Lazy-update Multigrid Preconditioners

Majid Rasouli[1,2], Vidhi Zala[1,3], Robert M. Kirby[1,4], and Hari Sundar[1,5]

[1] School of Computing, University of Utah, SLC, UT, USA
[2] rasouli@cs.utah.edu
[3] vidhi@sci.utah.edu
[4] kirby@sci.utah.edu
[5] 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.

Keywords: Algebraic Multigrid · Iterative Solver · Sparse · Preconditioned Conjugate Gradient · Preconditioner

1 Introduction

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 shown in (1) is often used to obtain a numerical approximation of the exact location and population of the chemical species. Here, D referes 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 way 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 [2, 6]. 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.

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

1.1 Variable Diffusion case in Elliptic PDE

We now build the model problem of variable diffusion which is an example of elliptic PDE, and motivate the application of multigrid preconditioning methods to numerically solve the resulting equation. Consider the set of 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 [3, 4].

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 ($P^{se,a}$)

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

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

$k_{adh}(\boldsymbol{x})$ is assumed to be a positive constant for points $\boldsymbol{x}$ within one platelet diameter of subendothelium and zero elsewhere.

$$W(\phi^T) = tanh(\pi(1 - \phi^T)), \tag{4}$$

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 [3].

In order to numerically solve the model equations of the kind (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. We use the Finite Element software package Nektar++ [1] 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 discribe our methods in the next section §2 followed by experiments demonstrating the effectiveness of our approach in §3. Finally, we conclude with directions for future research in §**??**.

2 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. The linear system can be written as

$$Ax = b \tag{5}$$

in which, $A \in R^{n\times n}$, 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's 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 *maximal independent set* 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 has two applications. It can interpolate a vector $v \in R^m$ to $v' \in R^n$, such that $m < n$. The restriction operator does the reverse task; 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, which is called *coarsening*:

$$A_c = R * A * P \tag{6}$$

such that $A_c \in R^{m \times m}$.

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)
3. Restriction Matrices (Rs)

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

$$As[l+1] = Rs[l] * As[l] * Ps[l], \; l = 0, 1, 2, ..., L-1 \tag{7}$$

For this paper, *smoothed aggregation AMG (SA-AMG)* is used from [5], for which the prolongation (P_t) and restriction operators (R_t) are smoothed by

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

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.

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 reduce the error:

1. Relaxation
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 (and in some cases no) increase in the solve time. This will be our goal in the three strategies that are explained in the next section.

2.1 AMG as a preconditioner

While AMG can be used directly as a solver for our target problem, using it as a preconditioner for Conjugate Gradients makes it far more robust, especially given the complexity of our target meshes (see Figure3). The pseudocode for using AMG as a preconditioner with CG is given in Algorithm 1.

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

Algorithm 1 Multigrid-preconditioned CG

```
Require: rhs and guess
Ensure: solution
 1: while not converged do
 2:     $\boldsymbol{h} = A\boldsymbol{p}$
 3:     $\rho_r = (\rho, \boldsymbol{r})$
 4:     $\alpha = \rho_r / (\boldsymbol{p}, \boldsymbol{h})$
 5:     $\boldsymbol{u} = \boldsymbol{u} + \alpha\boldsymbol{p}$
 6:     $\boldsymbol{r} = \boldsymbol{r} - \alpha\boldsymbol{h}$
 7:     Convergence Test
 8:     $\rho = M\boldsymbol{r}$                                    ▷ AMG V-cycle
 9:     $\beta = (\rho, \boldsymbol{r}) / \rho_r$
10:     $\boldsymbol{p} = \rho + \beta\boldsymbol{p}$
11: end while
```

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.

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 (Figure 1, left). One can see from Algorithm 1 that the multigrid hierararchy (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 number of iterations becomes `2x` the number of iterations taken by the first matrix. In practice this approach works reasonably well, and the number of solves per setup is increased sufficiently to keep the overall runtime low.

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

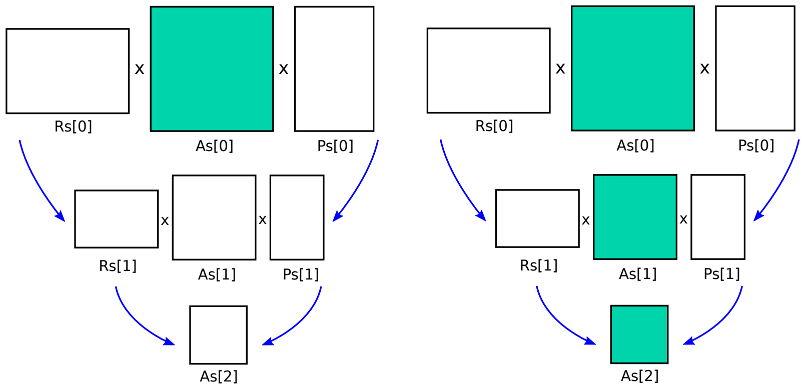

Fig. 1. This figure shows the multigrid hierarchy with 3 levels. Green boxes show which parts should be updated. (left) Strategy1: only update As[0]. (right) Strategy2: update all coarse matrices (As).

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 (Figure 1, right). 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 should be better than Strategy 1, especially for problems with large variations.

Strategy 3: Only update local: No Communication While Strategy 2 works well in the sequential case, it is not the most efficient while computing in parallel. This is 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 get 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 (Figure 2). The finest level As is updated completely because it is simply 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.

3 Numerical Results

3.1 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.

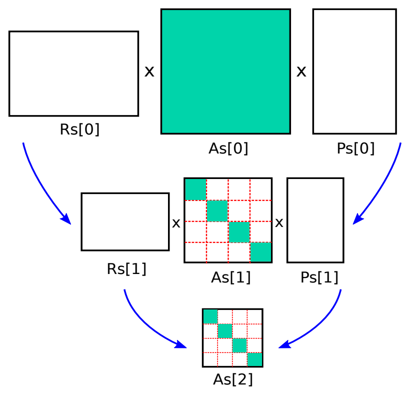

Fig. 2. Strategy3: Update the diagonal blocks of coarse matrices (As) to avoid the communication required for matrix-matrix product while performing the coarsening operation ($A_c = RAP$)

Two sets of experiments have been done for this section. The first experiment shows how lazy-update for AMG behaves on a matrix generated from a 2D mesh in *Nektar++* which is shown in Figure 3. The mesh has order 7 and has triangles and quads and is a challenging mesh for AMG. The generated matrix is of size 870 by 870 and has 33761 nonzeros.

The AMG hierarchy is created on an initial matrix A. Then the linear system $Ax = b$ is solved and number of vcycle iterations that it takes for the solve phase is written on the plot. Then, the difusion coefficients change based on a sine function to generate another matrix (B_1) which is slightly different from the initial one. The number of iterations to solve the system $B_1x = b$ is shown on the plot. The same process is repeated for multiple matrices.

The second set of experiments shows weak scaling and strong scaling of AMG as a preconditioner on 2D Poisson problem. One MPI task is assigned to each socket, so each node has two MPI tasks, corresponding to our hardware confguration. The number of OpenMP threads on each socket is 14. For the weak scaling each processor has almost 262k degree of freedom. In the strong scaling the total degree of freedom for all the experiments is 1048000.

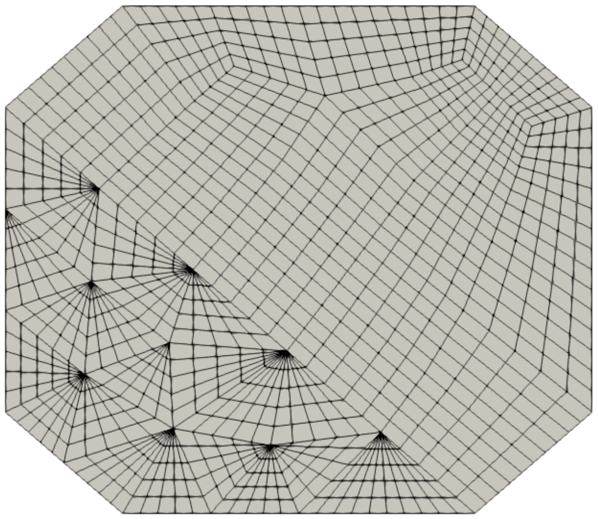

Fig. 3. The mesh that is used for the sinusoidal diffusion coefficient change experiment.

3.2 Results

Figure 4 shows the lazy-update AMG using the first strategy. The blue and red lines show how the coefficients are generated based on a sine function to create new matrices B_i. By using strategy1, only $As[1]$ of the whole AMG hierarchy will be updated to solve each new linear system $B_i x = b$. The plot shows that by saving the time using the same AMG hierarchy from the initial matrix, only a couple more (at most 6) iterations is required to solve a linear system similar to the initial one. So, even though we need to spend a little more time in the solve phase, almost the entire setup phase can be avoided.

Figure 5 shows a similar result for the second strategy. Here, all As are updated in the setup phase for the new matrices, so more setup time comparing to the first strategy is required, but at most 4 additional iterations are required during the solve phase.

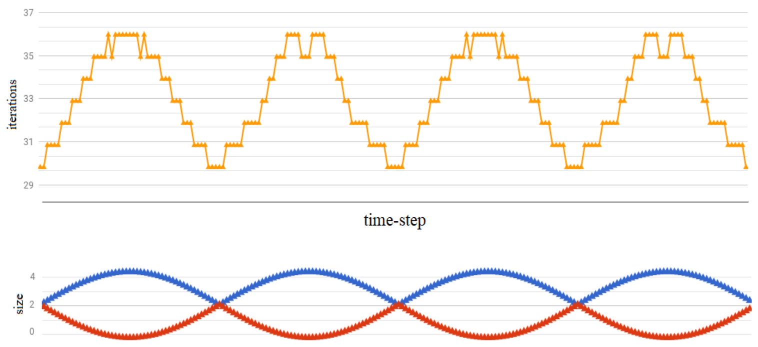

Fig. 4. Strategy1: The entire AMG hierarchy is re-used without incurring the cost of additional setup, except for the fine matrix. $As[1]$ is the only part that gets updated. The orange marks show how many iterations is required to achieve 10^{-12} relative tolerance. Each orange triangle corresponds to the number of iterations required to solve the system for a new updated matrix. The blue (red) triangles are the maximum (minimum) diffusion coefficients used to generate the new matrices.

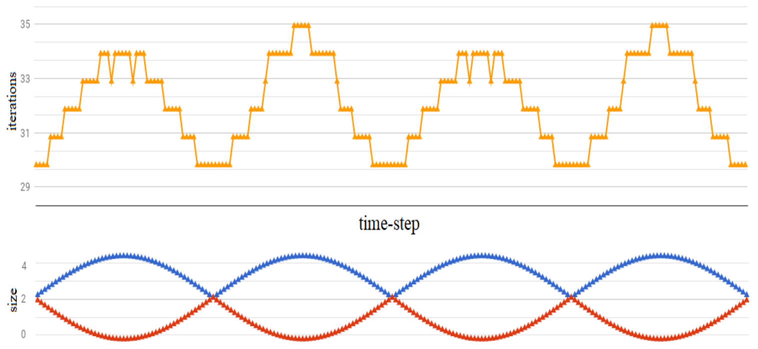

Fig. 5. Strategy2: The aggregations is not updated, but the coarse grid matrices are recomputed. All As matrices get updated, but Ps and Rs are used from the initial AMG hierarchy. The orange marks show how many iterations is required to achieve 10^{-12} relative tolerance. Each orange triangle corresponds to the number of iterations required to solve the system for a new updated matrix. The blue (red) triangles are the maximum (minimum) diffusion coefficients used to generate the new matrices.

Figure 6 shows the weak and strong scalability of the solve and setup phases, for the 2D Poisson problem. One can easily notice that the setup time takes more than 17 times more time than the setup phase for this experiment. From that, it is obvious why we accept a small increase in the solve time, to avoid doing some steps of the setup phase in the strategies we discussed.

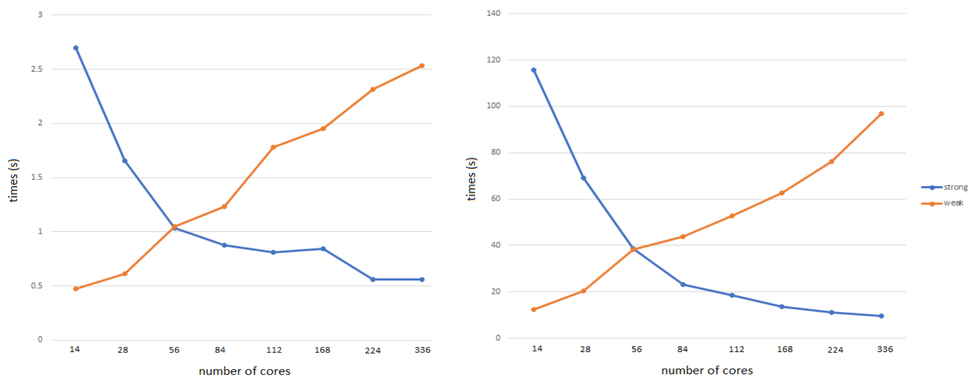

Fig. 6. strong (blue) and weak scaling (orange) of the AMG solve (left) and setup (right) phases. The setup phase takes significantly more time than the solve step.

References

1. Cantwell, C.D., Moxey, D., Comerford, A., Bolis, A., Rocco, G., Mengaldo, G., De Grazia, D., Yakovlev, S., Lombard, J.E., Ekelschot, D., et al.: Nektar++: An open-source spectral/hp element framework. Computer Physics Communications **192**, 205–219 (2015)
2. Carslaw, H., Jaeger, J.: Heat conduction in solids. Oxford University Press, Oxford p. 75 (1959)
3. Leiderman, K., Fogelson, A.L.: 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 **28**(1), 47–84 (2011)
4. Leiderman, K., Fogelson, A.L.: The influence of hindered transport on the development of platelet thrombi under flow. Bulletin of mathematical biology **75**(8), 1255–1283 (2013)
5. Treister, E., Yavneh, I.: Non-galerkin multigrid based on sparsified smoothed aggregation. SIAM Journal on Scientific Computing **37**(1), A30–A54 (2015)
6. Van Genuchten, M.T., Alves, W., et al.: Analytical solutions of the one-dimensional convective-dispersive solute transport equation. Tech. rep., United States Department of Agriculture, Economic Research Service (1982)