Components of a more robust multilevel solver for emerging architectures and complex applications

Luke N. Olson*
Department of Computer Science, University of Illinois at Urbana-Champaign
Email: lukeo@illinois.edu
URL: http://www.cs.illinois.edu/homes/lukeo

Jacob B. Schroder
Department of Applied Mathematics, University of Colorado at Boulder
Email: jacob.schroder@colorado.edu
URL: http://grandmaster.colorado.edu/~jacob/

Abstract. Computational simulation in the physical and information sciences continues to place demands on the underlying linear solvers used in the process. Algebraic-based multigrid methods offer a flexible medium for adapting to a wide range of problems, yet traditional approaches have designed the multigrid components for basic, isotropic, and well-behaved phenomena. As a result, out-of-the-box multilevel preconditioners do not handle a wide range of problems. In this article, we highlight several components of the process that are redefined in order to generalize the approach to more variable, anisotropic, asymmetric, and unexpected behavior.

1 Introduction

The solution to large, sparse linear systems represents a major portion of the computation time in many simulations. This *solve* may be in the form of a final static solution, a solution step within a Newton iteration of a nonlinear problem, a central component of a fully or partially implicit time stepping scheme, as part of an eigenspectrum calculation, and so forth. Not surprisingly, as computing power continues to increase, we also see a growth in complexity for both applications and ultimately the underlying numerical methods used to simulate the process. As a result, increased demands are placed on the linear solvers, requiring theoretical developments that lead to more robust methods and necessitating more flexible algorithms to handle the emerging architectures and complex physics present in cutting-edge scientific simulations.

New simulations insist on solvers that are designed to handle the challenging attributes of new problems and new discretizations. Wave problems, highly variable and anisotropic flows, and nonuniform materials properties in solid mechanics or electromagnetics are commonplace in simulations, yet many of the paradigms used in designing solution methods cannot handle these situations. Applications are pushing the size of the sparse linear systems to unprecedented levels, necessitating careful design but flexible solvers.

The continual growth in application complexity naturally results in more advanced discretization techniques. High-order discretizations, for example, have shown broad applicability and value in recent years, while at the same time placing higher demands on the solver. Mimetic discretizations such as curl-conforming Whitney elements, nonconforming discretizations such as used in discontinuous Galerkin finite-element methods, and hybrid discretizations such as multiscale methods all similarly present new challenges to the underlying solvers. However, in order for these highly effective discretization techniques to be useful, they must be designed concurrently with a competitive solver.

Emerging architectures are also redefining the role and impact of solvers in the simulation toolchain. High-performance computing now extends beyond the traditional supercomputing environment and encompasses highly heterogeneous settings such as smaller scale GPU accelerated workstations and clusters, as well as distributed cloud-based systems. Applications in these settings rely heavily on linear solvers; and it is evident that, as solvers develop, they should take advantage of the computational context.

As applications, mathematics, and hardware converge in solver and linear algebra kernel development, the software is also becoming increasingly more critical and important. Development time is becoming more complex; usability by a wider audience is a necessity; and portability is turning into a large asset. There is a need for solver development to become more accessible, extensible, and flexible as the methodology progresses.

In this paper, we outline several current directions being pursued to rethink and abstract the methods, algorithms, and implementations of linear solvers. What we seek is a more robust, efficient, and principled solver, which points toward an algebraic-based multigrid approach that is compatible with existing Krylov iterative methods. However, traditional algebraic multigrid (AMG) methods [10, 12] are designed for *basic* problems, do not scale well, and often have no sense of optimality despite having a strong theoretical basis.

We outline strategies for handling basic structural qualities, such as complex or non-symmetric systems. We also review some fundamental elements of the multigrid process such as strength of connection in the matrix graph and interpretation of interpolation that help build a more competitive multigrid cycle. Additionally, we highlight several directions where multigrid is successfully adapted to the computing environment, such as the GPU, and adapted to atypical application areas through accessible software.

The future is that the mathematical design and theoretical development of solvers will necessarily be tightly coupled to the underlying platform and the intended application scope. The *best* approach in large-scale scientific and information applications requires consideration of the total computational cost, and linear solver development continues to play a critical role.

1.1 Basic Algorithms

The efficiency of algebraic multigrid as a scalable solver relies on the complementary relationship between relaxation and coarse-grid correction. Relaxation quickly attenuates high-energy error (i.e., error with a large A-norm) on the fine grid, leaving behind low-energy error. This lower-dimensional space of low-energy functions is then captured by interpolation, which maps (i.e., restricts) the residual equation to a coarse-grid composed of this low energy space. The coarse-grid correction is then interpolated (i.e., prolongated) to the fine grid, where the current fine-grid solution guess is updated.

The complementary nature of relaxation and coarse-grid correction are summarized through the setup and solve phases in Algorithms 1 and 2, respectively. We refer to these algorithms in the following sections, noting the important changes as we extend the robustness of algebraic multigrid. In particular, we extend the applicability to a broader class of problems by refining how the low-energy modes of the problem are formed (Line 1, Algorithm 1), how we determine whether certain nodes are influential in the matrix graph (Line 2, Algorithm 1), and how the accuracy of interpolation is optimized (Line 4, Algorithm 1). Moreover, these extensions more naturally allow for complex or non-symmetric preconditioning by accounting for the impact on interpolation (Line 5, Algorithm 1) and relaxation (Lines 1 and 3, Algorithm 2). Furthermore, we also customize our methods to more general problem domains and discretizations by considering our cycle (Line 2, Algorithm 1) and accelerating components in the architectures such as aggregation (Line 3, Algorithm 1) or the triple-matrix product (Line 6, Algorithm 1).

Algorithm 1: AMG Setup

input: Sparse matrix: A, Low-energy modes: B
return: Hierarchy: $A_0, \ldots, A_k$, $P_0, \ldots, P_{k-1}$, $R_0, \ldots, R_{k-1}$

$A_0 = A$, $B_0 = B$, $k = 0$
while $k > maxlev$ *and* $size(A_k) < maxcoarse$
2 $C_k = \texttt{strength}(A_k)$ {strength-of-connection}
3 $Agg_k = \texttt{aggregate}(C_k)$ {construct aggregates}
$T_k,\ B_{k+1} = \texttt{tentative}(C_k, B_k)$ {form tentative}
4 $P_k = \texttt{prolongator}(A_k, T_k)$ {smoothed prolongator}
5 $R_k = P_k^T$
6 $A_{k+1} = R_k A_k P_k$ {coarse matrix, triple-matrix product}
7 $k = k + 1$

Algorithm 2: AMG Solve

input: Hierarchy: $A_k, R_k, P_k, G_k, u_k, f_k$
return: Solution: u_k

if $k = maxlev$
solve $A_k u_k = f_k$ {coarsest-grid solve}
else
1 $u_k \leftarrow G_k^{\mu_1}(u_k)$ {smooth μ_1 times on $A_k u_k = f_k$}
$r_k = f_k - A_k u_k$ {compute residual}
$r_{k+1} = R_k r_r$ {restrict residual}
2 $e_{k+1} \leftarrow \texttt{solve}(A_{k+1}, R_{k+1}, P_{k+1}, G_{k+1}, r_{k+1})$ {recursive call}
$e_k = P_k e_{k+1}$ {interpolate error correction}
$u_k \leftarrow u_k + e_k$ {correct solution}
3 $u_k \leftarrow G_k^{\mu_2}(u_k)$ {smooth μ_2 times on $A_k u_k = f_k$}

2 Building Blocks

2.1 Low-energy

Central to the compatibility of relaxation and coarse-grid correction in Algorithm 2 is the ability of interpolation to capture the low-energy modes not annihilated by relaxation. The basic smoothed aggregation setup algorithm in Algorithm 1 requires knowledge of the *near null-space* or lowest energy modes, B, for the system $Ax = b$. Often these are known a priori through information about the problem.

However, in many instances the a priori description of the near null-space is inadequate, by not capturing the character near the boundary or not reflecting the inaccuracy of the discrete problem. To this end, a simple, yet effective, correction is to execute several passes of relaxation prior to the setup phase on each level in order to reduce the energy of B. If the matrix does not define a norm—for example, in the case of indefinite problems—then we define energy in the A^*A-norm, which is defined below [4]. The prerelaxation to improve the near null-space modes then reduces the A-norm or the A^*A-norm of each column $B^{(j)}$ of B using a relaxation method such as Gauss-Seidel on $AB^{(j)} = 0$ or on the normal equations $A^*AB^{(j)} = 0$, which is accomplished without forming the matrix product. An example is given in Figure 1 for an

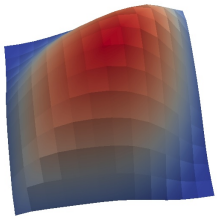

Figure 1: Prerelaxing user-provided constant vector for an anisotropic problem.

anisotropic diffusion problem rotated away from the coordinate axis. A constant vector $B = 1$ is provided, which is naturally isotropic; however, after relaxation, we see that the low-energy modes incorporate the problem data of anisotropy, yielding a more accurate representation of the near null-space. We incorporate this refinement of low-energy modes in Line 1 of Algorithm 3, a modified version of a more robust algebraic setup phase.

2.2 Interpreting Connections

The first two steps of the setup phase in Algorithm 1—namely, identifying the strength-of-connection between nodes in the matrix graph (Line 2) and grouping like-minded nodes between which low-energy error is strongly correlated in the matrix graph (Line 3)—have a tremendous impact on the remaining components and ultimate performance of the multigrid method. Misclassifying strength leads to unnecessary complexity and often limits the potential accuracy of interpolation by allowing the aggregation of poorly collaborating nodes.

The traditional approach to identifying the fitness of a node for aggregation is to simply view the weights in the matrix graph: nodes i and j are strongly connected (and are allowed to collapse to one node on the coarse grid) if $|A_{ij}| \geq \theta\sqrt{A_{ii}A_{jj}}$, where $\theta \in [0, 1]$. As identified in [7], a more general approach is to consider the effect of relaxation around a node and to relate this process to the expectations governed by the low-energy modes B:

Step 1: Evolve a point source (i.e., δ-function) centered at node i with relaxation on the matrix graph.

Step 2: Determine the flow of the point source from i to neighbors j in comparison to the low energy modes B.

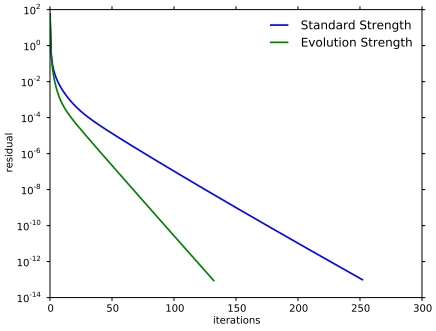

Figure 2: Impact of correct strength-of-connection.

Step 1 is executed globally with efficient operations such as a restricted matrix product. If we consider e to be the result of relaxing a point source at i across the graph, then we assess the ability of interpolation to interpolate e in the neighborhood of i, given that the span of B approximates the range of interpolation. That is, we locally let

$$S_{ij} = \left| \frac{e_j - e_j^B}{e_j} \right|, \tag{1}$$

where e_j is the value of the point source spread to node j and e_j^B is the value at node j of the projection of e into span$\{B\}$. This measures the potential ability of interpolation to effectively interpolate between nodes i and j for *general* low-energy modes, a critical component in extending multigrid to more complicated matrix problems. Moreover, as a drop-in replacement the "Evolution" measure is straightforward to incorporate into current multigrid codes — e.g. in Line 2 of a modified setup phase in Algorithm 3.

As an example of the impact of correctly identifying these decisions early in the process, consider the case of diffusion rotated away from the coordinate access by $\pi/8$ and weighted by four orders of magnitude in the x-direction. A convergence history of residual norms for the solver is shown in Figure 2, which highlights a common behavior: a correct identification of strength leads to significant improvement for non-isotropic or non-elliptic problems.

2.3 Optimality

Given a reliable description of strongly connected components in the matrix graph and an accurate representation of the low energy modes B, an interpolation operator with a range consistent with the low-energy modes is immediately available by injecting B onto the sparsity pattern formed by the aggregates (from the list of strongly connected nodes). However, accurate interpolation requires additional information, leading to overlap in the interpolation stencil, thus increasing the complexity [12].

Improving interpolation while maintaining complexity has long been a goal in multigrid methods, and there have been a number of successes such as postrelaxation and direct energy minimization. The approach taken in [9] attempts to force accurate interpolation for low-energy modes over a preset sparsity pattern, again using a general energy concept in either the A-norm or the A^*A-norm. The main heuristic is that interpolation P should have low-energy column-wise, such that a sparsity pattern is fixed for P. That is, we find P such that

$$AP_j = 0 \quad \text{for each column j.} \tag{2}$$

Thus, by using a Krylov method such as conjugate gradient (CG) for symmetric positive definite A or generalized minimum residual (GMRES) for general A to solve (2), we naturally find P with

$$\|P_j\|_A \to \min \quad \text{and} \quad \|P_j\|_{A^*A} \to \min, \tag{3}$$

respectively. In order to guarantee a nontrivial solution, however we constraint the process so that interpolation is consistent with the near null-space. That is, (3) is constrained with $PB_{coarse} = B$.

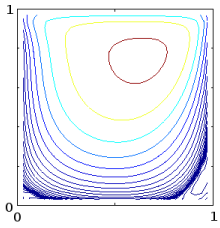

Figure 3: Streamline driven-cavity with $Re = 1000$.

Additionally, with a general strength matrix, S, the sparsity pattern is allowed to effectively *grow* along directions of strong connection through

$$P_{new} = S^j P \quad \text{for} \quad j \in \{1, 2, \dots\}. \tag{4}$$

This allows control of two critical portions of the interpolation process: the *complexity* of the interpolation operator and the *accuracy* of the interpolation operator. As a result, the multigrid process is much more robust because we force accuracy on interpolation, while intelligently controlling the sparsity pattern, leading to a more effective, yet richer multigrid hierarchy. Moreover, the process relies on fast, basic linear kernels such as the matrix-vector multiply and several graph methods.

As an example, consider the non-Hermitian lid-driven cavity with $Re = 1000$ given by the streamline solution in Figure 3. We see in Table 1 that alone results in an improved solver, yet the performance is limited because interpolation is inaccurate either due to the definition of interpolation or due to an inadequate sparsity pattern induced by the coarse nodes chosen through the strength routine. The improved interpolation strategy is highlighted in Line 4 of a modified setup phase in Algorithm 3.

Table 1: Preconditioned GMRES iterations for a lid-driven cavity with $h = 1/256$.

Standard	With Improved Strength	With Improved Interpolation	With Improved Strength and Interpolation
75	42	41	18

2.4 Non-symmetric and complex

A fundamental concept in the traditional approach to multigrid methods is that of the Galerkin product $A_{coarse} = RAP$. If the coarse matrix is expected to be symmetric, then $R = P^*$ is the natural choice. However, this is problematic for nonsymmetric problems. Instead, since the range of interpolation is expected to conform to the low-energy, *right* near null-space, the range of R^* is expected to represent the low-energy *left* near null-space—that is, the near null-spaces of A and A^*, respectively. This has been explored recently [4, 6, 11, 8], so that restriction R is defined to be the adjoint of interpolation for the adjoint of the operator A. That is, the interpolation strategies of the previous section are enforced on R^* through A^*, so that $A^*R^* \approx 0$ is enforced with left, low-energy modes B^*.

It is also important to distinguish the concept of *energy* in the general case. We assume that there are two spaces of low-energy functions,

$$B = \{e \,:\, \|Ae\|_0 = \|e\|_{A^*A} \approx 0\} \tag{5}$$

$$B^* = \{e \,:\, \|A^*e\|_0 = \|e\|_{AA^*} \approx 0\}, \tag{6}$$

which are the right and left *singular* vectors corresponding to small singular values, respectively, and are used to define algebraically smooth error. Moreover, in combination with Gauss-Seidel or other relaxation methods based on the normal equations A^*A or AA^*—otherwise termed Kazmarcz smoothing — these descriptions of the low-energy modes are compatible with the multigrid process [4]. In the modified setup phase of Algorithm 3 we incorporate this notion of dual low-energy in Line 1, by generating complementary left near null-space modes, and in Lines 4 and 5 by considering energy minimization for the construction of both P and R^*.

3 Accelerating the Solver

While applications continue to demand more robust *convergence* of the linear solvers in the simulation process, the advanced techniques discussed above need to conform to and take advantage of the underlying computing architecture. This is particularly crucial as we anticipate more everyday use of heterogeneous architectures, for example with acceleration units. GPU-enabled boxes are readily available and accessible to computational programming [2], and other architectures are going in this directions, including AMD Fusion accelerated processing units (APUs).

The major difficulty in taking advantage of these highly parallel architectures, however, is that our sparse matrix computations with multigrid are inherently *memory* bound. However, if we redesign the multigrid components and specializations to utilize efficient computing kernels on these accelerated architectures, then tangible speed-ups are achieved. The key is to incorporate the kernels, such as a sparse matrix-vector multiply, a fast sort-by-key, or an efficient reduction, into the mathematics as we extend the robustness of multigrid.

Exposing fine-grained parallelism in many of the components of Algorithms 1 and 2 is not straightforward, yet expressing the parallelism in terms of the highly tuned primitives of the

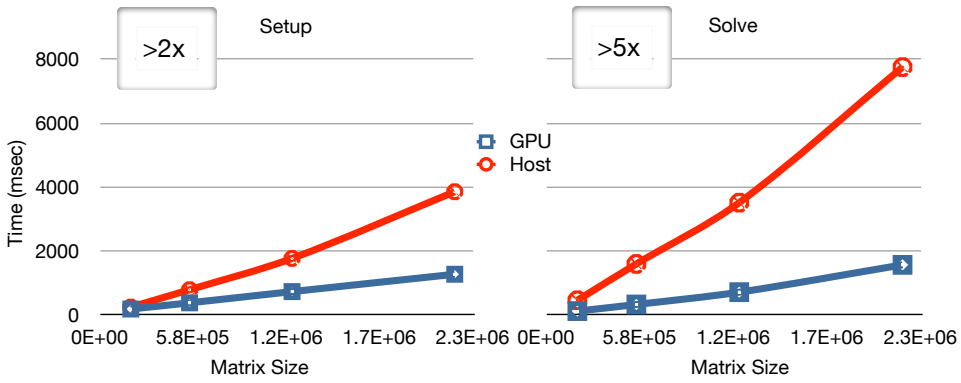

Figure 4: Timings and speedup for algebraic multigrid on the GPU.

architecture leads to valuable acceleration of the multigrid method [3] and a sustained way in which we approach multilevel solvers. Consider Algorithms 1 and 2 on an NVidia GPU. Central to performance on the NVidia GPU is our ability to coalesce the memory access of the sparse matrix computations [1] and to express the methods in terms of a gather, scan, sort, reduce, and other tuned routines on the GPU. Figure 4 summarizes a typical timing and speedup for the setup and solve (cycling) phases of algebraic multigrid. Typical speedups are in the area of 2× for the setup phase, while 5× for the solve phase.

With strength decisions, aggregation, interpolation, interpolation smoothing, and the triple matrix product, the setup phase of Algorithm 1 has several components that require adapting to a more parallel environment. The cycling phase relies almost exclusively on the sparse matrix-vector product, resulting in a direct benefit given a multigrid hierarchy with modest complexity. The limitation of a 2× speedup for a single GPU device is largely due to the dominance of the triple matrix produce in Line 5 of Algorithm 1, which is inherently sequential. Figure 5 shows that most of the time at each level of the setup phase is spent assembling the triple matrix product, which is common for algebraic-based multigrid methods. While we achieve up to 2× speedup for the matrix product on the GPU, there is a need to develop more mathematical tools to alleviate the setup phase of an *exact* triple matrix product *RAP*, by considering either multiple, thinner coarsenings or by approximating the coarse level matrix through another approach. We also observe that as the coarsening proceeds, the sparse matrix operations become more dense, making them less suitable for the GPU. Indeed, the speedup at finer (and larger) levels of the hierarchy is greater than exhibited at coarser levels, which points to a need for a hybrid cycling and setup routine that can automatically handle CPUs and GPUs more effectively for different components in the multigrid process.

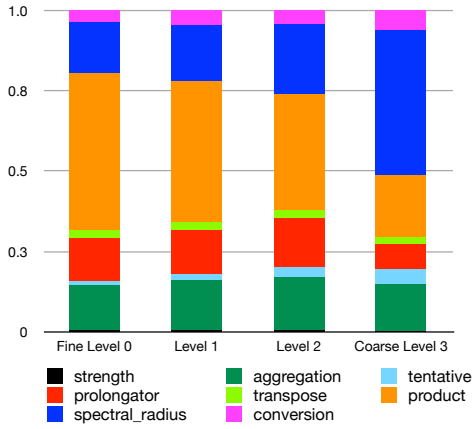

Figure 5: Timings at each level of the setup phase.

Exposing Parallelism: An Example of Aggregation

The traditional approach to aggregation is straightforward: sequentially group ungrouped nodes that are strongly connected. The process proceeds as follows:

Step 1: Select each node i. If node i has not been placed in a group (aggregate) and the strongly connected neighbors of i have not been placed in a group, then form a new group.

Step 2: For each node i not placed in a group from Step 1, either form a new group or add to a neighboring group.

This aggregation technique has several limitation including sequentiality and the inability to adjust the size of the aggregate in order to control complexity.

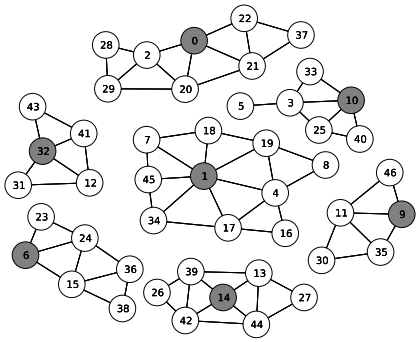

Figure 6: Example grouping of fine nodes of the matrix graph.

A typical collection of aggregates or groups of fine nodes is given in Figure 6. Here, we see that the root nodes for the aggregates (shaded gray) are more than two steps away from other root nodes in the graph and that if an unaggregated node were more than two steps away, it could safely be aggregated with its neighboring nodes. This defines the concept of a distance-2 maximal independent set (MIS-2), and governs the independence and maximality of the covering respectively. Thus, with an MIS-2 splitting of N root nodes, we generate an aggregate based on the primitives of the architectures, which we assume includes a sparse matrix-vector multiply (SpMV). We first enumerate the N root nodes in a list (giving an aggregate index) and proceed as follows (Line 3 of Algorithm 3):

Step 1: Communicate the aggregate index to strongly connected neighbors: SpMV with S.

Step 2: Communicate the aggregate index to additional unaggregated neighbors: SpMV with S.

The result is a highly efficient aggregation technique that mimics the standard approach up to a permutation of the index list, relies on the fast SpMV optimized for the architectures, and is extensible to larger aggregates as described in [3].

4 The Next Generation of Solvers

We have presented a more generalized setup phase in Algorithm 3 that attempts to produce a more robust solver. Yet, we have certainly not uncovered the full potential of generalizing the multigrid method. For example, consider the 23 test cases in Figure 7, where we have applied a version of the modified setup phase in Algorithm 3 to a collection of matrices from the University of Florida Sparse Matrix Collection [5]. We reduce the residual by 10^{-9}, and symmetric, positive definite (s.p.d.) systems are solved using the preconditioned CG framework for interpolation above, while general systems use preconditioned GMRES. The results show a consistent benefit as a preconditioner. All s.p.d. systems resulted in an effective preconditioner, however the nonsymmetric systems benefited only in 8 out of 40 cases, highlighting that general systems require a much broader interpretation of the multigrid hierarchy.

This result emphasizes the need for more theoretical development of multigrid methods for general systems. A competitive framework will seamlessly incorporate the following features:

Algorithm 3: AMG Modified Setup

input: Sparse matrix: A, Low-energy modes: B
return: Hierarchy: $A_0, \ldots, A_m$, $P_0, \ldots, P_{m-1}$

$A_0 = A$, $B_0 = B$, $k = 0$
while $k > maxlev$ *and* $size(A_k) < maxcoarse$

1 B_k, $B_k^* \leftarrow$ `improve_modes`(B_k, B_k^*) `{refine low-energy}`
2 $C_k =$ `evolution_strength`(A_k) `{strength-of-connection}`
3 $Agg_k =$ `parallel_aggregates`(C_k) `{construct aggregates}`

4 P_k, $B_{k+1} =$ `optimize_interpolation`(A_k, Agg_k, B_k) `{construct interpolation}`
5 R_k, $B_{k+1}^* =$ `optimize_restriction`(A_k^*, Agg_k, B_k^*) `{construct restriction}`

6 $A_{k+1} = R_k A_k P_k$ `{coarse matrix, triple-matrix product}`
7 $k = k + 1$

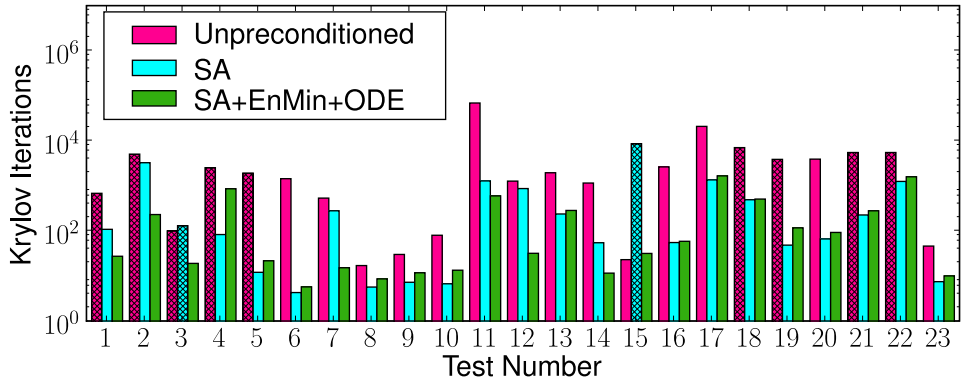

Figure 7: Iterations to convergence for a sample of matrices from the University of Florida Sparse Matrix Collection using Algorithm 3 as a blackbox preconditioner.

.

user-provided knowledge of the problem Many of the problems can benefit from user data such as a description of the physics for low-energy modes, or the nature of the degrees-of-freedom in high-order or discontinuous discretizations, or the topology of the problems, such as dimension.

automatically coarsen For many problems, the coarse-grids are not rich enough to host accurate interpolation, thus requiring a reassessment of the construction of coarse-grid connections.

optimally construct a hierarchy An optimal interpolation strategy will seek an optimal compatibility between left and right relaxation, left and right near null-space modes, and interpolation and restriction.

multiple cycling For many problems, a single view of the hierarchy is not realistic, particularly when the near null-space is rich or when the left and right energy of the problem provide diverging views of the problem.

self-correction A multilevel process is inherently a global construction, requiring many passes to fine-tune the method.

Moreover, this mathematical development requires careful integration with the intended computing kernels in order to yield a competitive solver with respect to total computational cost. Coarse-grid selection, interpolation strategies, correction methods, and cycling options need to effectively harness the available computing power. Given the flexibility of the multigrid framework it is poised to be a powerful solution method for many more applications as robustness continues to progress.

References

[1] N. Bell and M. Garland, *Efficient sparse matrix-vector multiplication on CUDA*, NVIDIA Technical Report NVR-2008-004, NVIDIA Corporation, Dec. 2008.

[2] ——, *CUSP : Generic parallel algorithms for sparse matrix and graph computations*. `http://code.google.com/p/cusp-library/`, 2009-.

[3] W. N. Bell, S. Dalton, and L. N. Olson, *Amg on the gpu*, in preparation, (2011). see cusp.precond at http://code.google.com/p/cusp-library/.

[4] M. Brezina, T. Manteuffel, S. Mccormick, J. Ruge, and G. Sanders, *Towards adaptive smoothed aggregation (αSA) for nonsymmetric problems*, SIAM J. Sci. Comput., (Submitted 2009).

[5] T. A. Davis, *The University of Florida sparse matrix collection.* `http://www.cise.ufl.edu/research/sparse/matrices`. Submitted to ACM Transactions on Mathematical Software.

[6] S. P. MacLachlan and C. W. Oosterlee, *Algebraic multigrid solvers for complex-valued matrices*, SIAM J. Sci. Comput., 30 (2008), pp. 1548–1571.

[7] L. N. Olson, J. Schroder, and R. S. Tuminaro, *A new perspective on strength measures in algebraic multigrid*, Numerical Linear Algebra with Applications, 17 (2010), pp. 713–733.

[8] L. N. Olson and J. B. Schroder, *Smoothed aggregation for Helmholtz problems*, Numer. Linear Algebra Appl., 17 (2010), pp. 361–386.

[9] L. N. Olson, J. B. Schroder, and R. S. Tuminaro, *A general interpolation strategy for algebraic multigrid using energy minimization*, SIAM Journal on Scientific Computing, 33 (2011), pp. 966–991.

[10] J. W. Ruge and K. Stüben, *Algebraic multigrid (AMG)*, in Multigrid Methods, S. F. McCormick, ed., Frontiers Appl. Math., SIAM, Philadelphia, 1987, pp. 73–130.

[11] M. Sala and R. S. Tuminaro, *A new Petrov-Galerkin smoothed aggregation preconditioner for nonsymmetric linear systems*, SIAM J. Sci. Comput., 31 (2008), pp. 143–166.

[12] P. Vaněk, J. Mandel, and M. Brezina, *Algebraic multigrid based on smoothed aggregation for second and fourth order problems*, Computing, 56 (1996), pp. 179–196.