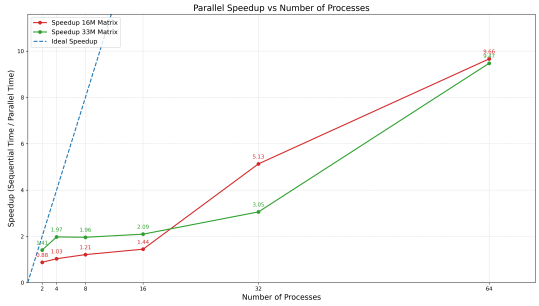
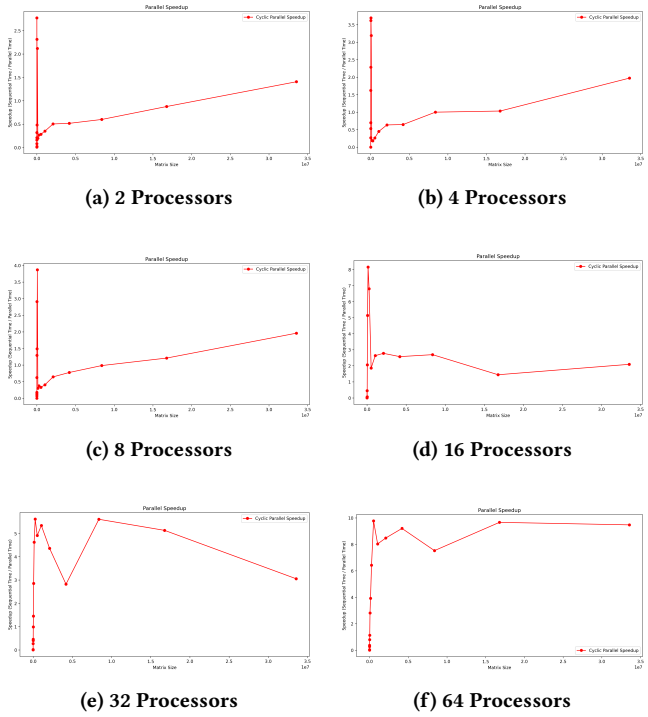


Figure 3: Parallel Speedup vs Number of Processes for 16M and 33M Matrices, with Ideal Speedup as a Reference.

to significantly increased execution times. In **Figure 1b**, DGESV is excluded for clarity. It can be seen that our parallel solution outperforms both the sequential algorithm and the DGTSV solver. However, as expected, the sequential solution lags behind DGTSV. While both algorithms share the same computational complexity, DGTSV leverages Gaussian elimination with partial pivoting, which has a lower constant factor. **Figure 2** allows for a comparison of the performance of the parallel code across different runs, showing that the runtimes of DGTSV and the sequential algorithm remain similar, as they do not utilize multiple cores.



(a) 2 Processors (b) 4 Processors

(c) 8 Processors (d) 16 Processors

(e) 32 Processors (f) 64 Processors

Figure 4: Speedup analysis for Cyclic Parallel Algorithm across different processor counts.

The performance of the Parallel Cyclic Reduction (CR) algorithm was further evaluated by calculating the speedup, defined as the ratio of sequential execution time to parallel execution time. Speedup was measured for various numbers of processes (2, 4, 8, 16, 32, and 64) and for two matrix sizes: 16 million and 33 million. As anticipated, the speedup increased with the number of processes, aligning with the principle that parallelization reduces execution time by distributing the workload across multiple processors. For smaller process counts (2, 4, 8, and 16), the speedup increase was more pronounced for the 33 million-sized matrices. This is because larger matrices offer more computational work, enabling better utilization of available processes and mitigating the relative impact of communication overhead. Refer to **Figure 4**.

The ideal speedup is linear, implying that the speedup should theoretically equal the number of processes (PP). However, practical speedup is often lower due to factors such as communication overhead, load imbalance, and the serial portions of the algorithm. For the 33 million-sized matrices, the speedup closely approximates the ideal speedup for smaller process counts (2, 4, 8, and 16). This indicates that the algorithm scales efficiently for large problem sizes when the number of processes remains moderate. However, as the process count increases, the speedup deviates from the ideal due to the growing impact of communication overhead. See **Figure 3**.

As illustrated in **Figure 3**, the speedup for 32 and 64 processes is higher for the 16 million-sized matrices. This can be attributed to the increased significance of communication overhead relative to computational work as the number of processes grows. For smaller

matrices (16 million), the communication overhead has a less pronounced impact compared to larger matrices (33 million), resulting in improved speedup. However, even for 16 million-sized matrices, the speedup falls short of the ideal linear speedup for 32 and 64 processes, indicating that communication overhead become more pronounced at higher process counts.

6 CONCLUSION AND FUTURE WORK

6.1 Conclusion

In this study, an efficient Parallel Cyclic Reduction (CR) algorithm using the Message Passing Interface (MPI) to solve tridiagonal linear systems has been developed and proper analysis conducted. The algorithm was implemented to leverage parallelism, achieving significant speedup while maintaining solution accuracy. The performance of the parallel CR implementation compared against traditional solvers such as LAPACK's DGTSV and DGESV, as well as a sequential version of the Cyclic Reduction algorithm.

The results demonstrated that the parallel Cyclic Reduction algorithm scales effectively, with speedup increasing as the number of processes grows. For smaller numbers of processes, the speedup was more pronounced for larger matrices. However as the number of processes increased communication overhead preventing the speedup from reaching the ideal linear speedup.

6.2 Future Work

The study highlights the viability of the parallel CR algorithm for solving tridiagonal-banded linear systems in cluster-type environments. Future work will focus on optimizing communication patterns and extending the approach to more complex systems, such as block banded matrices, which include block tridiagonal matrices.

REFERENCES

[1] 2025. https://en.wikipedia.org/wiki/Tridiagonal_matrix_algorithm
[2] n.d.. *LAPACK Users' Guide – Third Edition.* https://www.netlib.org/lapack/lug/
[3] D. A. Bini and B. Meini. 2009. The cyclic reduction algorithm: from Poisson equation to stochastic processes and beyond. *Numerical Algorithms* 51, 1 (2009), 23–60. https://doi.org/10.1007/s11075-008-9253-0
[4] William Briggs, Van Henson, and Steve McCormick. 2000. *A Multigrid Tutorial, 2nd Edition.*
[5] O. Buneman. 1969. *A compact non-iterative Poisson solver.* Technical Report SUIPR Rep. 294. Institue for Plasma Research, Stanford University, Paolo Alto, California.
[6] B.L. Buzbee, G.H. Golub, and C.W. Nielson. 1970. On direct methods for solving Poisson's equations. *SIAM J. Numer. Anal.* 7 (1970), 627–656.
[7] Walter Gander and Gene H. Golub. 1997. Cyclic reduction—history and applications. In *Scientific Computing*. 73–85.
[8] R. W. Hockney. 1965. A Fast Direct Solution of Poisson's Equation Using Fourier Analysis. *J. ACM* 12, 1 (January 1965), 95–113.
[9] R. A. Sweet. n.d.. A Parallel and Vector Variant of the Cyclic Reduction Algorithm. (n.d.).