Scalable Lazy-update Multigrid Preconditioners

Majid Rasouli, Vidhi Zala, Robert M. Kirby, Hari Sundar
School of Computing
University of Utah
Salt Lake City, Utah, USA
rasouli@cs.utah.edu, vidhi@sci.utah.edu, kirby@sci.utah.edu, hari@cs.utah.edu

***Abstract*—Multigrid is one of the most effective methods for solving elliptic PDEs. It is algorithmically optimal and is robust when combined with Krylov methods. Algebraic multigrid is especially attractive due to its blackbox nature. This however comes at the cost of increased setup costs that can be significant in case of systems where the system matrix changes frequently making it difficult to amortize the setup cost. In this work, we investigate several strategies for performing lazy updates to the multigrid hierarchy corresponding to changes in the system matrix. These include delayed updates, value updates without changing structure, process local changes, and full updates. We demonstrate that in many cases, the overhead of building the AMG hierarchy can be mitigated for rapidly changing system matrices.**

***Index Terms*—Algebraic Multigrid, AMG, Iterative Solver, Sparse, Preconditioned Conjugate Gradient, Preconditioner**

I. Introduction

Elliptic operators are ubiquitous in science and engineering, and solving elliptic PDEs efficiently is a fundamental problem in computational science and engineering. For large-scale systems that need to be solved in parallel, iterative solvers with $\mathcal{O}(n)$ complexity and mesh-independent convergence are preferred. A good example of a mesh-independent solver and preconditioner is the multigrid methods when applied to matrices generated from the discretizations of elliptic operators [1]–[3]. The mathematical predictability of the benefits of the multigrid approach combined with the ability to exploit the regularity of the data structures (in terms of indexing, coarsening, etc.) has made multigrid, especially in conjunction with preconditioned conjugate gradient (PCG) [4], [5], the solver of choice when solving engineering applications that require large-scale parallel solution approaches.

Algebraic Multigrid (AMG) in particular is extremely attractive due to its ability to work directly with the discretized operator without additional knowledge about the discretization or meshing. While AMG is very attractive due to its blackbox nature [6]–[8], it does have significantly higher setup costs compared to its geometric counterpart [9]. For many problems, this added setup cost is not a problem as this is amortized over several solves. However in many problems, such as our target problem §II, the operator changes with time and we only perform a single solve for a given operator. In such cases, the added cost of setup and a single solve is inefficient and highly wasteful. In this work, we consider several strategies to address this problem. Specifically, we consider variants that are able to amortize the cost of AMG setup across multiple solves for cases where the operator changes frequently. We consider both sequential and distributed scenarios and demonstrate effective amortization of the AMG setup.

The rest of the paper is organized as follows. In the next section, we motivate our methods by a brief description of our target problem, requiring the development of our methods and describe our methods in §III followed by experiments demonstrating the effectiveness of our approach in §IV. Finally, we conclude with directions for future research in §V.

II. Target Problem

Chemical transport is a highly interesting phenomenon in a variety of fields like environmental pollutants tracking, biological processes like blood clotting and industrial applications like solvent manufacturing. To study this effectively, we need accurate models that track and predict these chemicals. Many of these applications often involve more than one chemical species which move and interact among themselves. As such, modeling of these processes highly depend on the accuracy with which we can track the interplay between these chemicals. Variation in the chemical population often occurs because of movement and reactions among themselves, thus creating more chemical species or destroying the existing ones. Due to the complex nature of the problem, the advection-diffusion equation

$$\frac{\partial \boldsymbol{c}}{\partial t} = -U\frac{\partial \boldsymbol{c}}{\partial x} + D\frac{\partial^2 \boldsymbol{c}}{\partial x^2}, \tag{1}$$

is often used to obtain a numerical approximation of the exact location and population of the chemical species. Here, D refers to the diffusion coefficient and U refer to the flow velocity. It is equally important that we track the accuracy of these models while tracking the chemical species. One of the most common ways to ensure accuracy is to compare the numerical solution against the analytical solution. The advection-diffusion equation is being extensively studied and analytically solved for the case of constant diffusion coefficient, with uniform flow [10], [11]. However, many naturally occurring processes are more involved than the constant diffusion coefficient or uniform flow can model. One such process is described in the next section.

A. Variable Diffusion case in Elliptic PDE

We build the model problem of variable diffusion and motivate the application of multigrid preconditioning methods to numerically solve the resulting equation. Consider the set of

978-1-7281-5020-8/19/$31.00 ©2019 IEEE

equation that arise out of the modeling of chemical transport and interactions resulting from the biological process of blood clotting (specifically, *thrombosis*), popularly knows as the Leiderman-Fogelsen model [12], [13].

Depending on the roles they play in the process, the interesting quantities of this model can be divided into the following classes:

1) Mobile unactivated ($P^{m,u}$),
2) Mobile activated ($P^{m,a}$),
3) Platelet-bound activated ($P^{b,a}$)
4) Subendothelium-bound activated ($P^{se,a}$)

$$\begin{aligned}\frac{\partial P^{m,u}}{\partial t} = &\underbrace{-\nabla\cdot\{W(\phi^T)(\boldsymbol{u}P^{m,u} - D\nabla P^{m,u}))\}}_{\text{Transport by advection and diffusion}}\\ &\underbrace{-k_{adh}(\boldsymbol{x})\{P_{max} - P^{se,a}\}P^{m,u}}_{\text{Adhesion to subendothelium}}\\ &\underbrace{-\{A_1(e_2) + A_2([ADP])\}P^{m,u}}_{\text{Activation by thrombin or ADP}} \end{aligned} \quad (2)$$

$$\begin{aligned}\frac{\partial P^{m,a}}{\partial t} = &-\nabla\cdot\{W(\phi^T)(\boldsymbol{u}P^{m,a} - D\nabla P^{m,a})\}\\ &-k_{adh}(\boldsymbol{x})\{P_{max} - P^{se,a}\}P^{m,a}\\ &+\{A_1(e_2) + A_2([ADP])P^{m,u}\\ &\underbrace{-k_{coh}g(\eta)P_{max}P^{m,a}}_{\text{Cohesion to bound platelets}}. \end{aligned} \quad (3)$$

$k_{adh}(\boldsymbol{x})$ is assumed to be a positive constant for points $\boldsymbol{x}$ within one platelet diameter of subendothelium and zero elsewhere. We define $W(\phi^T)$ as follows:

$$W(\phi^T) = tanh(\pi(1-\phi^T)), \quad (4)$$

where, $\phi^T = P^{se,a} + P^{b,a} + \frac{P_0}{P_{maxse}}(P^{m,u} + P^{m,a})$, P_{maxse} and P_{max} are a constants for maximum number density for platelets as per [12].

In order to numerically solve the model equations of the kind given by Equations (2) and (3), we need to vary the diffusion coefficient (depicted by $W(\phi^T)D$) per time-step. The resulting elliptical PDEs can be solved using the Finite Element Method (FEM). We use the high-order finite element software package Nektar++ [14] version 4.4.1 to solve the continuous Galerkin problem arising out of the changing diffusion coefficient per time-set. It provides an unsteady diffusion solver that expects the mesh file describing the geometry of the domain and the related solver parameters.

While multigrid methods, specifically algebraic multigrid (AMG) methods in conjunction with Krylov methods are very effective in solving Equations (2) and (3), the high setup cost associated with AMG makes it less attractive when the operator changes very rapidly (due to varying diffusion coefficient). In this work, we explore different strategies for mitigating the high setup cost, while still retaining the efficiency of multigrid. We find that by performing lazy updates for the AMG preconditioner, we can amortize the high setup costs without adversely affecting the convergence rate. We now describe our lazy-update variant to AMG preconditioners.

III. Methods

We start with a brief description of our AMG framework. Algebraic multigrid (AMG) has been a popular method for solving linear systems of elliptic partial differential equations, especially for large sparse systems. AMG can be used as either a solver or a preconditioner to solve the linear system

$$Ax = b \quad (5)$$

where, $A \in R^{n\times n}$, x and $b \in R^n$.

AMG consists of a setup and a solve phase. During the setup phase, *aggregation* is applied to the equivalent graph G of the matrix A. Every row of the matrix A is considered as a node in the graph G and there is an edge between nodes i and j if entry (i,j) is nonzero in A. Let us say there are n nodes in the graph G. By doing the aggregation on them, m nodes will be chosen as *roots* such that $m < n$ and the rest of the nodes of the graph will be assigned to them. For our implementation, we have used the *maximal independent set* approach (similar to [15]) as the aggregation method.

The prolongation matrix $P \in R^{n\times m}$ will then be defined based on this aggregation. If node i is assigned to root j, then $P(i,j) = 1$, otherwise it is 0. The restriction operator $R \in R^{m\times n}$ is then made by transposing P. The prolongation operator is used in two ways. It interpolates a vector $v \in R^m$ to $v' \in R^n$, such that $m < n$. In other words, it takes vectors from a coarser grid to a finer one. The restriction operator does the reverse task; it takes $w \in R^n$ to R^m. The other purpose of P and R is creating a smaller version of the left-hand side matrix A:

$$A_c = R * A * P \quad (6)$$

such that $A_c \in R^{m\times m}$. This is called *coarsening*.

Progressively coarser versions of the matrix are created during the setup phase. An AMG hierarchy of $L+1$ levels consists of three categories of operators:

1) Coarse Matrices (As),
2) Prolongation Matrices (Ps), and
3) Restriction Matrices (Rs).

The coarse matrices are created similar to A_c for each level:

$$As[l+1] = Rs[l] * As[l] * Ps[l],\ l = 0, 1, 2, ..., L-1. \quad (7)$$

For this paper, *smoothed aggregation AMG (SA-AMG)* is used from [16], for which the prolongation (P_t) and restriction operators (R_t) are smoothed by

$$P = (I - \omega Q A^F)P_t, \quad R = R_t(I - \omega A^F Q) \quad (8)$$

where Q is a the inverse of the diagonal of A and ω is the damped Jacobi parameter and A^F is the filtered matrix of A. Smoothing the operators improves the convergence of AMG.

978-1-7281-5020-8/19/$31.00 ©2019 IEEE

The second step of AMG is the solve phase. To solve the linear system $Ax = b$, we start with an initial guess for x. Then, we use two methods to approximate the solution:

1) Relaxation (Jacobi, Chebyshev, ...), and
2) Coarse-grid Correction.

The setup phase is usually more expensive than the solve phase. We want to avoid doing at least some parts of the setup phase whenever it is possible, at the cost of a low increase in the solve time. This will be our goal in the three strategies that are explained in the next section.

A. AMG as a preconditioner

While AMG can be used directly as a solver for our target problem, using it as a preconditioner for the Conjugate Gradient method makes it far more robust, especially given the complexity of our target meshes. The pseudocode for using AMG as a preconditioner with CG is given in Algorithm 1.

Algorithm 1 Multigrid-preconditioned CG

Input: rhs and guess
Output: solution

```
while not converged do
    h = Ap
    ρ_r = (ρ, r)
    α = ρ_r/(p, h)
    u = u + αp
    r = r − αh
    Convergence Test
    ρ = Mr                          ▷ AMG V-cycle
    β = (ρ, r)/ρ_r
    p = ρ + βp
end while
```

As previously mentioned, the main downside of using AMG–with or without PCG–for our target problem is the variation of the diffusion coefficient. This effectively means that the high cost of the AMG setup is not offset by a sufficiently high number of corresponding solves. By using AMG as a preconditioner along with PCG, we can update the preconditioner–the multigrid hierarchy–in a lazy fashion, effectively lowering the effective cost of AMG setup. Of course using a *stale* preconditioner can be less efficient and can increase the number of iterations needed for convergence. We study this trade-off and consider three different strategies in order to get the overall best runtime. Note that the overall runtime will involve both the AMG setup as well as the cost of the PCG solves, therefore there is an incentive to reduce the number of AMG setups, even if it marginally increases the number of PCG iterations for convergence. We will now discuss the different strategies for lazy updates of the AMG hierarchy.

1) Strategy 1: Reuse the same AMG hierarchy: The first and simplest strategy is to simply use the same AMG hierarchy and only update the input matrix $As[0]$ with the updated matrix (Fig. 1). One can see from Algorithm 1 that the multigrid hierarchy (M) from the previous matrix can be reused. In this scenario, PCG will use the updated matrix A and so will the fine-grid of the AMG hierarchy. But the coarse grid operators along with the restriction and prolongation operators will not be recomputed. Effectively we will not incur an additional setup cost. We only create the updated matrix. The number of iterations taken by PCG will be higher, especially as the diffusion coefficients start to vary significantly from the original operator. However marginal increase in the number of iterations is still cheaper than the cost associated with the AMG setup, so this is likely to be faster. For long-running simulations, our heuristic is to update the AMG hierarchy by a fresh setup when the total time of updating the hierarchy and solving the linear system gets more expensive than the total time of setup and solve time for the initial matrix. Then, we consider the matrix in that step as the initial matrix, i.e. we repeat the setup phase completely and create a new multigrid hierarchy and repeat the previous process. In practice this approach works reasonably well, and the number of solves per setup is increased sufficiently to keep the overall runtime low.

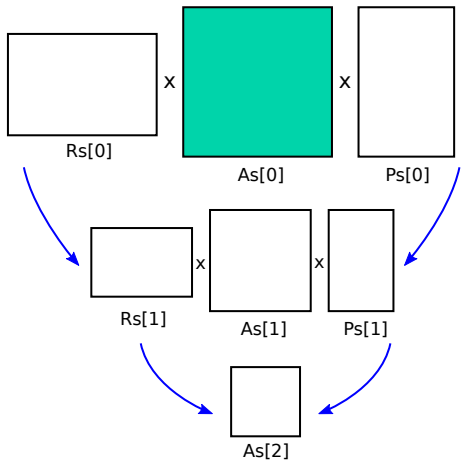

Fig. 1. This figure shows the multigrid hierarchy with 3 levels. The parts that are being updated are green. For Strategy 1 we only update As[0].

2) Strategy 2: Keep the same structure, only update As: While the previous case works well for many problems, it does not perform very well when there is large variation in the diffusion coefficients. We observe that variations in the diffusion coefficients do not affect the overall structure of the matrix. Therefore, we can use the aggregation from the previous AMG setup and simply update the coarse grid operators. In other words, we will keep the same aggregations, and therefore the restriction and prolongation operators, but update the coarse grid operators (Fig. 2). Unlike the previous case, we do have to pay a cost for re-computing the coarse-grid operators but this is not as expensive as a full AMG setup. At the same time, the convergence for this approach is better than Strategy 1, especially for problems with large variations.

3) Strategy 3: Only update local: No Communication: While Strategy 2 works well in the sequential case, it may not be the most efficient while computing in parallel. This is

978-1-7281-5020-8/19/$31.00 ©2019 IEEE

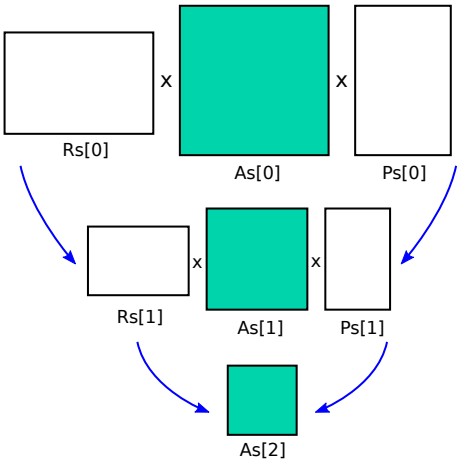

Fig. 2. Strategy2: update all coarse matrices (As). The parts that are being updated are green.

mainly due to the need to perform matrix multiplications in parallel to compute the coarse grid operator $A_c = RAP$. The communication required can become a bottleneck, especially as the coarser operators start to become dense. Therefore, as a final approximation, in the parallel case, we only update the local block of the coarse grid operators and do not incur any communication costs (Fig. 3). The finest level $As[0]$ is updated completely because it can simply be replaced by the updated matrix and does not require any matrix-matrix product. While this can affect convergence slightly, for the most part this simply behaves like Strategy 2, but is more efficient for large-scale parallel cases.

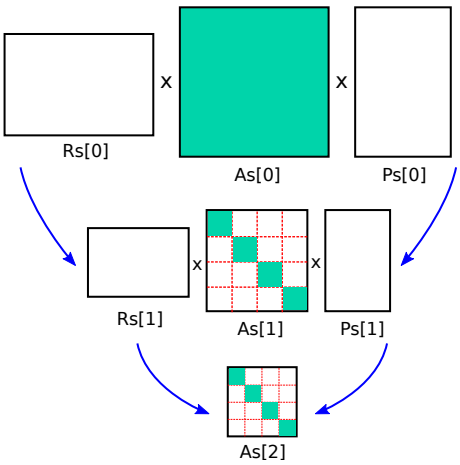

Fig. 3. Strategy3: Update the diagonal blocks of coarse matrices (As) to avoid the communication required for matrix-matrix product while performing the coarsening operation ($A_c = RAP$). The parts that are being updated are green. $As[0]$ is updated completely because it can just be replaced by the update matrix.

IV. Numerical Results

A. Experimental Setup

All experiments were conducted on a 12-node cluster at the Center for High Performance Computing (CHPC) at the University of Utah. Each node consists of a dual-socket Intel Xeon Haswell processors with 14 cores each for a total of 28 cores and 128GB per node.

To simulate the target problem discussed in §II, we use *HoMG* [17], which is a geometric multigrid library in MATLAB, to generate a set of matrices that have the same nonzero structure but with slightly different values. We start with a simple $2D$-mesh, and transform it to generate a set of matrices A_i's. Then we solve the linear systems $A_i x = b$, for all A_i's, using our lazy-update strategies.

First, we create the AMG hierarchy based on an initial matrix A_1. Then we use *HoMG* to generate A_2 which is slightly different from A_1. Similarly, matrix A_3 which is again slightly different from A_2 gets generated and so on.

After generating the matrices, we solve the linear system $A_1 x = b$. The right-hand side b is randomly generated. We use one iteration of *Chebyshev* for both pre-smoothing and post-smoothing for our AMG preconditioner.

Then, we update the AMG hierarchy for A_2 using one of our three strategies and then again solve the linear system $A_2 x = b$. We repeat this process to solve the linear system for all A_i's.

To study our update methods, we record three numbers:

- AMG Setup Time
- AMG Solve Time
- Number of Vcycle Iterations

and based on them show the effectiveness of our update methods.

B. Results

For our first experiment, we assemble a diffusion operator (A_1) using 4^{th}-order finite elements on a 64×64 $2D$ quad-mesh. A_1 is of size 66049×66049 with $2.3M$ nonzeros. Then, we create 9 other matrices by stretching the mesh that was used for creating A_1, in the x direction. The stretching of the mesh is equivalent to an anisotropic diffusion and allows us to consider various scenarios in a convenient manner. Fig. 4 shows a smaller version of the starting mesh (left) and how it gets transformed to the final mesh (right).

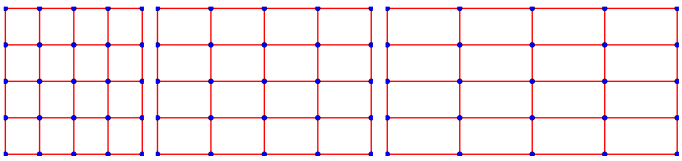

Fig. 4. This figure shows how we change the initial mesh to generate the new matrices A_i's. The left mesh shows (a smaller version of) the starting mesh that was used to generate A_1. The middle one is the transformed mesh for A_5. And, the last one is A_{10}'s mesh.

Table I shows the number of vcycle iterations that it takes to solve the linear systems $A_i x = b$. Strategy 2 takes less number of iterations, and the trade-off is a higher setup time. The result for Strategy 3 is almost identical to Strategy 1. The experiments of this section are done on 56 MPI tasks.

978-1-7281-5020-8/19/$31.00 ©2019 IEEE

TABLE I
NUMBER OF VCYCLE ITERATIONS FOR SOLVING THE LINEAR SYSTEMS. A_1 IS THE ORIGINAL MATRIX.

A_1	41		
	Strategy1	***Strategy2***	***Strategy3***
A_2	44	45	44
A_3	50	50	50
A_4	59	57	59
A_5	69	63	69
A_6	80	70	80
A_7	92	76	92
A_8	106	84	106
A_9	118	91	118
A_{10}	132	98	133

Figures 5-7 show the setup, solve and total time for the update strategies 1-3 respectively. The first point in each figure is the initial matrix A_1, so we choose its total time as the threshold. After reaching that threshold, we repeat the AMG setup phase and create a new hierarchy and repeat this process.

Fig. 5 shows the improvement that we gain by using Strategy 1 up to update-step 9. For matrix A_{10} we need to recreate the AMG hierarchy and use that instead. Fig. 6 shows that the setup time for Strategy 2 is less than the complete AMG setup time (the first point on the same figure) but it is significantly higher than the other two update methods. What we gain from that is a better solve time than Strategy 1 and 3. Up to matrix A_6 we gain speed-up using Strategy 2. Strategy 3 (Fig. 7) has a higher setup time than Strategy 1 but the same solve time, so Strategy 1 is preferred. For Strategy 3, up to matrix A_8 we have the time improvement using the updated hierarchy.

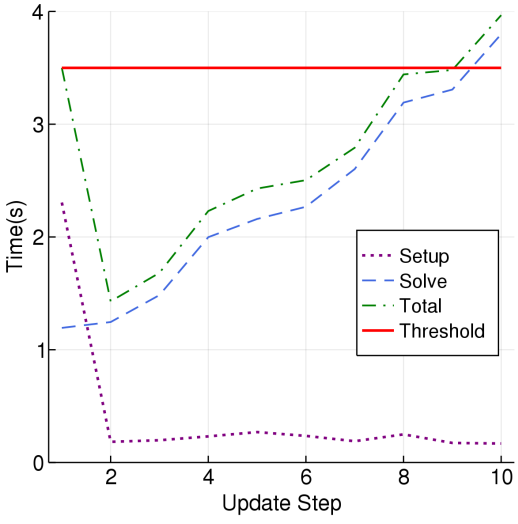

Fig. 5. Strategy 1: The lowest setup time but the highest slope for the solve time. For up to A_9 we have a lower total time if we use Strategy 1 to update the AMG hierarchy instead of creating the whole hierarchy for each matrix.

For the next experiment, we try a more complicated transformation to see if our update methods are still valid in a more complex situation. This time the operator is assembled using

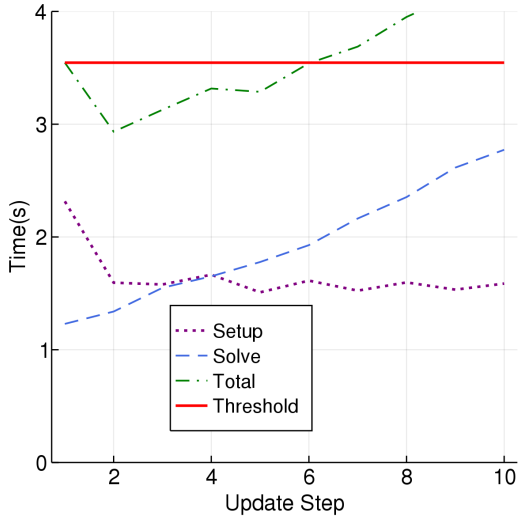

Fig. 6. Strategy 2: The highest setup times but the lowest solve times for the updated systems.

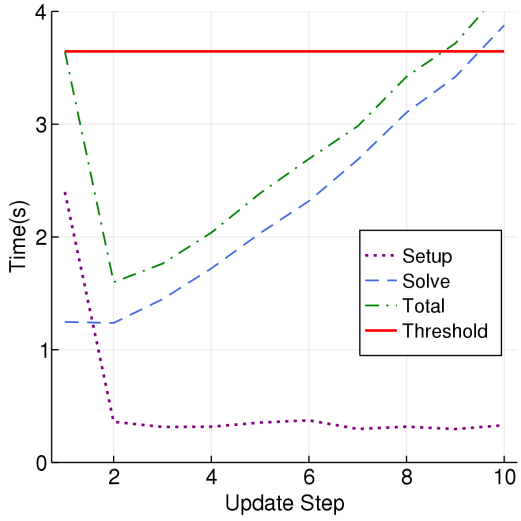

Fig. 7. Strategy 3: The setup time is in the middle but the solve time is almost identical to Strategy 1.

4^{th}-order finite elements on a 128×128 quad-mesh. A_1 is of size 263169×263169 with $9.3M$ nonzeros. We create 19 other matrices by transforming the mesh that was used for creating A_1 (Fig. 8).

Table II shows the number of vcycle iterations that it takes to solve the linear systems $A_i x = b$ and the results of the first experiment is again seen for this experiment.

Figures 9-11 confirm the same result about using our update methods that we observed in the first experiment. Fig. 9 shows the improvement that we gain by using Strategy 1 up to update-step 18. We see the speed-up using Strategy 2 and also For Strategy 3 up to matrix A_{16}.

978-1-7281-5020-8/19/$31.00 ©2019 IEEE

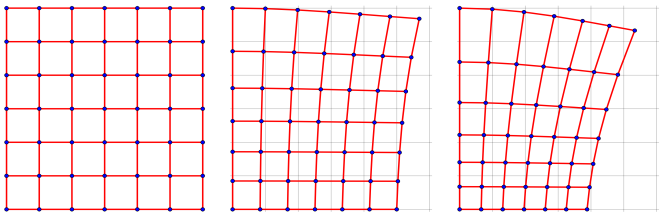

Fig. 8. This figure shows how we change the initial mesh to generate the new matrices A_i's for the second experiment. The left mesh show (a smaller version of) the starting mesh that was used to generate A_1. The middle one is the transformed mesh for A_{10}. And, the last one is A_{20}'s mesh.

TABLE II
NUMBER OF VCYCLE ITERATIONS FOR SOLVING THE LINEAR SYSTEMS. A_1 IS THE ORIGINAL MATRIX.

A_1	79		
	Strategy1	***Strategy2***	***Strategy3***
A_2	92	81	92
A_3	95	82	96
A_4	98	82	98
A_5	100	82	100
A_6	103	84	103
A_7	106	85	106
A_8	110	86	109
A_9	113	87	113
A_{10}	116	89	115
A_{11}	119	91	119
A_{12}	124	92	124
A_{13}	127	94	127
A_{14}	132	95	132
A_{15}	133	97	133
A_{16}	137	97	137
A_{17}	139	100	140
A_{18}	145	101	145
A_{19}	150	103	150
A_{20}	155	105	155

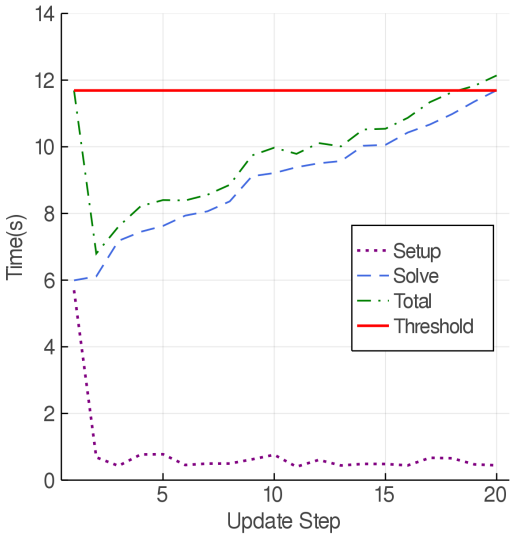

Fig. 9. Strategy 1: Similar to the first experiment, we see the lowest setup time but the highest slope for the solve time. For up to A_{18} we have a lower total time if we use Strategy 1 to update the AMG hierarchy instead of creating the whole hierarchy for each matrix.

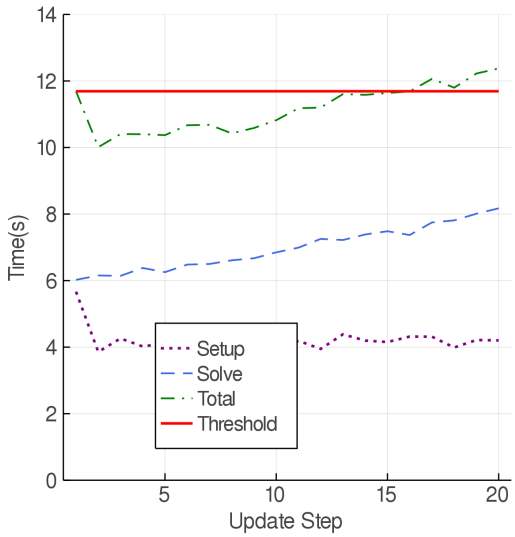

Fig. 10. Strategy 2: The highest setup times but the lowest solve times for the updated systems.

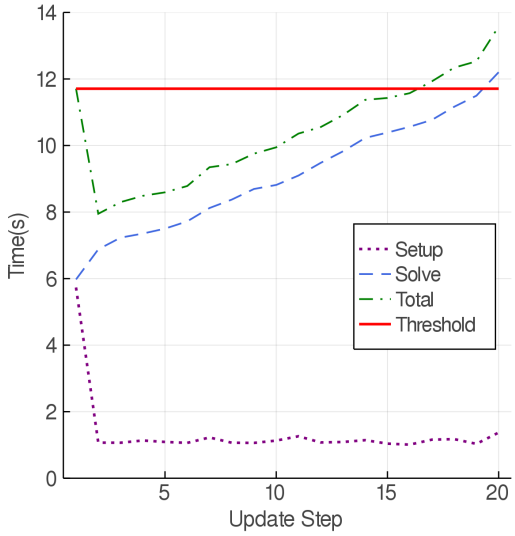

Fig. 11. Strategy 3: The setup time is in the middle but the solve time is almost identical to Strategy 1.

V. CONCLUSION

To avoid repeating the whole setup time of AMG for linear systems that have a similar left-hand side matrix, we studied three strategies to create the multigrid hierarchy once and only update that for solving linear systems of the similar matrices. By performing two different transformations, one simple and one more complicated, we generated similar matrices and showed how much time we can save using the mentioned update strategies. For future research, we want to try bigger experiments on higher number of MPI tasks to see if we gain any advantage of Strategy 3, which is the local update approach, over the complete update method (Strategy 2).

978-1-7281-5020-8/19/$31.00 ©2019 IEEE

VI. Acknowledgment

This work was supported in part by the National Science Foundation grant CCF-1704715 and Army Research Office W911NF1510222 (Program Manager Dr. Mike Coyle). This research used resources of the Extreme Science and Engineering Discovery Environment (XSEDE) allocation TG-PHY180002 and also the Center for High Performance Computing (CHPC) at the University of Utah.

References

[1] Y. Maday and R. Muñoz, "Spectral element multigrid. II. Theoretical justification," *Journal of scientific computing*, vol. 3, no. 4, pp. 323–353, 1988.

[2] J. H. Bramble and X. Zhang, "The analysis of multigrid methods," in *Handbook of numerical analysis, Vol. VII*, ser. Handb. Numer. Anal., VII. Amsterdam: North-Holland, 2000, pp. 173–415.

[3] S. C. Brenner, "Smoothers, mesh dependent norms, interpolation and multigrid," *Applied Numerical Mathematics*, vol. 43, no. 1-2, pp. 45–56, 2002, 19th Dundee Biennial Conference on Numerical Analysis (2001).

[4] D. Braess, "On the combination of the multigrid method and conjugate gradients," in *Multigrid Methods II*, W. Hackbusch and U. Trottenberg, Eds. Berlin: Springer–Verlag, 1986, pp. 52–64.

[5] O. Tatebe and Y. Oyanagi, "Efficient implementation of the multigrid preconditioned conjugate gradient method on distributed memory machines," in *Supercomputing'94. Proceedings*. IEEE, 1994, pp. 194–203.

[6] J. E. Dendy, Jr., "Black box multigrid," *Journal of Computational Physics*, vol. 48, no. 3, pp. 366–386, 1982.

[7] P. Vanek, J. Mandel, and M. Brezina, "Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems," Denver, CO, USA, Tech. Rep., 1995.

[8] P. Vaněk, M. Brezina, J. Mandel *et al.*, "Convergence of algebraic multigrid based on smoothed aggregation," *Numerische Mathematik*, vol. 88, no. 3, pp. 559–579, 2001.

[9] H. Sundar, G. Biros, C. Burstedde, J. Rudi, O. Ghattas, and G. Stadler, "Parallel geometric-algebraic multigrid on unstructured forests of octrees," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 43:1–43:11. [Online]. Available: http://dl.acm.org/citation.cfm?id=2388996.2389055

[10] H. Carslaw and J. Jaeger, "Heat conduction in solids," *Oxford University Press, Oxford*, p. 75, 1959.

[11] M. T. Van Genuchten, W. Alves *et al.*, "Analytical solutions of the one-dimensional convective-dispersive solute transport equation," United States Department of Agriculture, Economic Research Service, Tech. Rep., 1982.

[12] K. Leiderman and A. L. Fogelson, "Grow with the flow: a spatial–temporal model of platelet deposition and blood coagulation under flow," *Mathematical medicine and biology: a journal of the IMA*, vol. 28, no. 1, pp. 47–84, 2011.

[13] ——, "The influence of hindered transport on the development of platelet thrombi under flow," *Bulletin of mathematical biology*, vol. 75, no. 8, pp. 1255–1283, 2013.

[14] C. D. Cantwell, D. Moxey, A. Comerford, A. Bolis, G. Rocco, G. Mengaldo, D. De Grazia, S. Yakovlev, J.-E. Lombard, D. Ekelschot *et al.*, "Nektar++: An open-source spectral/hp element framework," *Computer Physics Communications*, vol. 192, pp. 205–219, 2015.

[15] N. Bell, S. Dalton, and L. N. Olson, "Exposing fine-grained parallelism in algebraic multigrid methods," *SIAM Journal on Scientific Computing*, vol. 34, no. 4, pp. C123–C152, 2012.

[16] E. Treister and I. Yavneh, "Non-galerkin multigrid based on sparsified smoothed aggregation," *SIAM Journal on Scientific Computing*, vol. 37, no. 1, pp. A30–A54, 2015.

[17] H. Sundar, G. Stadler, and G. Biros, "Comparison of multigrid algorithms for high-order continuous finite element discretizations," *Numerical Linear Algebra with Applications*, vol. 22, no. 4, pp. 664–680. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/nla.1979

978-1-7281-5020-8/19/$31.00 ©2019 IEEE