Combinatorial Multigrid: Advanced Preconditioners For Ill-Conditioned Linear Systems

M. Harper Langston, Mitchell Tong Harris, Pierre-David Letourneau, Richard Lethin and James Ezick
Reservoir Labs Inc.
New York, NY 10012
Email: {langston,harris,letourneau,lethin,ezick}@reservoir.com

***Abstract*—The Combinatorial Multigrid (CMG) technique is a practical and adaptable solver and combinatorial preconditioner for solving certain classes of large, sparse systems of linear equations. CMG is similar to Algebraic Multigrid (AMG) but replaces large groupings of fine-level variables with a single coarse-level one, resulting in simple and fast interpolation schemes. These schemes further provide control over the refinement strategies at different levels of the solver hierarchy depending on the condition number of the system being solved [1]. While many pre-existing solvers may be able to solve large, sparse systems with relatively low complexity, inversion may require $O(n^2)$ space; whereas, if we know that a linear operator has $\tilde{n} = O(n)$ nonzero elements, we desire to use $O(n)$ space in order to reduce communication as much as possible.**

Being able to invert sparse linear systems of equations, asymptotically as fast as the values can be read from memory, has been identified by the Defense Advanced Research Projects Agency (DARPA) and the Department of Energy (DOE) as increasingly necessary for scalable solvers and energy-efficient algorithms [2], [3] in scientific computing. Further, as industry and government agencies move towards exascale, fast solvers and communication-avoidance will be more necessary [4], [5]. In this paper, we present an optimized implementation of the Combinatorial Multigrid in C using `Petsc` and analyze the solution of various systems using the CMG approach as a preconditioner on much larger problems than have been presented thus far. We compare the number of iterations, setup times and solution times against other popular preconditioners for such systems, including Incomplete Cholesky and a Multigrid approach in `Petsc` against common problems, further exhibiting superior performance by the CMG. [1] [2]

***Index Terms*—combinatorial algorithms, spectral support solver, linear systems, fast solvers, preconditioners, multigrid, graph laplacian, benchmarking, iterative solvers**

I. Introduction

Due to their power and flexibility, Krylov subspace solvers are typically employed for the solution to systems of linear equations. In general, Krylov-based iterative solvers project an n-sized system into a Krylov subspace of lower dimension. There are several iterative methods of note, including *Conjugate Gradient (CG)* and *Generalized Minimal Residuals (GMRES)*, details of which are available in [6]–[8] as well as a more contemporary look at their future in [9]. The solution of a generic system $Ax = b$ commonly involves iterative solvers; in the case of positive definite and symmetric systems CG is among the most popular. Described in Algorithm 1, it is clear that the only operations involved in the CG solver are 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

For many problems in the physical sciences, the system $Ax = b$ is often solved using the normal equation transformation by forming $A^T Ax = A^T b$ in order to be able to utilize a variant of the CG algorithm: (*CGNE/CGNR*) [10] or BiCGstab [11].

As discussed in [12] many complex solvers rely on accelerating the matrix-vector multiplication phase of iterative solver algorithms such as CG method [13] through efficient preconditioners. In particular, for a system of linear equations A with condition number $\kappa(A)$, the CG method's rate of convergence (the number of iterations in the solver for a desired precision ϵ_{prec}) can be slow when $\kappa(A)$ is large [14]. The Preconditioned Conjugate Gradient (PCG) method instead seeks to construct a system, M, which approximates A in some desired ways and whose inverse is relatively easy to compute in order to solve:

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

In developing or constructing M, the goals are to balance the difficulty in inverting the resulting system with an increased rate of convergence. As discussed in [8], preconditioning is a common, often required step in efficiently solving complex systems of linear equations, and these preconditioners are often constructed from the original system; e.g., the diagonal of the original matrix, the upper triangular portion of the matrix,

[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.

or a sparse factorization such as incomplete Cholesky (see section III for more details) when the system has few non-zeros.

The *Multigrid Method* is a technique which uses the classic iterative methods and a *divide and conquer* approach through a hierarchy of fine and coarse grids to approximate solutions, removing high-frequency error [15], [16]. Multigrid approaches can aid in strong scaling issues in some physical systems: often, as the problem size grows, the condition number of the system increases, resulting in a larger computational expense for traditional iterative approaches [17]. Algebraic Multigrid (AMG) approaches build the hierarchical operators directly from the linear system (as opposed to the regular Multigrid, often referred to as Geometric Multigrid) [18]–[20].

Built upon an introduction of combinatorial preconditioners [21]–[23], Spielman and Teng in [24] began developing a series of theoretical works on solving certain classes of sparse systems [24]–[27]. Using spectral support solvers, inverse problems on sparse systems (especially graphs) can be solved in significantly lower asymptotic time than with traditional solvers. For the restricted planar graph case, which includes computer vision and image processing, an untuned implementation in `MATLAB` of the related CMG algorithm significantly outperformed previous algorithms [1]. Parallel versions of such algorithms have appeared [28].

Spielman and Teng used the connection between a weighted graph G and a symmetric diagonally dominant matrix, A, for which $A = A^T$ and $A_{ii} \geq \sum_{j \neq i} |A_{ij}|$ for all i. That is, let $G = (V, E, \omega)$ where V is a set of vertices, E a set of edges, and ω a set of weights associated with V and E. If W is a weighted adjacency matrix and D is a diagonal matrix of weighted degrees, then $L_G = D - W$ is known as the *graph Laplacian*. This relationship can also be expressed in quadratic form,

$$\mathbf{x}^T L_G \mathbf{x} = \sum_{(i,j) \in E} \omega_{ij} (\mathbf{x}_i - \mathbf{x}_j)^2. \tag{1}$$

This definition of the Laplacian highlights the connection between graphs and positive definite systems. With graph-based systems, the support theory approaches show that by looking at the quadratic form in equation (1) with Laplacian L_G, one can sparsify these systems and use approximate Laplacians. For example, thinking of this quadratic form as the *energy dissipation* of a graph (or network) [29] L_G can be approximated by a sparsified system L_H when

$$\forall \mathbf{x} \in \mathcal{R}^n, \quad \mathbf{x}^T L_H \mathbf{x} \approx \mathbf{x}^T L_G \mathbf{x}.$$

So, as in [29], we can think of these two systems as being similar when their "energy profiles" are similar. Hence, the overall idea is that we can use a simplified system to approximate a more complex, dense one.

The CMG has only been displayed on small problems in `MATLAB` up until this point; we have re-implemented the solvers in the context of `Petsc` and performed a study using it as a preconditioner on a number of systems. We begin with a discussion of background and motivation in section II followed by numerical results and comparisons in section III.

II. Background and Support Theory Motivation for Combinatorial Mutltigrid

Combinatorial Multigrid (CMG) falls into the class of solvers for symmetric diagonally dominant (SDD) systems based on support-graph theory [30] and can be used as a combinatorial preconditioner in the context of the Conjugate Gradient (CG) iterative solver. Combinatorial preconditioners take a different approach than typical algebraic preconditioners. As often-cited from [21] and detailed further in [23], [31], combinatorial preconditioners are based on the idea that for a graph A, its Laplacian can be *preconditioned* by a sparsified or simplified graph Laplacian B which has been derived from A. Further background for support graph theory comes from the connections between electrical networks and graph Laplacians [32] with further details in [30], [33]. Many of these works are based on constructing a graph Laplacian that produces a low-stretch spanning tree, a variant of a maximal spanning tree [24]. As an example, consider Figure 1 from [29] in which two edges have been removed.

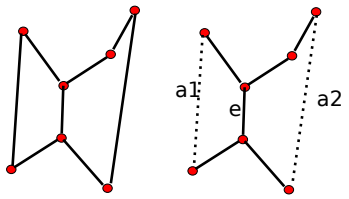

Fig. 1. On the *left* we see an original graph $G = (V, E)$ (assume unit weight edges); on the right $H = (V, E - \{a1, a2\})$, a maximal spanning tree where the removed edges are *supported* by the edge e.

[29] relies on Steiner preconditioners [34], [35], based on Steiner trees, which partition a graph $G = (V, E)$ into clusters of vertices V_i where $\sum_i |V_i| = |V| = n$. Each cluster is rooted by a *new* vertex r_i, forming star graphs S_i. The root vertices are connected directly to form a *quotient* graph Q. As an example, see figure 2 from [29].

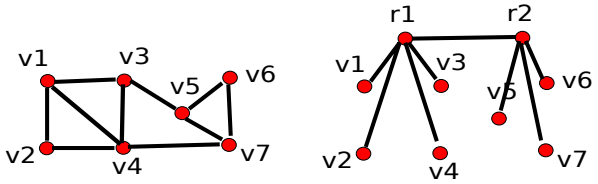

Fig. 2. On the *left* is an original graph $G = (V, E)$ with its Steiner tree $H = (\tilde{V}, \tilde{E})$ on the *right*. Note that $\tilde{V} = V \cup \{r_1, r_2\}$.

Following the discussion in [1], for a Steiner preconditioner S, let D' be the sum of the degrees of the leaves. For an $n \times m$ matrix R, let $R_{ij} = 1$ if $v_i \in S_j$ (0 otherwise). The Laplacian of S with $n + m$ vertices has the form:

$$S = \begin{pmatrix} D' & -D'R \\ -R^T D' & Q + R^T D' R \end{pmatrix}$$

In order to mitigate the issue of the additional vertices in S, [34] solve $S[z; z']^T = [y; \mathbf{0}]^T$ in lieu of $By = z$ at each iteration of the PCG solver where z' are variables that are dropped after solving. They also show this is equivalent to preconditioning with

$$B = D' - V(Q + D_Q)^{-1}V^T, \quad V = D'R, \quad D_Q = R^T D' R. \tag{2}$$

The algebraic form of B is further denoted as the Schur complement of S with r_i removed, and B is a Laplacian. [1] goes into further discussion of the analysis of support $\sigma(A/S)$, showing that it is bounded by a constant independent of n and that the conductance of the graphs induced by the vertex clusters is also bounded; this is a key result of [35].

Knowing that the support of the Steiner preconditioner is bounded, while a powerful result, does not provide a good way to choose how to build clusters, but it helps avoid bad clusters. Following the Multigrid method approach of [36], [35] shows how to build coarse grids in the same way that the quotient matrix is built, and further provides a direct way of building the restriction matrix. These results facilitate the graph decomposition algorithm discussed below.

A. Transformations from non-Laplacian SDD systems to Laplacian

Before discussing how the CMG algorithm decomposes a graph, we first note that most spectral support solvers require that the systems being solved are strict Laplacians. That is for the Laplacian A of $G = (V, E, \omega)$ where $\omega_{ij} > 0$, $A_{ij} = A_{ji} = \omega_{ij}$ and $A_{ii} = -\sum_{j \neq i} A_{ij}$; that is, A is *strictly symmetric diagonally dominant* (SDD). When A has positive off-diagonal entries [37] shows the often-proposed reduction known as *double-cover*; that is, let $A = A_p + A_n + D$ where $D = diag(A)$ and A_p is the matrix of positive off-diagonal entries (A_n is negative off-diagonal elements, or $A_n = A - A_p$). Then solving $A\mathbf{x} = \mathbf{b}$ is equivalent to solving

$$\begin{pmatrix} A_n + D & -A_p \\ -A_p & A_n + D \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ -\mathbf{x} \end{pmatrix} = \begin{pmatrix} \mathbf{b} \\ -\mathbf{b} \end{pmatrix}.$$

[1] uses this transformation for the CMG, though harnessing symmetries allows for solving systems smaller than this one, which is double the size of the original. Additionally, [1] discusses how to use the CMG for systems of the type $A = B + D_e$ where B is Laplacian and D_e is a positive diagonal matrix (such that $(B + D_e)_{ii} \neq \sum_{i \neq j} (B + D_e)_{ij}$). Similar to the above transformation, solving $A\mathbf{x} = \mathbf{b}$ is equivalent to solving

$$\begin{pmatrix} B + D_e & 0 & -d_e \\ 0 & B + D_e & -d_e \\ -d_e^T & -d_e^T & \sum d_e \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ -\mathbf{x} \\ \mathbf{0} \end{pmatrix} = \begin{pmatrix} \mathbf{b} \\ -\mathbf{b} \\ \mathbf{0} \end{pmatrix},$$

where d_e is the vector representation of D_e. Discussions on these and further transformations of SDD systems to Laplacians required for the CMG are discussed in [1].

B. From Theory to Combinatorial Multigrid Algorithm

As discussed in Section II, using the background work for [35] requires the construction of a good Steiner preconditioner. In [1], the authors introduce a graph decomposition called **Decompose-Graph** by letting $vol_G(v)$ be the weight incident to a vertex v in graph G, defining the weighted degree of v as

$$wt_d(v) = \frac{vol(v)}{max_{u \in N(v)} \omega(u, v)},$$

and the average weighted degree of G as

$$\widehat{w_d(G)} = \frac{\sum_{v \in V} wt_d(v)}{n}.$$

We now reproduce the **Decompose-Graph** algorithm from [1] below:

Algorithm 2 **Decompose-Graph**: Input $A = (V, E, \omega)$, Output Clusters V_i s.t. $V = \cup_i V_i$

1. Let $\kappa > 4$ and $W \subseteq V$ such that $wt_d(v) > \kappa \cdot \widehat{w_d(A)}$.
2. Build $F \subseteq G$, keeping the heaviest incident edges $\forall v \in V$ (F is a forest of trees).
3. $\forall w \in W$ s.t. $vol_T(w) < \frac{vol_G(w)}{\widehat{w_d(A)}}$, remove the edge contributed in Step 2. from F.
4. Decompose each $T \in F$ into vertex-disjoint trees, optimizing for max conductance.

[1] proves that **Decompose-Graph**'s resulting partition on V satisfies the conditions on its conductance by κ and the average weighted degree. Step 3 involves multiple passes over A, but the authors suggest this can be performed in multiple ways.

The connections and intuition for connecting Steiner trees/preconditioners to Multigrid solvers are discussed in [38]. For a two-level Multigrid solver, the residual error $\mathbf{r} = \mathbf{b} - A\mathbf{x}$ (when smooth) is used to compute the correction $\mathbf{c} = R^T Q^{-1} R\mathbf{r}$ for the smaller quotient graph Q and restriction matrix R, and the resulting $\mathbf{c}$ is added to $\mathbf{x}$ for the next iteration. For residual $\mathbf{r}$, low-rank operator $R^T Q^{-1} R$ approximates A^{-1} well. In [38] the author notes that the algebraic analogue for this is that the correction is equivalent to applying $(I - R^T Q^{-1} R A)$ to error vector $\mathbf{e}$. Conditions on Q and R are discussed further in [1].

Now, from [1], define $\kappa(A, B)$ to be the condition number of preconditioner B applied to A such that for $\mathbf{x} \notin nullspace(A)$,

$$\kappa(A, B) = \max_{\mathbf{x}} \frac{\mathbf{x}^T A\mathbf{x}}{\mathbf{x}^T B\mathbf{x}} \cdot \max_{\mathbf{x}} \frac{\mathbf{x}^T B\mathbf{x}}{\mathbf{x}^T A\mathbf{x}}. \tag{3}$$

For the CMG algorithm, the key idea is that $\kappa(A, B) = \kappa(\tilde{A}, \tilde{B})$ for B as defined in equation (2) and $\tilde{A} = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$, $\tilde{B} = D^{-\frac{1}{2}} B D^{-\frac{1}{2}}$. This leads to the **Two-Level Combinatorial Multigrid** Algorithm for performing relaxation below, reproduced from [1]:

The two-level algorithm is extended by [1] with recursive calls to the CMG algorithm, producing a hierarchy of graphs A_i, $i = 0, ..., d$. The full **CMG** algorithm is seen below:

Letting $nz(A)$ be the set of non-zeros of A, set t_i as:

$$t_i = max\{\lceil \frac{|nz(A_i)|}{|nz(A_{i+1})|} - 1 \rceil, 1\}.$$

Algorithm 3 **Two-Level Combinatorial Multigrid**: Input Laplacian $A = (V, E, \omega)$, right-hand side $\mathbf{b}$, intermediate solution $\mathbf{x}_j$,restriction operator R; Output updated solution $\mathbf{x}_{j+1}$

$D = diag(A)$
$\hat{A} = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$
$\mathbf{z} = (I - \hat{A}) D^{\frac{1}{2}} \mathbf{x}_j + D^{-\frac{1}{2}} \mathbf{b}$
$\mathbf{r} = D^{-\frac{1}{2}} \mathbf{b} - A\mathbf{z}$
$\mathbf{w} = RD^{\frac{1}{2}} \mathbf{r}$
$Q = RAR^T$
Solve $Q\mathbf{y} = \mathbf{w}$
$\mathbf{z} = \mathbf{z} + D^{\frac{1}{2}} R^T \mathbf{y}$
$\mathbf{x}_{j+1} = D^{-\frac{1}{2}}((I - \hat{A}\mathbf{z} + D^{-\frac{1}{2}} \mathbf{b})$

Algorithm 4 **CMG**: Input A_i, $\mathbf{b}_i$; Output $\mathbf{x}$

$D = diag(A)$
$\mathbf{x} = D^{-1}\mathbf{b}$
$\mathbf{r}_i = \mathbf{b}_i - A_i(D^{-1}\mathbf{b})$
$\mathbf{b}_{i+1} = R\mathbf{r}_i$
$\mathbf{z} = CMG(A_{i+1}, b_{i+1})$
for $i = 1 : t_i - 1$ **do**
  $\mathbf{r}_{i+1} = \mathbf{b}_{i+1} - A_{i+1}\mathbf{z}$
  $\mathbf{z} = \mathbf{z} + CMG(A_{i+1}, \mathbf{r}_{i+1})$
end for
$\mathbf{x} += R^T \mathbf{z}$
$\mathbf{x} = \mathbf{r}_i - D^{-1}(A_i\mathbf{x} - \mathbf{b}_{i+1})$

In practice, we have found that increasing the initial choice for t_0 from 1 (for the first grid in the hierarchy) to 2 can have a dramatic effect on decreasing the number of iterations while having no discernible effect on the running time.

III. Combinatorial Multigrid Numerical Results

We begin with a discussion of the Incomplete Cholesky preconditioner versus CMG in `MATLAB` and then present results for CMG versus ICC and traditional Multigrid solvers in our `Petsc` implementation.

A. CMG versus Incomplete Cholesky (ICC)

For symmetric positive definite matrices, the Cholesky factorization is a popular dense solver as it reduces the number of operations by half of a Gaussian elimination factorization in a straightforward fashion [39]. It also turns out that *Incomplete* Cholesky (ICC) factorizations are popular for sparse versions of these matrices [8]. 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, let $K_{ij} = 0$ 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 Figure 3, we show the resulting sparsity patterns for a sample matrix A and its Cholesky and ICC Factorizations.

Even in the case of ICC, the resulting factorization may be too dense (or too sparse), or not "good enough." One modification is called the *Modified Incomplete Cholesky* (MIC) Factorization, whose sparsity structure off-diagonal remains the same, but the diagonal is changed to enforce the relationship $A\mathbf{1}^T = (KK^T)\mathbf{1}^T$ [8].

Finally, we discuss another approach: *Incomplete Cholesky with Thresholding* (ICT), in which all off-diagonal elements below some $\epsilon_{tol} > 0$ are dropped [8]. Several complications arise with the ICT approach. First, the resulting Cholesky factorization may be quite dense if ϵ_{tol} is too low or too sparse if too large. Secondly, choosing ϵ_{tol} on the fly may be tricky and often involves experimentation, slowing overall solver time.

In order to see the potential power of the CMG approach, consider the following test where we construct a simple matrix A using a five-point discrete Laplacian stencil on top of a regular two-dimensional grid. The result is a very sparse system as the number of points grows large. We then solve $A\mathbf{x} = \mathbf{1}^T$ for $\mathbf{x}$ using a PCG solver with a residual tolerance set to $\epsilon_{res} = 10^{-8}$ and several preconditioners as we increase the size of the grid used to construct A, thereby increasing the number of non-zeros and the resulting $\kappa(A)$ condition number. We stop the solver at 1,000 iterations if needed and only plot up to 250 for scale; results are showing in figure 4.

For the ICT approach, we use a fixed drop tolerance of 10^{-3}. All of these tests were run in `MATLAB` using the off-the-shelf CMG implementation from [40].

As can be seen, the CMG approach performs quite well for a simple system as compared to the popular ICC preconditioner and its variants. In the next section, we use our CMG implementation to profile the various stages of the CMG, ICC and Multigrid approaches against more complex systems.

B. CMG in Petsc Results versus ICC and Mulrigrid

As indicated, we have translated the CMG algorithm from `MATLAB` [40] to `C` and `Petsc` [41] in order to utilize existing PCG implementations as well as test against standard preconditioners. For the initial recursion call to the CMG preconditioning algorithm, we invoke two recursive calls rather than one as was done in the original algorithm; we have found that the overall runtime is equivalent while the number of CG iterations is lower. Further, at the last/coarsest level of the solver, a single direct solver is called, and we utilize a fast Cholesky solver when the size of the coarsest grid falls below $n = 1000$. These two options can be changed at the command line, but we keep them constant here for these tests.

We begin by looking at a synthetic sample problem generated from the code in Figure 5; this code is a good sample test because, while it is very sparse, the last row and column are relatively dense. The systems are generated in `MATLAB` and converted to a `Petsc`-friendly format. For the right-hand side, we use a standard input of $\mathbf{b} = [1, \mathbf{0}]^T$.

We will focus on comparing CMG to the ICC preconditioner and `Petsc`'s off-the-shelf Multigrid solver/preconditioner. We compute the solution up to a request level of relative precision, 10^{-12} or a maximum of 10^4 iterations. When the number of iterations is greater than the maximum, we indicate the residual achieved; note that in these cases, the flop and wall-time counts are not necessarily accurate. The results for this test are presented below in Table I.

As can be seen in these results in Table I, the CMG significantly outperforms the other preconditioners in the total number of iterations and total wall time. We have also computed the number of flops per iteration of the solve step as well as the amount of time per iteration. As would be expected, the

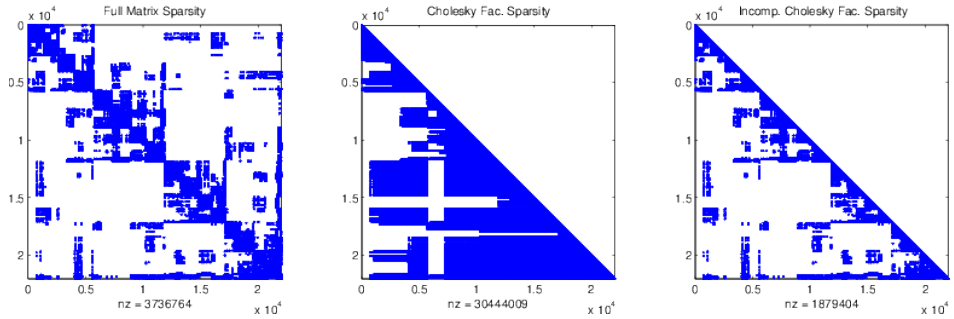

Fig. 3. *Left*: A matrix A which is positive definite, symmetric, and relatively sparse; *middle*: A's Cholesky factorization L; *right*: A's ICC Factorization. In each of these figures, the number of non-zeros is indicated by nz with a total matrix size of $10^4 \times 10^4$. This matrix A was constructed from the CSparse sample non-positive definite matrix Chen/pkustk01, C, and by composing $A = CC^T$ and then increasing the size of the weight of the diagonal.

S_{typ}	N_{its}	$\mathbf{r}$	$flps_{set}$	$flps_{slv}$	$flps_{slv}/it$	T_{set}	T_{slv}	T_{slv}/it
$n = 25600$, $m = 127360$, $\kappa(A) = 1.5e4$								
ICC	1198	$8.0e-13$	$3.6e4$	$2.6e9$	$2.2e6$	$3.0e-2$	$4.8e0$	$4.0e-3$
MG	412	$4.5e-13$	$3.3e5$	$1.2e9$	$2.9e6$	$2.2e-2$	$3.0e0$	$7.3e-3$
CMG	29	$4.7e-12$	$1.4e6$	$1.4e8$	$4.8e6$	$1.0e-1$	$5.0e-1$	$1.7e-2$
$n = 102400$, $m = 510720$, $\kappa(A) = 6.1e4$								
ICC	3923	$8.1e-13$	$1.4e6$	$3.4e10$	$8.7e6$	$1.1e-1$	$6.4e1$	$1.6e-2$
MG	1404	$3.7e-13$	$1.3e6$	$1.7e10$	$1.2e7$	$8.1e-2$	$4.0e1$	$2.8e-2$
CMG	30	$6.8e-12$	$5.8e6$	$5.7e8$	$1.9e7$	$3.9e-1$	$2.2e0$	$7.3e-2$
$n = 409600$, $m = 2045440$, $\kappa(A) = 2.4e5$								
ICC	10000	$2.2e-10$	$5.7e6$	$3.5e11$	$3.5e7$	$7.3e-1$	$6.5e2$	$6.5e-2$
MG	4608	$3.7e-13$	$5.3e6$	$2.2e11$	$4.8e7$	$3.0e-1$	$5.3e2$	$1.2e-1$
CMG	32	$6.9e-12$	$2.3e7$	$2.4e9$	$7.5e7$	$1.5e0$	$9.6e0$	$3.0e-1$
$n = 1638400$, $m = 8186880$, $\kappa(A) = 9.7e5$								
ICC	10000	$1.8e-5$	$2.3e7$	$1.4e12$	$1.4e8$	$1.7e0$	$2.7e3$	$2.7e-1$
MG	10000	$6.1e-9$	$2.1e7$	$1.9e12$	$1.9e8$	$1.1e0$	$4.7e3$	$4.7e-1$
CMG	33	$3.2e-11$	$9.3e7$	$1.0e10$	$3.0e8$	$6.8e0$	$4.0e1$	$1.2e0$
$n = 6553600$, $m = 32757760$, $\kappa(A) = 3.9e6$								
ICC	> 6000	$> 1e-4$					$> 1e4$	
MG	> 5000	$> 1e-5$					$> 1e4$	
CMG	35	$1.2e-11$	$3.7e8$	$4.2e10$	$1.2e9$	$2.2e1$	$1.7e2$	$4.9e0$

TABLE I
PETSC'S ICC AND MG PRECONDITIONERS VS. OUR C/PETSC CMG PRECONDITIONER.

amount of setup overhead for the CMG is much greater than the other preconditioners, but for larger problems in particular, the work in the setup is well worth it for significantly fewer iterations and much lower wall times. We do not report the total time due to the fact that for all preconditioners, the setup phase barely affects the overall time for solving the problem, and in many cases they are nearly identical (especially for ICC and MG). We also note that in the final test above, the ICC and MG solvers took greater than 4 hours before only reaching errors on the order of 10^{-5}, so we truncated the execution of those tests. In Table II, we quickly indicate how many levels resulted for the CMG setup phase, which clearly affects the setup time and amount of time to solve per iteration.

$\|V\| = n$	$\|E\| = m$	CMG_{lvls}
25600	127360	3
102400	510720	4
409600	2045440	5
1638400	8186880	6
6553600	32757760	7

TABLE II
NUMBER OF LEVELS IN THE GRID HIERARCHY FOR THE CMG PRECONDITIONER CREATED DURING THE SETUP PHASE.

As a final example, we consider a problem from [42], [43]. Specifically, matrix number "1403" with a given right-hand side. This problem comes from an unstructured FEM formulation of a steady state thermal problem. The undirected graph visualization for this graph is shown in Figure 6 along with its sparsity pattern.

Fig. 6. *Left*: For the unstructured FEM, steady state thermal problem, this graphic visualization is provided by [42], [43]. *Right*: The sparsity pattern for this system.

The problem and resulting matrix presented in Figures 6 has $n = 1228045$ vertices in the resulting graph and $m = 8580313$ edges with a condition number of $\kappa = 7.5 \times 10^6$. Again, we run this system through our PCG solver in Petsc with the various preconditioners and present our results in Table III.

TABLE III
RESULTS FOR THE THE UNSTRUCTURED FEM, STEADY STATE THERMAL PROBLEM IN FIGURE 6.

$n = 1228045$, $m = 8580313$, $\kappa(A) = 7.5e6$								
S_{typ}	N_{its}	$\mathbf{r}$	$flps_{set}$	$flps_{slv}$	$flps_{slv}/it$	T_{set}	T_{slv}	T_{slv}/it
ICC	10000	$2.2e7$	$1.2e12$	$2.6e3$	$1.2e8$	$1.8e0$	$2.6e3$	$2.6e-1$
MG	10000	$1.3e0$	$2.1e7$	$1.6e12$	$1.6e8$	$1.1e0$	$4.9e3$	$4.9e-1$
CMG	33	$8.7e-5$	$9.4e7$	$9.3e9$	$2.8e8$	$6.7e0$	$4.1e1$	$1.2e0$

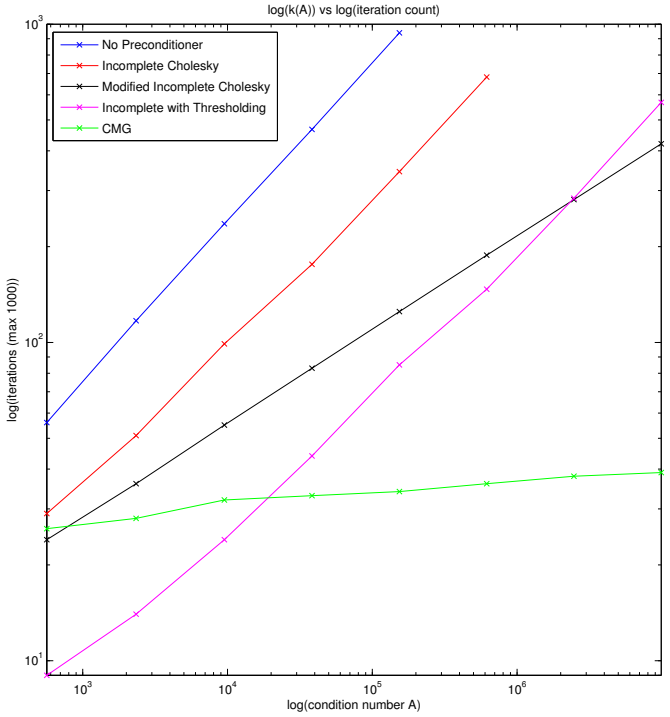

Fig. 4. For each of the experiments, we plot the condition number against the number of iterations. For small $\kappa(A)$, various IC approaches are successful, but as $\kappa(A)$ grows, CMG outperforms all other approaches.

```
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);
```

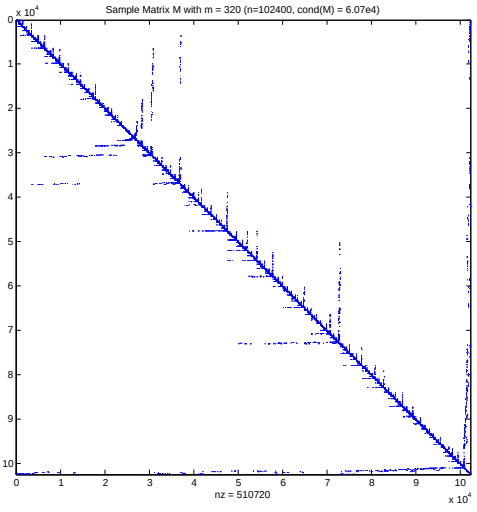

Fig. 5. Sample input to the CMG preconditioner/solver. *Left*: Sample MATLAB code to generate the systems of various sizes; *Right*: System with $n = 102400$, 510720 non-zeros, and condition number, $\kappa \approx 6e4$. This code can generate systems of various sizes, which we study in Table I.

This system is ill-conditioned and the right-hand side is intentionally poorly-designed for this problem; regardless, the CMG terminates after very few iterations with an acceptable level of relative error. If we force more iterations, we do achieve a precision of 10^{-12} within 50 iterations. Even with more iterations, it is clear that for this real-world problem, the CMG outperforms standard preconditioners for such systems. In fact, the ICC and MG preconditioners do not even come close to converging to the desired tolerance after more than 1000 iterations.

IV. CONCLUSIONS AND FUTURE WORK

We have extended the `MATLAB` implementation of [1] into C and `Petsc`, successfully showing good results on large systems that scale exceptionally in terms of the number of iterations in the solver, and we are able to solve large systems that standard preconditioners do not handle well.

In the future, we are planning to parallelize the tree structure construction for the CMG solver during setup and accelerate the internals of each iteration in the CMG using shared and distributed memory optimizations. Further, while we have incorporated the CMG into `Petsc`, we plan to build the parallelized solver into the Trilinos packages [44] for maximum portability across DOE applications.

Additionally, we have implemented an optimized version of the Approximate Inversion Chain (AIC) [12] spectral support solver in `Petsc`, which we plan to compare against the CMG solver in a future publication to further highlight the strengths of spectral support and combinatorial solvers for the types of linear systems often observed in the physical sciences.

Finally, since there is such a strong connection between spectral graph theory and graph Laplacians, the types of systems that are best suited for the CMG and spectral support solvers are often seen in network sciences and graph analytics, so we plan to utilize these solvers to explore applications to the types of problems seen in these fields and study the effectiveness of our implementations. We feel the computational savings we have observed will translate from algebraic problems for linear solvers to graph problems due to the strong link between graphs and matrices in SDD systems.

REFERENCES

[1] 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.

[2] J. Ang, K. Evans, A. Geist, M. Heroux, P. Hovland, O. Marques, L. C. McInnes, E. Ng, and S. Wild, "Workshop on extreme-scale solvers: Transition to future architectures," DOE, Tech. Rep., 2012.

[3] 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.

[4] 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.

[5] J. Dongarra, J. Hittinger, J. Bell, L. Chacon, R. Falgout, M. Heroux, P. Hovland, E. Ng, and C. W. S. Wild, "Applied mathematics research for exascale computing," DOE, Tech. Rep., 2014.

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

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

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

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

[10] M. R. Hestenes and E. Stiefel, "Methods of Conjugate Gradients for Solving Linear Systems," *Journal of Research of the National Bureau of Standards*, vol. 49, pp. 409–436, Dec. 1952.

[11] H. A. van der Vorst, "Bi-cgstab: A fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems," *SIAM J. Sci. Stat. Comput.*, vol. 13, no. 2, pp. 631–644, Mar. 1992. [Online]. Available: http://dx.doi.org/10.1137/0913035

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

[13] 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

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

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

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

[17] J. Brannick, R. Brower, M. Clark, J. Osborn, and C. Rebbi, "Adaptive Multigrid Algorithm for Lattice QCD," *Phys.Rev.Lett.*, vol. 100, p. 041601, 2008.

[18] J. W. Ruge and K. Stüben, "Algebraic multigrid," in *SIAM Frontiers on Applied Mathematics, Volume 3, Multigrid Methods*, S. F. McCormick, Ed. SIAM, 1987, pp. 73–130.

[19] M. Brezina, R. Falgout, S. MacLachlan, T. Manteuffel, S. McCormick, and J. Ruge, "Adaptive algebraic multigrid," *SIAM Journal on Scientific Computing*, Submitted September 7, 2004.

[20] O. E. Livne and A. Brandt, "Lean algebraic multigrid (lamg): Fast graph laplacian linear solver," *SIAM Journal of Scientific Computing*, 2011, accepted. [Online]. Available: http://arxiv.org/abs/1108.1310v1

[21] P. Vaidya, "Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners. Unpublished manuscript UIUC 1990. A talk based on the manuscript was presented at the IMA Workshop on Graph Theory and Sparse Matrix Computation, October 1991, Minneapolis," 1990.

[22] A. Joshi, "Topics in optimization and sparse linear systems," Ph.D. dissertation, University of Illinois at Urbana-Champaign, Champaign, IL, USA, 1997, uMI Order No. GAX97-17289.

[23] D. Chen and S. Toledo, "Vaidya's preconditioners: Implementation and experimental study," *Electronic Transactions on Numerical Analysis*, vol. 16, pp. 30–49, 2003.

[24] D. A. Spielman and S.-H. Teng, "Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems," in *STOC*, L. Babai, Ed. ACM, 2004, pp. 81–90.

[25] ——, "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.

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

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

[28] G. E. Blelloch, A. Gupta, I. Koutis, G. L. Miller, R. Peng, and K. Tangwongsan, "Near linear-work parallel SDD solvers, low-diameter decomposition, and low-stretch subgraphs," in *SPAA*, R. Rajaraman and F. Meyer auf der Heide, Eds. ACM, 2011, pp. 13–22.

[29] 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.

[30] 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

[31] U. Naumann and O. Schenk, *Combinatorial Scientific Computing*, 1st ed. Chapman & Hall/CRC, 2012.

[32] P. G. Doyle and J. L. Snell, "Random walks and electric networks," 2006.

[33] M. W. Bern, J. R. Gilbert, B. Hendrickson, N. Nguyen, and S. Toledo, "Support-graph preconditioners," *SIAM J. Matrix Analysis Applications*, vol. 27, no. 4, pp. 930–951, 2006. [Online]. Available: http://dx.doi.org/10.1137/S0895479801384019

[34] K. D. Gremban, B. Maggs, and O. Ghattas, "Combinatorial preconditioners for sparse, symmetric, diagonally dominant linear systems," School of Computer Science, Carnegie Mellon University, Tech. Rep., 1996.

[35] I. Koutis and G. L. Miller, "Graph partitioning into isolated, high conductance clusters: Theory, computation and applications to preconditioning," in *Proceedings of the Twentieth Annual Symposium on Parallelism in Algorithms and Architectures*, ser. SPAA '08. New York, NY, USA: ACM, 2008, pp. 137–145. [Online]. Available: http://doi.acm.org/10.1145/1378533.1378559

[36] L. Grady, "A lattice-preserving multigrid method for solving the inhomogeneous poisson equations used in image analysis," in *Computer Vision ECCV 2008*, ser. Lecture Notes in Computer Science, D. Forsyth, P. Torr, and A. Zisserman, Eds. Springer Berlin Heidelberg, 2008, vol. 5303, pp. 252–264.

[37] J. A. Kelner, L. Orecchia, A. Sidford, and Z. A. Zhu, "A simple, combinatorial algorithm for solving SDD systems in nearly-linear time," in *STOC*, D. Boneh, T. Roughgarden, and J. Feigenbaum, Eds. ACM, 2013, pp. 911–920.

[38] I. Koutis, "Combinatorial and algebraic tools for multigrid algorithms," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, May 2007, cMU CS Tech Report CMU-CS-07-131.

[39] L. N. Trefethen and D. Bau, *Numerical linear algebra*. SIAM, 1997.

[40] "Combinatorial multigrid software package," http://ccom.uprrp.edu/ ikoutis/cmg.html.

[41] 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.

[42] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," *ACM Trans. Math. Softw.*, vol. 38, no. 1, pp. 1:1–1:25, Dec. 2011. [Online]. Available: http://doi.acm.org/10.1145/2049662.2049663

[43] T. A. Davis, "SuiteSparse," http://faculty.cse.tamu.edu/davis/suitesparse.html, 2014.

[44] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley, "An overview of the trilinos project," *ACM Trans. Math. Softw.*, vol. 31, no. 3, pp. 397–423, 2005.