BNL-219940-2020-PUCP

Approximate Inverse Chain Preconditioner: Iteration Count Case Study for Spectral Support Solvers

M. Langston, M. Lin

Submitted to the 2020 IEEE High Performance Extreme Computing Virtual Conference Conference
to be held at Virtual Conference
September 21 - 25, 2020

June 2020

Computational Science Initiative

Brookhaven National Laboratory

U.S. Department of Energy

USDOE Office of Science (SC), Advanced Scientific Computing Research (SC-21)

DISCLAIMER

Approximate Inverse Chain Preconditioner: Iteration Count Case Study for Spectral Support Solvers

M. Harper Langston, Pierre-David Letourneau, Julia Wei,
Larry Weintraub, Mitchell Harris, Richard Lethin
Reservoir Labs, Inc., New York, NY 10012
Email: {langston,letourneau,wei,weintraub,harris,lethin}@reservoir.com

Eric Papenhausen
Akai Kaeru
New York, NY 10128
Email: epapenha@akaikaeru.com

Meifeng Lin
Brookhaven National Lab,
Upton, NY 11973
Email: mlin@bnl.gov

***Abstract*—As the growing availability of computational power slows, there has been an increasing reliance on algorithmic advances. However, faster algorithms alone will not necessarily bridge the gap in allowing computational scientists to study problems at the edge of scientific discovery in the next several decades. Often, it is necessary to simplify or precondition solvers to accelerate the study of large systems of linear equations commonly seen in a number of scientific fields. Preconditioning a problem to increase efficiency is often seen as the best approach; yet, preconditioners which are fast, smart, and efficient do not always exist.**

Following the progress of [1], we present a new preconditioner for symmetric diagonally dominant (SDD) systems of linear equations. These systems are common in certain PDEs, network science, and supervised learning among others. Based on spectral support graph theory, this new preconditioner builds off of the work of [2], computing and applying a V-cycle chain of approximate inverse matrices. This preconditioner approach is both algebraic in nature as well as hierarchically-constrained depending on the condition number of the system to be solved. Due to its generation of an Approximate Inverse Chain of matrices, we refer to this as the AIC preconditioner.

We further accelerate the AIC preconditioner by utilizing precomputations to simplify setup and multiplications in the context of an iterative Krylov-subspace solver. While these iterative solvers can greatly reduce solution time, the number of iterations can grow large quickly in the absence of good preconditioners. Initial results for the AIC preconditioner have shown a very large reduction in iteration counts for SDD systems as compared to standard preconditioners such as Incomplete Cholesky (ICC) and Multigrid (MG). We further show significant reduction in iteration counts against the more advanced Combinatorial Multigrid (CMG) preconditioner.

We have further developed *no-fill* sparsification techniques to ensure that the computational cost of applying the AIC preconditioner does not grow prohibitively large as the depth of the V-cycle grows for systems with larger condition numbers. Our numerical results have shown that these sparsifiers maintain the sparsity structure of our system while also displaying significant reductions in iteration counts.[1] [2]

***Index Terms*—spectral support solver, linear systems, fast solvers, preconditioners, multigrid, graph laplacian, benchmarking, iterative solvers, precomputations, approximate inverse chain, sparsifiers, iterative solvers**

[1]The research in this document was performed in connection with contract/instrument DARPA HR0011-12-C- 0123 with the U.S. Air Force Research Laboratory and DARPA. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. Distribution Statement "A" (Approved for Public Release, Distribution Unlimited). The information in this report is proprietary information of Reservoir Labs, Inc.

[2]Further support from the Department of Energy under DOE STTR Phase I/II Projects DE-FOA-00000760/DE-FOA-000101.

I. Introduction

For many numerical methods and problems, scientific computing approaches have been reaching the limits at which we can accelerate solvers and approaches for HPC [3]. The 20^{th} and 21^{st} centuries have seen a massive explosion in the number of algorithms for accelerating the solutions of systems of linear equations [4]. Following the exponential growth of computing power through the 1960s to 1980s, these algorithmic advances slowed in favor of leveraging ever-increasing computational power [5]–[8]. Despite the reliance on compute power, a number of important algorithmic innovations such as Multigrid methods (i.e., hierarchical representations) [9], [10], the Fast Multipole Method [11]–[14] and the exploitation of sparsity and patterns (e.g., FFTs [15], [16], sparse FFTs [17], [18] and reduced-communication FFTs [14], [19], [20]) have further advanced scientific discovery; however, we are rapidly approaching the point at which additional compute power will not yield significantly better results. It is becoming increasingly important to develop new algorithmic advances as well as find new ways of accelerating existing computational techniques.

Consider solving large systems of linear equations of the form $Ax = b$, which have benefited greatly from computational and algorithmic advances, consider. As introduced and discussed in [1], Krylov-subspace solvers, particularly *Conjugate Gradient (CG)*- and *Generalized Minimal Residuals (GMRES)*-based solvers [21]–[25], are commonly-employed approaches to iteratively solving such systems. These approaches are often extremely fast and efficient at each step of the solver, but the number of steps can become quite large, and communication or parallelization across iterations is often not feasible. In the absence of new algorithms or more compute power, it is often necessary to precondition a problem, or condition it to be solved more quickly with existing techniques in fewer steps.

The development and advancement of new preconditioners such as [1], where the system can be preconditioned in terms of its underlying structure, allow for faster solvers with existing techniques and computational resources. Preconditioners, which can further exploit the hierarchical nature and structural sparsity of systems of linear equations, lead to *smarter* solvers. Preconditioners that are smart, adaptive, and can be constructed on the fly in a problem-specific fashion will have a great impact on the types of problems and sizes of problems that computational scientists will need and want to tackle in the next 50 years.

In this paper, we follow the discussion in [1] by looking at a preconditioner based on spectral support-graph theory [26]–[30]. In particular, we focus on the method of [2], in which an *approximate inverse chain* of matrices is constructed and then applied at each iteration of the linear solver. We begin by outlining the theory and algorithms for this solver in sections II-III; we then look at initial results in Section IV; sparsification techniques, necessary for retaining structural sparsity for efficiency, are introduced and discussed in Section V with results in Section VI. Our focus here is mainly in studying effective ways of reducing iteration counts, and we discuss plans for incorporating the results here with [1] in Section VII.

II. Background And Motivation for Symmetric Diagonally Dominant Iterative Solvers

As in [1], we remain interested in positive-definite, symmetric systems, for which solutions are required in a number of areas, including in the solution of certain PDEs with finite elements [12], [31], in semi-supervised learning problems [32]–[34], SDP approaches for graph partitioning [35], and in studying graph flow problems such as max-flow [36] and congestion control [37]. For these systems the CG in Algorithm 1 can be leveraged due to simplicity and reliance solely on vector-vector additions and matrix-vector multiplications.

Algorithm 1 CG algorithm for solving $Ax = b$ with a positive definite matrix A.

if $k := 0$ **then**
  $x_0 := 0$
  $r_0 := b - Ax_0$, $p_0 := r_0$
end if
while $||r_k|| > \epsilon$ **do**
  $\alpha_k := \frac{(r_k, r_k)}{(p_k, Ap_k)}$
  $x_{k+1} := x_k + \alpha_k p_k$, $r_{k+1} := r_k - \alpha_k A p_k$
  $\beta_k := \frac{(r_{k+1}, r_{k+1})}{(r_k, r_k)}$
  $p_{k+1} := r_{k+1} + \beta_k p_k$
  $k := k + 1$
end while

As discussed in [1], [2], for systems with large condition number, $\kappa(A)$, the CG method's rate of convergence can be adversely affected [38]. The approach we discussed in [1] involved using Combinatorial Multigrid (CMG) as preconditioner to compare against Incomplete Cholesky (ICC) and Multigrid-based preconditioning solvers. The Preconditioned Conjugate Gradient (PCG) constructs a matrix $\mathbf{P}$, which approximates $\mathbf{A}$ in some desired ways and whose inverse is relatively easy to compute in order to solve:

$$\mathbf{P}^{-1}\mathbf{A}\mathbf{x} = \mathbf{P}^{-1}\mathbf{b}. \tag{1}$$

Here, $\mathbf{P}$ is constructed such that it both lowers the iteration count (increases the rate of convergence of the iterative solver) while not increasing the computational cost of each step [23]. For more information on the Combinatorial Multigrid and Incomplete Cholesky approaches, we refer to [1].

As in our prior work, we are focused primarily on symmetric diagonally-dominant (SDD) matrices, $\mathbf{A}$ where $A_{ij} = A_{ji}$ and $A_{ii} \geq \sum_{j \neq i} |A_{ij}|$; these systems are common in a variety of fields as mentioned above.

For a symmetric real matrix, $\mathbf{A} = \mathbf{A}^T$, and we use the notation $\mathbf{A} \succcurlyeq \mathbf{0}$ to indicate that $\mathbf{A}$ is positive semi-definite (all eigenvalues $\lambda_j \geq 0$); similarly, $\mathbf{A} \succ \mathbf{0}$ indicates $\mathbf{A}$ is positive definite ($\lambda_j > 0$). For matrices $\mathbf{A}, \mathbf{B}$, $\mathbf{A} \succcurlyeq \mathbf{B} \Leftrightarrow \mathbf{A} - \mathbf{B} \succcurlyeq \mathbf{0}$. Using this notation, we can say that $\mathbf{A} \approx_\epsilon \mathbf{B} \Leftrightarrow \exp(\epsilon\mathbf{A}) \succcurlyeq \mathbf{B} \succcurlyeq \exp(-\epsilon\mathbf{A})$.

Using these definitions and notation, [2] have developed a new algorithm for solving systems of equations with SDDM matrices, which we will focus on for the remainder of this paper. An SDDM matrix $\mathbf{M}$ is a generalization of an SDD matrix which can decomposed as $\mathbf{M} = \mathbf{D} - \mathbf{A}$ where $\mathbf{D}$ is a diagonal matrix, $\mathbf{A}$ is symmetric, both are non-negative,and $\mathbf{D} \succ \mathbf{A}$. Inverting $\mathbf{A}$ is potentially expensive, so [2] invoke the following identity for SDDM matrices:

$$(\mathbf{D} - \mathbf{A})^{-1} = \frac{1}{2}\left[\mathbf{D}^{-1} + (\mathbf{I} + \mathbf{D}^{-1}\mathbf{A})(\mathbf{D} - \mathbf{A}\mathbf{D}^{-1}\mathbf{A})^{-1}(\mathbf{I} + \mathbf{A}\mathbf{D}^{-1})\right] \tag{2}$$

Now, instead of inverting $\mathbf{A}$, one needs to invert $\mathbf{D}$, a trivial task. The problem, however, is that this approach also involves potentially expensive matrix-matrix multiplications. Further, we would still have to invert $(\mathbf{D} - \mathbf{A}\mathbf{D}^{-1}\mathbf{A})$, but if this process could be simplified, or if the matrices could be *sparsified*, the process outlined by [2] can be streamlined. Indeed, [2] uses the above identity to create an *approximate inverse chain* of matrices, which provides the basis for a preconditioning approach, leading to an iteration count significantly lower than those seen in other advanced preconditioners, including the CMG approach shown in [1], [39]. It is further desirable that this chain of matrices maintain a *relative structural sparsity* in order to remain computationally efficient. We discuss this in further detail in Section V.

III. Theory and Algebraic Motivation for Approximate Inverse Chain Preconditioner

Since $\mathbf{D} - \mathbf{A}$ is SDDM, it can be shown that if it is sparse, $(\mathbf{D} - \mathbf{A}\mathbf{D}^{-1}\mathbf{A})$ is sparse as well (or maintains the same level of general sparsity). Given the structure of $\mathbf{D}$ and $\mathbf{A}$, we present the following identity from [40]:

$$\mathbf{D} - (\mathbf{A}\mathbf{D}^{-1}\mathbf{A})(\mathbf{D}^{-1}(\mathbf{A}\mathbf{D}^{-1}\mathbf{A})) = \mathbf{D} - \mathbf{D}(\mathbf{D}^{-1}\mathbf{A})^4. \tag{3}$$

In particular, applying the left-hand side of equation (3) to a vector $k > 1$ times results in the right-hand side of the equation becoming $\mathbf{D} - \mathbf{D}(\mathbf{D}^{-1}\mathbf{A})^{2k}$. Since $||\mathbf{D}^{-1}\mathbf{A}|| \leq 1$ (from SDDM property), $(\mathbf{D}^{-1}\mathbf{A})^{2k} \to 0$ and $\mathbf{D} - \mathbf{D}(\mathbf{D}^{-1}\mathbf{A})^{2k} \to \mathbf{D}$ as $k \to \infty$. This naturally leads to the following approach [2]:

For a system $\mathbf{M}\mathbf{x} = \mathbf{b}$, where $\mathbf{M}$ is SDDM, let $\mathbf{M}_0 = \mathbf{M}$ such that $\mathbf{M} = \mathbf{D}_0 - \mathbf{A}_0$. Let $\mathbf{M}_1 = (\mathbf{D}_0 - \mathbf{A}_0\mathbf{D}_0^{-1}\mathbf{A}_0) := \mathbf{D}_1 - \mathbf{A}_1$. Iterate in this way, i.e., $\left\{\mathbf{M}_i = (\mathbf{D}_{i-1} - \mathbf{A}_{i-1}\mathbf{D}_{i-1}^{-1}\mathbf{A}_{i-1}) := \mathbf{D}_i - \mathbf{A}_i\right\}$ until a maximum number of iterations is reached..

Combining this with (2), we can now *approximate* $\mathbf{M}_i^{-1} = (\mathbf{D}_i - \mathbf{A}_i)^{-1}$ with,

$$\frac{1}{2}\left[\mathbf{D}_i^{-1} + (\mathbf{I} + \mathbf{D}_i^{-1}\mathbf{A}_i)(\mathbf{D}_{i+1} - \mathbf{A}_{i+1})^{-1}(\mathbf{I} + \mathbf{A}_i\mathbf{D}_i^{-1})\right]. \tag{4}$$

From equations (3) and (4), as i grows large, $(\mathbf{D}_{i+1} - \mathbf{A}_{i+1}) \to \mathbf{D}_{i+1}$. This suggests that for inverse chain parameter

Algorithm 2 Approx. Inverse Chain Solve $\mathbf{M}_0\mathbf{x} = (\mathbf{D}_0 - \mathbf{A}_0)\mathbf{x} = \mathbf{b}_0$ for depth d

for $i = 1 : d$ **do**
  Compute $\mathbf{M}_i = \mathbf{D}_{i-1} - \mathbf{A}_{i-1}\mathbf{D}_{i-1}^{-1}\mathbf{A}_{i-1} = \mathbf{D}_i - \mathbf{A}_i$.
  Compute $\mathbf{b}_i = (\mathbf{I} + \mathbf{A}_i\mathbf{D}_i^{-1})\mathbf{b}_{i-1}$.
end for
Compute $\mathbf{x}_d = \mathbf{D}_d^{-1}\mathbf{b}_d$.
for $i = d - 1 : 0$ **do**
  Compute $\mathbf{x}_i = \frac{1}{2}\left(\mathbf{D}_i^{-1}\mathbf{b}_i + (\mathbf{I} + \mathbf{D}_i^{-1}\mathbf{A}_i)\mathbf{x}_{i+1}\right)$.
end for

depth d, we can solve $\mathbf{M}_0\mathbf{x} = \mathbf{b}_0$ in Algorithm 2 [2] . Note that if the condition number of $\mathbf{M}_0$ is κ, then the maximum eigenvalue, i.e., $\lambda_{max}\left(\mathbf{D}^{-1}\mathbf{A}\right)$, is $O(1 - \frac{1}{\kappa})$ (Proof see [2]) such that setting $d = O(\log(\kappa))$, then $\lambda_{max}\left(\mathbf{D}^{-1}\mathbf{A}\right)^{2d} \approx 0$.

As can be seen in the algorithm above, the structure of the algorithm provides a *V*-cycle in the same spirit as many Multigrid algorithms including the CMG [1], [39], with a few matrix vector multiplications except for the construction of the chain of matrices. The V-cycle is shown in Figure 1. Precalculations can reduce the amount of per-iteration calculations in the algorithm.

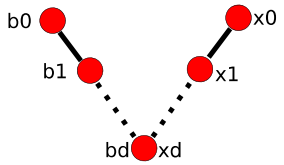

Fig. 1: Example of the V-cycle in calculation the chain of right-hand side approximations, b_i, allowing for the construction of the chain of approximate solution vectors, x_i. b_0 is the initial right-hand side at each iteration of the preconditioner, and we calculate up to a maximum depth of d (prespecified manually or calculated based on the system's condition number). x_0 is the final approximation to the solution vector at each step of the iteration.

This Approximate Inverse Chain (AIC) preconditioner [2] is framed as a solver, but really is more of a preconditioner used in the context of a CG Krylov subspace solver. Similar to [41] and [42], the approaches of [2] can be employed as a preconditioning step of an iterative solver to decrease the overall iteration count. It can further be seen from the algorithm above that it is best to precompute many of the matrix-matrix multiplications and diagonal matrix inversions such that the overall cost of each preconditioning step is $O(3dN)$ where $O(N)$ is the cost of each matrix-vector multiply. We separate the above algorithm into a precomputation step and a dynamic computation step in Algorithm 3. It can be seen how Algorithm

Algorithm 3 Precomputation with Input: $\mathbf{M}_0$, depth d and output $\left\{\mathbf{C}_i, \mathbf{E}_i, \tilde{\mathbf{D}}_i\right\}$

for $i = 1 : d$ **do**
  Let $\mathbf{M}_i = \mathbf{D}_{i-1} - \mathbf{A}_{i-1}\mathbf{D}_{i-1}^{-1}\mathbf{A}_{i-1} = \mathbf{D}_i - \mathbf{A}_i$.
  Compute $\tilde{\mathbf{D}}_i = \mathbf{D}_i^{-1}$.
  Compute $\mathbf{C}_i = (\mathbf{I} + \mathbf{A}_i\tilde{\mathbf{D}}_i)$.
  Compute $\mathbf{E}_i = (\mathbf{I} + \tilde{\mathbf{D}}_i\mathbf{A}_i)$.
end for

2 reduces to a $O(3d)$ matrix-vector multiplications in the solver with the precomputation input from Algorithm 2 (shown in Algorithm 3). In fact, these $\mathbf{M}_i$ matrices can be pre-computed,

Algorithm 4 Approx. Inverse Chain Solve $\mathbf{M}_0\mathbf{x} = \mathbf{b}_0$ for depth d with input $\left\{\mathbf{C}_i, \mathbf{E}_i, \tilde{\mathbf{D}}_i\right\}$ as precomputed in Algorithm 3

for $i = 1 : d$ **do**
  Compute $\mathbf{b}_i = \mathbf{C}_i\mathbf{b}_{i-1}$.
end for
Compute $\mathbf{x}_d = \tilde{\mathbf{D}}_d\mathbf{b}_d$.
for $i = d - 1 : 0$ **do**
  Compute $\mathbf{x}_i = \frac{1}{2}\left(\tilde{\mathbf{D}}_i\mathbf{b}_i + \mathbf{E}_i\mathbf{x}_{i+1}\right)$.
end for

so that step in the initial descent can be pulled out of the loop. As mentioned in Section I, if care is not taken in Algorithms 2 and 3, the inverse chain can become prohibitively dense. We introduce an approach for tackling this potential bottleneck in Section V. First, in the next section, we discuss initial prototype results for the Approximate Inverse Chain preconditioner as compared to other preconditioners.

IV. Initial Results for a Non-Sparsified solver

The Approximate Inverse Chain approach in [2] utilizes a Richardson iteration approach with the original Algorithm 1 solver in the preconditioning step while [40] uses a Chebyshev iterative method for optimal theoretical asymptotics; however, in practice neither of these solvers are particularly good for general problems as compared to more stable and mature iterative solvers. As discussed in [1], for sparse SDD (or SDDM) systems the Conjugate Gradient (CG) algorithm, is a better, more stable candidate [21]–[23]. As previously discussed in further detail in [1], for large condition numbers, the number of iterations required for convergence can become unacceptably high but these iteration counts can be reduced with *preconditioners*. To review, we seek to construct a preconditioner $\mathbf{P}$ from (1) that reduces the number of steps for an iterative method while not introducing significant cost to setup and apply at each iteration. For the purposes of a symmetric positive definite $\mathbf{A}$, $\mathbf{P}$ is also required to have the same symmetry properties as $\mathbf{A}$; the convergence rate of the iterative solver would therefore be based on the structure and condition number of $\mathbf{P}^{-1}\mathbf{A}$. In [1], we showed results for simple preconditioners such as the Modified Incomplete Cholesky (MIC) as well for the more complex Combinatorial Multigrid (CMG) preconditioner.

For testing the effectiveness of the AIC preconditioner, we first implemented and tested a `MATLAB` prototype. We utilized the same sample test as in [1] seen in Figure 2, in which we construct a simple matrix $\mathbf{A}$ using a 5-point discrete Laplacian stencil on top of a regular two-dimensional grid. The result is a very sparse system. We then solve $\mathbf{A}\mathbf{x} = \mathbf{1}^T$ for $\mathbf{x}$ using a PCG solver with a residual tolerance set to $\epsilon_{\text{res}} = 1e - 8$ and several preconditioners. In Figure 3, we investigate the number of iterations needed to reach the desired residual for no preconditioner, an IC, MIC, CMG, and the new Approximate Inverse Chain (AIC) preconditioner as we increase the size of the grid used to construct $\mathbf{A}$, thereby increasing the number of nonzeros and the resulting condition number $\kappa(A)$. We observe that from a pure iteration count perspective, for larger problems the AIC preconditioner performs even better than our CMG

```
r = zeros(1,m);
r(1:2) = [2, -1];
T = toeplitz(r);
E = speye(m);
A = kron(T,E) +
kron(E,T);
p = symamd(A);
A = A(p,p);
```

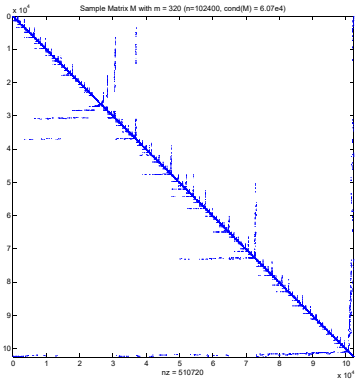

Fig. 2: Sample input to the CMG preconditioner/solver. *Left*: Sample MATLAB code to generate the systems of various sizes; *Right*: System with $n = 102400$, 510720 non-zeros, and condition number, $\kappa \approx 6e4$. This code can generate systems of various sizes.

implementation, which in itself performs better than standard approaches.

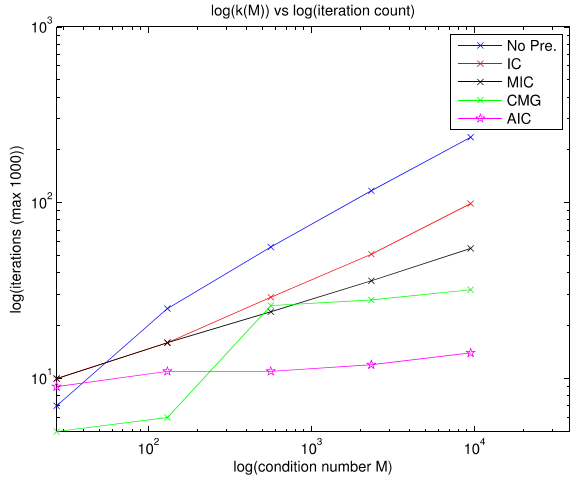

Fig. 3: For experiments from [1], we plot the condition number against the number of iterations. For small $\kappa(A)$, various Incomplete Cholesky (IC/MIC) approaches are successful, but CMG outperforms all other approaches as $\kappa(A)$ grows. However, our implementation of the AIC method from [2] greatly outperforms CMG in terms of iteration count for larger problems.

We note that in Figure 3, we set the depth of the AIC solver to be no greater than $d = 10$; however, as noted, the chain of matrices can become increasingly dense, resulting in an effectively-dense solver in terms of matrix-vector multiplications. In the next section, we discuss a simple approach to producing a set of *sparsified* matrices in the approximate inverse chain to guarantee that structural sparsity is maintained.

V. Sparsification Approaches for Simplified Chains

When $\mathbf{A}_0$ with condition number κ is passed as input to Algorithms 2 and 3, the matrix product $\mathbf{A}_{i-1}\mathbf{D}_{i-1}^{-1}\mathbf{A}_{i-1}$ can become dense for some $i \leq d$. It is desirable to maintain a level of sparsity in the chain of matrices that is on the same relative order of $\mathbf{A}_0$ in order to avoid computational bottlenecks. [2] note that these bottlenecks can be partially overcome through parallelization approaches but more importantly through sparsification techniques as outlined in [29] and [43]. Sparsifying the matrix construction on-the-fly at each step in the chain construction i results in an approximate calculation $\mathbf{M}_i \approx_\epsilon \mathbf{A}_{i-1}\mathbf{D}_{i-1}^{-1}\mathbf{A}_{i-1}$. Sparsification methods are often based on a low-stretch spanning tree (LSST) construction; however, we have implemented an LSST algorithm and found this approach in practice to be computationally prohibitive.

Instead of an LSST-based sparsification approach, we have developed an approach for sparsifying the inverse chain of matrices by measuring the sparsity of the original system and then maintaining a relative level of sparsity in the subsequent chain of matrices based on the ideas of Incomplete Cholesky (ICC) preconditioner methods. ICC methods are popular as preconditioners in solving systems of linear equations with sparse matrices [23]. The basic idea is to use *no fill* or *zero fill* in constructing the preconditioner $\hat{K}$ when corresponding locations in $A = KK^T$ are zero during Cholesky factorization. That is, ICC methods set $K_{ij} = 0$ when $A_{ij} = 0$ during the Cholesky factorization. For sparse matrices A, this guarantees that K is as sparse as A's lower-triangular half. In the context of developing a sparsifier for the AIC preconditioner, we use a *no-fill* strategy by leveraging the ideas from ICC preconditioners.

We sparsify the approximate inverse chain of matrices in our preconditioner implementation by performing a *no-fill* approach to $\mathbf{D}_i - \mathbf{A}_i\mathbf{D}_i^{-1}\mathbf{A}_i$, based on a specified threshold of heuristics obtained from $\mathbf{A}_0$. In general, the main requirement in sparsification routines is that the number of new edges in the product of matrices remain bounded [2]. That is, if $\mathbf{A}_i$ is of size $n \times n$ with m non-zeros (in the context of support-graph theory, this is analogous to a graph with n vertices and $O(m)$ edges), then successfully sparsifying $\mathbf{D}_i - \mathbf{A}_i\mathbf{D}_i^{-1}\mathbf{A}_i$ should result in $O(n+mlog(n/\epsilon^2))$ non-zero entries, and then further reduced to $O(nlog^c(n/\epsilon^2))$, where ϵ is a nonnegative value chosen to be less than $1/2$ and c is constant. We can achieve the same resulting sparsity by effectively not creating new entries in the chain of matrices in the AIC solver once a new matrix A_i in the chain has achieved a constant multiplier level of sparsity from the original matrix. That is, if $\mathbf{A}_0$ has m non-zero entries, we can specify maximum density in the chain of matrices to be $C \cdot m$ for some small constant $C << m$ such that at some depth d in the chain $\{\mathbf{A}_i\}$, the number of nonzeros in all matrices $\mathbf{A}_j$, where $j \geq d$, will be $\approx C \cdot m$.

In Figure 4, we show examples for performing the matrix-matrix multiplications in developing the chain of matrices for the AIC preconditioner, displaying the resulting density from (1) not sparsifying the matrix product as well as from (2) applying the no-fill sparsifier. In Figure 4, a system of size $n = 1600$ (small for illustrative purposes) is built to a chain of depth $d = 6$, initiating the no-fill strategy at $i = 4$. It can be seen how the general structure of the matrices remains consistent while maintaining a desired level of sparsity. We further note that while it is possible to simply zero entries not desired after computing the matrix product, in practice we do not perform a full level of computations, instead ignoring

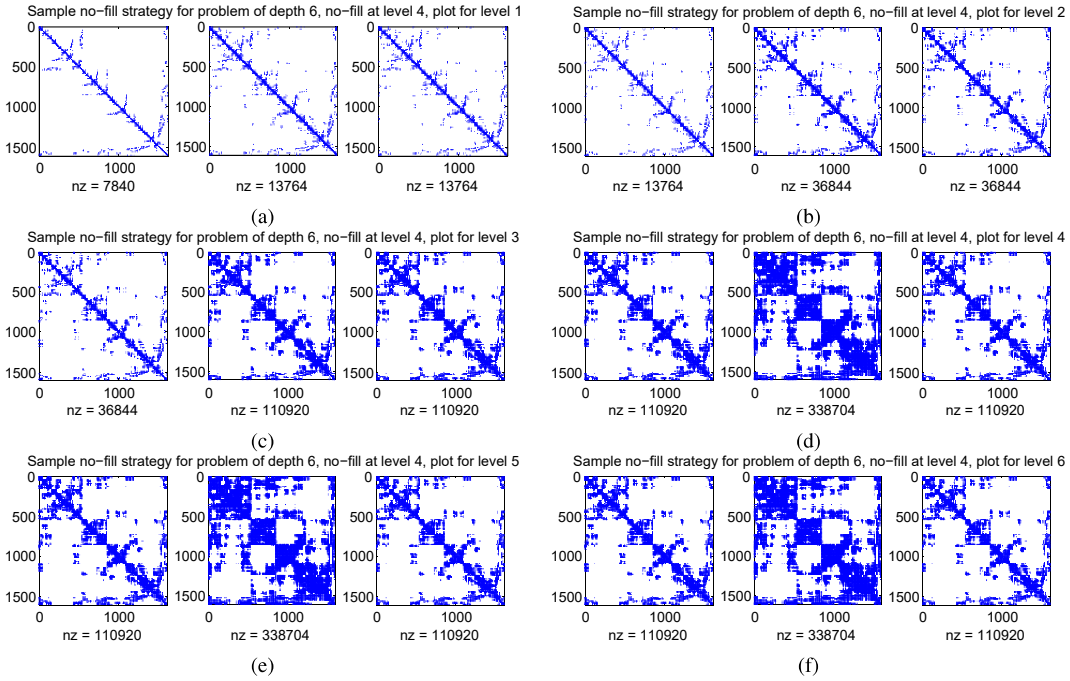

Fig. 4: Example *no-fill* sparsification strategy for input of size $n = 1600$. Here, we arrest the fill of nonzero locations in computing $M_{i+1} = (D_i - A_i D_i^{-1} A_i)$ at level $i = 4$ for maximum depth $d = 6$. For each of the size figures, there are three subfigures. For example, for figure (a) on the left, we see the original A matrix at the prior step (in this case A_0); in the middle is the computed $(D_{i-1} - A_{i-1} D_{i-1}^{-1} A_{i-1})$ with full *fill*; the right figure is what the results looks like if *no fill* is performed (this case only occurs here for $i \geq 4$. In the figure, (a), (b), (c) and depths 1,2, and 3 (the original M_0 is at depth 0), the fill is computed and stays. For (d), (e), and (f) and depths 4, 5, and 6, no fill is performed, and the results of the nonzeroes remaining the same can be seen after zeroing out the unnecessarily filled entries.

undesired entries in the matrix multiplication phases. In the next section, we present results from an implementation of the AIC preconditioner with our no-fill sparsification techniques.

VI. Numerical Results for No-Fill Approaches

As in [1] where we developed a CMG preconditioner, we have implemented a version of the AIC preconditioner in `C/Petsc` [44], [45]. We again use the sample input from Figure 2; this input provides a good test because, while it is very sparse, the last row and column are relatively dense. Therefore, as we compute the chain of approximate inverse matrices, the resulting M_i matrices grow dense quickly if sparsification is ignored. The systems are generated in `MATLAB` and converted to a `Petsc`-friendly format using scripts. For the right-hand side, we use a standard input of $\mathbf{b} = [10]^T$. From this input, we construct the M_i from Algorithms 2 to 4, stopping the fill of the $(D_i - A_i D_i^{-1} A_i)$ at a pre-determined level to enact our *no-fill* strategy.

The preconditioner shows good results in reduced iteration counts against Incomplete Cholesky and `Petsc`'s Multigrid preconditioner for our test systems.

In Figure 5, we compare the AIC preconditioner against a standard Incomplete Cholesky (ICC) preconditioner (no-fill Cholesky) and an off-the-shelf Multigrid preconditioned Conjugate Gradient solver (PCG). We consider 12 digits of relative accuracy for the PCG solver and a maximum number of $10,000$ iterations, looking at various levels of problem size, condition number, depth of approximate inverse chain and no-fill level. As an example of executing our solver in `Petsc` on a system for a maximum of $d = 7$ levels in the chain of matrices and no-fill beginning at $i = 4$ and 12 digits of relative numerical accuracy (when PCG stops), we execute:

```
./Solver -f mat_vec_sampl.in  -pc_type AIC
-pc_ps_numlevels 7 -pc_ps_nofill_it 4 -ksp_rtol 1e-12
```

The fill strategy can also be based on an ϵ-type parameter, determining the percentage of fill based on a relative number of edges and vertices.

For our tests in Figure 5(a), we observe that for moderate levels of no-fill, the AIC preconditioner performs significantly better in terms of pure iteration count than ICC. Continuing this trend in Figure 5(b)-(f), we observe that the AIC preconditioner performs significantly fewer iterations even at the no-fill level of $NF = 3$, as compared to both the ICC and Multigrid preconditioners.

VII. Conclusions and Future Work

We have presented an implementation of a new preconditioner based on support-graph theory as introduced in [2]. Following our work on implementing the Combinatorial Multigrid (CMG) preconditioner in [1], we have shown how certain classes of SDD systems of linear equations can be solved with iterative Krylov solvers and reduced iteration counts. This new preconditioner builds an approximate inverse chain (AIC) of matrices and applies them using a V-cycle

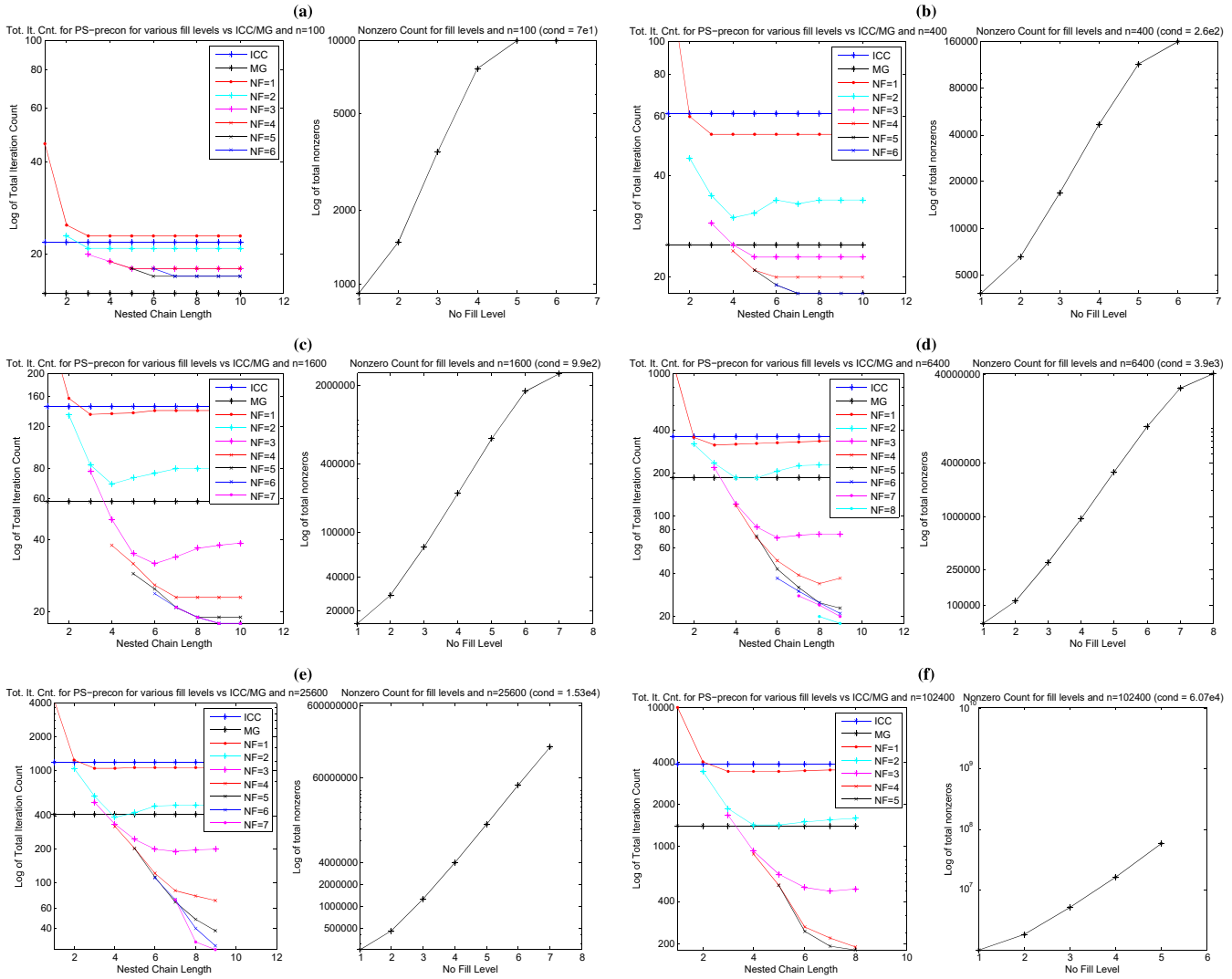

Fig. 5: We compare the total iteration count against the maximum depth d of the AIC preconditioner. We also show various *no-fill* levels and compare these results against the ICC(0) and MG preconditioners. As described above, the *no-fill level* is the level in the inverse chain of matrices at which we stop adding new entries to the subsequent matrices in the chain in the interest of maintaining sparsity. For various problem sizes and condition numbers, we further investigate the total number of maximum nonzeroes in the deepest matrix in the chain; the number of nonzeroes is based on the no-fill level. (a): Problem size of $n = 100$ and $\kappa = 7.0e1$; (b): problem size of $n = 400$ and $\kappa = 2.6e2$; (c): problem size of $n = 1600$ and $\kappa = 9.9e2$; (d): problem size of $n = 6400$ and $\kappa = 3.9e3$; (e): problem size of $n = 25600$ and $\kappa = 1.5e4$; (f): problem size of $n = 102400$ and $\kappa = 6.1e4$. For each graph, *ICC* refers to Incomplete Cholesky, *MG* refers to Multigrid, and *NF* followed by the value indicates the level at which no further matrices deeper in the the AIC solver's inverse chain add entries to existing locations with zero values.

as shown in the Algorithms in Sections II- III. We further showed how precomputations can simplify the algorithmic implementation. Prototype results showed the efficacy of the AIC preconditioner in Section IV; we discussed the need for sparsification approaches to maintain the structural sparsity of the system and introduced our no-fill sparsifier in Section V; results for the full AIC preconditioner in `C/Petsc` were presented in Section VI, where we showed that the AIC preconditioner possesses significantly-reduced iteration counts as compared to commonly-employed Incomplete Cholesky and Multigrid preconditioners.

We are currently working to combine the strengths of the new AIC and no-fill sparsification approaches with the CMG preconditioner. Additionally, we are working on parallelization techniques for computing the no-fill sparsifiers in the setup of the inverse chain of matrices; indeed, these matrices mostly require sparse matrix-matrix multiplications, which lend themselves well to parallelization.

Finally, one can further reduce the size of the V-cycle in Figure 1 by reducing the value of d; this would increase the iteration count, but there could be circumstances in which this is more computationally efficient for more dense initial systems. Such an option has not been explored, but it would be useful to ascertain if such a trade-off could be computed in an *a priori* manner, especially as we move to combine the AIC, CMG and sparsification approaches.

REFERENCES

[1] M. H. Langston, M. T. Harris, P.-D. Letourneau, R. Lethin, and J. Ezick, "Combinatorial multigrid: Advanced preconditionersfor ill-conditioned linear systems," in *IEEE Conference on High Performance Extreme Computing (HPEC), Waltham, MA, USA*. IEEE, Sep. 2019.

[2] R. Peng and D. A. Spielman, "An efficient parallel solver for SDD linear systems," *CoRR*, vol. abs/1311.3286, 2013.

[3] R. Giles, "Report from the advanced scientific computing advisory committee (ASCAC) visiting committee on exascale transition draft report for comment," U.S. Department of Energy, Office of Science, Tech. Rep., 2020.

[4] B. A. Cipra, "The best of the 20th century: Editors name top 10 algorithms," *SIAM News*, vol. 33, no. 4, p. 2, 2000.

[5] National Research Council; Committee on the Mathematical Sciences in 2025; Board on Mathematical Sciences And Their Applications; Division on Engineering and Physical Sciences, *Fueling Innovation and Discovery: The Mathematical Sciences in the 21st Century*. The National Academies Press, 2012.

[6] S. Ashby, P. Beckman, J. Chen, P. COlella, B. Collins, D. Vrawford, J. Dongarra, D. Kothe, R. Lusk, P. Messina, T. Mezzacappa, P. Moin, M. Norman, R. Rosner, V. Sarkar, A. Siegel, A. White, and M. Wright, "The opportunities and challenges of exascale computing: Summary of the ascac subcommittee," DOE, Tech. Rep., 2010.

[7] D. Brown, J. Bell, D. Estep, W. Gropp, B. Hendrickson, S. Keller-McNulty, D. Keyes, J. T. Oden, L. Petzold, and M. Wright, "Applied mathematics at the u.s. department of energy: Past, present and a view to the future," DOE, Tech. Rep., 2008.

[8] J. Ang, K. Bergman, S. Borkar, W. Carlson, L. Carrington, G. Chiu, R. C. W. Dally, J. Dongarra, A. Geist, G. Grider, R. Haring, J. Hittinger, A. Hoisie, D. Klein, P. Kogge, R. Lethin, V. Sarkar, R. Schreiber, J. Shalf, and R. Stevens, "Top ten exascale research challenges: Doe ascac subcommittee report," DOE, Tech. Rep., 2014.

[9] W. Hackbusch and U. Trottenberg, *Multigrid Methods, Lecture Notes in Mathematics Volume 960*, 1st ed. Springer-Verlag, 1982.

[10] W. Hackbusch, *Multigrid Methods and Applications*, 1st ed. Springer, 1985.

[11] L. Greengard and V. Rokhlin, "The rapid evaluation of potential fields in three dimensions," in *Vortex Methods*, ser. Lecture Notes in Mathematics, C. Anderson and C. Greengard, Eds. N.Y.: Springer Verlag, 1988.

[12] M. H. Langston, L. Greengard, and D. Zorin, "A free-space adaptive FMM-based PDE solver in three dimensions," *Communications in Applied Mathematics and Computational Science*, vol. 6, no. 1, pp. 79–122, 2011.

[13] I. Lashuk, A. Chandramowlishwaran, H. Langston, T.-A. Nguyen, R. S. Sampath, A. Shringarpure, R. W. Vuduc, L. Ying, D. Zorin, and G. Biros, "A massively parallel adaptive fast multipole method on heterogeneous architectures," *Commun. ACM*, vol. 55, no. 5, pp. 101–109, 2012.

[14] M. H. Langston, M. M. Baskaran, B. Meister, N. Vasilache, and R. Lethin, "Re-introduction of communication-avoiding FMM-accelerated FFTs with GPU acceleration," in *IEEE Conference on High Performance Extreme Computing (HPEC), Waltham, MA, USA*. IEEE, Sep. 2013.

[15] J. Cooley and J.W.Tukey, "An algorithm for the machine computation of the complex Fourier series," *Mathematics of Computation*, vol. 19, pp. 297–301, Apr. 1998.

[16] M. Frigo and S. G. Johnson, "FoFTW: An adaptive software architecture for the FFT," in *Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing*, vol. 3, Seattle, WA, May 1998, pp. 1381–1384. [Online]. Available: citeseer.ist.psu.edu/frigo98fftw.html

[17] P.-D. Letourneau, M. H. Langston, B. Meister, and R. Lethin, "A sparse multidimensional FFT for real positive vectors," *ArXiv e-prints: arXiv:1604.06682 [cs.DS*, Apr. 2016. [Online]. Available: http://arxiv.org/pdf/1604.06682v3.pdf

[18] P.-D. Letournau, M. H. Langston, and R. Lethin, "A sparse multi-dimensional fast fourier transform with stability to noise in the context of image processing and change detection," in *IEEE High Performance Extreme Computing Conference (HPEC)*, Sep. 2016.

[19] P. T. P. Tang, J. Park, D. Kim, and V. Petrov, "A framework for low-communication 1-d FFT," in *SC*, J. K. Hollingsworth, Ed. IEEE/ACM, 2012, p. 42.

[20] C. Cecka, "Low communication FMM-accelerated FFT on GPUs," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2017.

[21] O. Axelsson, *Iterative Solution Methods*. Cambridge University Press, 1994.

[22] W. Hackbusch, *Iterative solution of large sparse systems of equations*, ser. Applied mathematical sciences. New York, NY: Springer, 1994, vol. 95.

[23] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2003.

[24] O. Axelsson, "Milestones in the development of iterative solution methods," *J. Electrical and Computer Engineering*, vol. 2010, 2010.

[25] J. Shewchuk, "An introduction to the conjugate gradient method without the agonizing pain," 1994, http://www.cs.cmu.edu/ quake-papers/painless-conjugate-gradient.pdf. [Online]. Available: http://www.cs.cmu.edu/ quake-papers/painless-conjugate-gradient.pdf

[26] E. G. Boman and B. Hendrickson, "Support theory for preconditioning," *SIAM J. Matrix Anal. Appl.*, vol. 25, no. 3, pp. 694–717, Mar. 2003. [Online]. Available: http://dx.doi.org/10.1137/S0895479801390637

[27] D. A. Spielman and S.-H. Teng, "Solving sparse, symmetric, diagonally-dominant linear systems in time $0(m^{1.31})$," in *IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society, 2003, pp. 416–427.

[28] ——, "Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems," in *STOC*, L. Babai, Ed. ACM, 2004, pp. 81–90.

[29] ——, "Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems," *CoRR*, vol. cs.DS/0310051, 2003.

[30] ——, "Nearly-linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems," *CoRR*, vol. abs/cs/0607105, 2006.

[31] E. G. Boman, B. Hendrickson, and S. Vavasis, "Solving elliptic finite element systems in near-linear time with support preconditioners," *SIAM Journal on Numerical Analysis*, vol. 46, no. 6, pp. 3264–3284, 2008. [Online]. Available: https://doi.org/10.1137/040611781

[32] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf, "Learning with local and global consistency," in *Proceedings of the 16th International Conference on Neural Information Processing Systems*, ser. NIPS'03. Cambridge, MA, USA: MIT Press, 2003, p. 321–328.

[33] D. Zhou and B. Schölkopf, "A regularization framework for learning from graph data," in *ICML 2004*, 2004.

[34] X. Zhu, Z. Ghahramani, and J. D. Lafferty, "Semi-supervised learning using Gaussian fields and harmonic functions," in *ICML*, T. Fawcett and N. Mishra, Eds. AAAI Press, 2003, pp. 912–919.

[35] L. Orecchia and N. K. Vishnoi, "Towards an sdp-based approach to spectral methods: A nearly-linear-time algorithm for graph partitioning and decomposition," in *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '11. USA: Society for Industrial and Applied Mathematics, 2011, p. 532–545.

[36] Y. P. Liu and A. Sidford, "Faster energy maximization for faster maximum flow," in *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, ser. STOC 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 803–814. [Online]. Available: https://doi.org/10.1145/3357713.3384247

[37] J. Ros-Giralt, A. Bohara, S. Yellamraju, M. H. Langston, R. Lethin, Y. Jiang, L. Tassiulas, J. Li, Y. Tan, and M. Veeraraghavan, "On the bottleneck structure of congestion-controlled networks," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 3, no. 3, Dec. 2019. [Online]. Available: https://doi.org/10.1145/3366707

[38] G. Golub and C. Van Loan, *Matrix Computations*. USA: The Johns Hopkins University Press, 1996.

[39] I. Koutis, G. L. Miller, and R. Peng, "A nearly-m log n time solver for SDD linear systems," in *FOCS*, R. Ostrovsky, Ed. IEEE, 2011, pp. 590–598.

[40] R. Peng, "Algorithm design using spectral graph theory," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, August 2013.

[41] I. Koutis, G. L. Miller, and D. Tolliver, "Combinatorial preconditioners and multilevel solvers for problems in computer vision and image processing," *Computer Vision and Image Understanding*, vol. 115, no. 12, pp. 1638–1646, 2011.

[42] D. Saunders and Z. Wan, "Smith normal form of dense integer matrices fast algorithms into practice," in *Proceedings of the 2004 international symposium on Symbolic and algebraic computation*. ACM Press, 2004, pp. 274–281. [Online]. Available: http://portal.acm.org/citation.cfm?id=1005285.1005325

[43] D. A. Spielman and S.-H. Teng, "Spectral sparsification of graphs," *CoRR*, vol. abs/0808.4134, 2008.

[44] S. Balay, J. Brown, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, "PETSc Web page," 2013, http://www.mcs.anl.gov/petsc.

[45] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, K. Rupp, B. F. Smith, and H. Zhang, "PETSc users manual," Argonne National Laboratory, Tech. Rep. ANL-95/11 - Revision 3.5, 2014. [Online]. Available: http://www.mcs.anl.gov/petsc