Lawrence Berkeley National Laboratory
LBL Publications

Title
Tuning the Coarse Space Construction in a Spectral AMG Solver

Permalink
https://escholarship.org/uc/item/3hb1k6wt

Authors
Marques, Osni
Druinsky, Alex
Li, Xiaoye S
et al.

Publication Date
2016

DOI
10.1016/j.procs.2016.05.311

Peer reviewed

eScholarship.org Powered by the California Digital Library
University of California

Procedia Computer Science

Volume 80, 2016, Pages 212–221

ICCS 2016. The International Conference on Computational Science

Tuning the Coarse Space Construction in a Spectral AMG Solver*

Osni Marques[1], Alex Druinsky[1], Xiaoye S. Li[1], Andrew T. Barker[2], Panayot Vassilevski[2], and Delyan Kalchev[3]

[1] Lawrence Berkeley National Laboratory, [oamarques,adruinsky,xsli]@lbl.gov
[2] Lawrence Livermore National Laboratory, [barker29,vassilevski1]@llnl.gov
[3] University of Colorado Boulder, delyank@gmail.com

Abstract

In this paper, we discuss strategies for computing subsets of eigenvectors of matrices corresponding to subdomains of finite element meshes achieving compromise between two contradicting goals. The subset of eigenvectors is required in the construction of coarse spaces used in algebraic multigrid methods (AMG) as well as in certain domain decomposition (DD) methods. The quality of the coarse spaces depends on the number of eigenvectors, which improves the approximation properties of the coarse space and impacts the overall performance and convergence of the associated AMG or DD algorithms. However, a large number of eigenvectors affects negatively the sparsity of the corresponding coarse matrices, which can become fairly dense. The sparsity of the coarse matrices can be controlled to a certain extent by the size of the subdomains (union of finite elements) referred to as agglomerates. If the size of the agglomerates is too large, then the cost of the eigensolvers increases and eventually can become unacceptable for the purpose of constructing the AMG or DD solvers. This paper investigates strategies to optimize the solution of the partial eigenproblems of interest. In particular, we examine direct and iterative eigensolvers for computing those subsets. Our experiments with a well-known model of an oil-reservoir simulation benchmark indicate that iterative eigensolvers can lead to significant improvements in the overall performance of an AMG solver that exploits such spectral construction of coarse spaces.

Keywords: spectral algebraic multigrid, interpolator, eigenvectors

*This material is based upon work supported by the US Department of Energy (DOE), Office of Science, Office of Advanced Scientific Computing Research (ASCR), Applied Mathematics program under contract number DE-AC02-05CH11231. This work was performed under the auspices of the DOE under Contract DE-AC52-07NA27344, and used resources of the National Energy Research Scientific Computing Center, which is supported by ASCR under contract DE-AC02-05CH11231.

Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2016
© The Authors. Published by Elsevier B.V.

doi:10.1016/j.procs.2016.05.311

1 Introduction

Multigrid is probably the only class of algorithms suitable for solving extreme-scale problems thanks to its potential for optimal computational complexity. This is certainly the case when we consider problems arising from the discretization of elliptic partial differential equations (PDEs) with some extensions to Darcy and Maxwell equations. When targeting unstructured meshes, i.e., problems for which we do not have access to a hierarchy of discretizations, the choice is to consider algebraic versions of multigrid, such as classical AMG and aggregation-based AMG, which rely only on the matrix problem to generate the hierarchy needed in the multilevel algorithm.

To make the methods more robust and handle problems with high variation and anisotropy of the PDE coefficients, one may use AMG methods that utilize additional (fine-grid only) problem information, such as the topology of the mesh; for finite element problems one may utilize the local element matrices. These AMG methods are referred to as element-based AMG, or AMGe for short. In this paper, we are concerned with the *Smoothed Aggregation Spectral Element Agglomeration Algebraic Multigrid* algorithm, in the form introduced in Brezina and Vassilevski [5], for solving difficult elliptic PDEs with variable coefficients that can be resolved only using a fine-grained discretization. We focus on the two-level variant of the method, implemented in the serial C++ code `SAAMGe` [11, 12]. The method requires the computation of subsets of eigenvectors on subdomains (union of fine-grid elements) of the fine discretization that are then used for constructing the coarse-grid. More specifically, we are interested in a portion of the lower part of the spectrum of these local subproblems. The computed eigenvectors are used to form the coarse-to-fine interpolation matrix (prolongator). In many cases, these preliminary computations are very time consuming, in part because the dimensions of the subsets are difficult to determine a priori. Therefore, speeding up these computations can have a significant impact in the overall performance of the multigrid solver. (The impact of the coarse-grid solver on the performance of the algorithm has been studied in [8].)

Our main contributions are the following. First, we examine different strategies for computing eigenvectors in the context of `SAAMGe`, using direct and iterative algorithms. Second, we examine alternatives to determine appropriate dimensions for the subsets of eigenvectors in iterative algorithms which is needed to provide guidance for optimizing the setup while maintaining the quality of the coarse spaces. The rest of the paper is organized as follows. We start by explaining how the `SAAMGe` algorithm works, and the mechanism for building a prolongator operator from subsets of eigenvectors on subdomains of a fine mesh (Section 2). We discuss the characteristics of the eigenvalue problems to be solved, algorithms that can be used for that purpose, and observations that motivated our quest for a strategy to adaptively compute eigenvectors of those subdomains (Section 3). We then discuss the experiments that we have performed, and the results for the different eigensolution strategies (Section 4). Finally, we summarize our conclusions and potential future work (Section 5).

2 Smoothed Aggregation Spectral Element-Based Algebraic Multigrid

In this section, we outline briefly the AMG method that we consider. We are given a symmetric positive definite matrix A that is assembled from local element matrices $\{A_\tau\}$, where τ runs over the elements from a finite element mesh $\mathcal{T}_h$. The local matrices are symmetric but can be semi-definite. We have $\mathbf{v}^T A\mathbf{v} = \sum_{\tau\in\mathcal{T}_h} \mathbf{v}_\tau^T A_\tau \mathbf{v}_\tau$, for any vector $\mathbf{v}$, and $\mathbf{v}_\tau = \mathbf{v}|_\tau$ is the restriction of $\mathbf{v}$ to the element τ viewed as a set of degrees of freedom (DOF). For example, if τ is a triangle, then $\mathbf{v}_\tau$ can be the restriction of $\mathbf{v}$ to the set of vertices of τ. A main step in AMG is the construction of an interpolation matrix P. It is a rectangular matrix whose number of columns equals the dimension of the coarse vector space. The coarse matrix A_c is obtained via the triple-matrix product P^TAP. There are several requirements on P to have an efficient two-level (or two-grid, TG) algorithm. They relate the coarse space and the smoothing process, which is defined from a stationary iteration such as Jacobi or Gauss-Seidel. For a parallel multigrid, the smoother has to be well-parallelizable, which is ensured if the respective smoothing matrix M is diagonal, or the smoothing iteration can be represented as a sequence of sparse matrix vector products. This is the case of polynomial smoothers, i.e., when $M^{-1} = (I - p_\nu(D^{-1}A))A^{-1}$, where $p_\nu(t)$ is a polynomial ($p_\nu(0) = 1$) and D is diagonal. Once the matrices P and A_c are constructed, to solve $Ax = b$, the TG-solver iterates by alternating between the fine and coarse spaces. Specifically, cycle k transforms the current iterate x_k into x_{k+1} using the following steps:

1. Pre-smoothing: $y_k \leftarrow x_k + M^{-1}(b - Ax_k)$
2. Restriction: $r_c \leftarrow P^T(b - Ay_k)$
3. Coarse solution: $x_c \leftarrow A_c^{-1}r_c$
4. Interpolation: $z_k \leftarrow y_k + Px_c$
5. Post-smoothing: $x_{k+1} \leftarrow z_k + M^{-1}(b - Az_k)$

In a convergent TG method, the matrix D and the interpolation matrix are related via an important *weak approximation property*, $\|\mathbf{v} - P\mathbf{v}_c\|_D \leq \eta_w \|\mathbf{v}\|_A$, which says that every fine-grid vector $\mathbf{v}$ can be approximated well from the coarse space, i.e., there is a coarse vector $\mathbf{v}_c$ such that its interpolant $P\mathbf{v}_c$ is close to $\mathbf{v}$ in the norm defined from the smoother (D). The constant η_w is related to the convergence factor of the TG iteration.

In the element-based AMG, we build P by solving local eigenproblems using the locally assembled matrices A_T and the restriction of the smoothing matrix D, D_T, to the set of fine DOFs associated with T. T is a subdomain formed of fine-grid elements. The computational domain is covered by the sets $\{T\}$, which form a non-overlapping partition of the fine-grid element mesh. However, as sets of DOF of two neighboring elements have shared DOFs, the sets $\{T\}$ are overlapping in terms of DOFs. The shared DOFs are assigned uniquely to one set called aggregate $\mathcal{A}$, and an aggregate is contained in a unique agglomerate T. The set of aggregates $\{\mathcal{A}\}$ forms a non-overlapping partition of the fine DOFs.

To build P, we select a set of eigenvectors that we compute on each T, a rectangular matrix from each aggregate, and build first the so-called tentative prolongation operator $\bar{P}$. More specifically, $\bar{P}$ is obtained from the eigenvectors associated with the smallest eigenvalues of the local stiffness matrices of the agglomerates. They are chosen, so that a local version of the above weak approximation property holds. We set $\bar{P} = diag(\ \bar{P}_1 \ \ \bar{P}_2 \ \ldots \ \bar{P}_{na}\ )$ where the columns of $\bar{P}_i$ contain a subset of eigenvectors of agglomerate i, and na is the number of agglomerates. To obtain the ultimate prolongator P, in smoothed aggregation (SA) methods, we premultiply $\bar{P}$ by a matrix polynomial smoother S: $P = S\bar{P}$. The composite smoother S is of the form $S = s(D^{-1}A)$, where $s(\cdot)$ is a polynomial that is defined by its roots and D is a diagonal matrix that comes from A. To premultiply $\bar{P}$ by S, we explicitly form the sparse matrices that represent the polynomial factors of S and premultiply $\bar{P}$ with each of these matrices. The resulting P is a tall-and-skinny sparse matrix. Therefore, the column dimension of P is a function of the number of agglomerates and the number of eigenvectors that are selected from each agglomerate, both determined by parameters of the algorithm. Finally, the coarse-space matrix is computed as $A_c = P^T AP$, where A is the fine-grid sparse matrix. Details are given in [5, 12].

The performance of the algorithm is strongly influenced by how we choose its parameters. To solve the local eigenvalue problems, we use a tolerance θ. This spectral tolerance determines the number of eigenvectors we select from each agglomerate: we select the eigenvectors whose corresponding eigenvalues are smaller than θ. Larger θ implies more eigenvectors in each $\bar{P}_i$, a larger coarse problem and an improved convergence rate. The downside is greater operator complexity and lower computational performance. In turn, a small number of elements per agglomerate leads to a larger number of agglomerates of smaller size, a larger coarse problem, and a potentially higher operator complexity, all without guaranteeing improvements in the convergence rate. The other relevant parameters are the polynomial degrees of the smoother S and of the relaxation operator M. The former is related to the number of elements per agglomerate. It should increase with the number of elements per agglomerate to guarantee the necessary approximation properties of the coarse space. The latter should also increase with the number of elements per agglomerate, but large values imply more relaxation and therefore a higher computational cost for each AMG iteration. A small degree (2 or 3) for both polynomials is sufficient in most cases. In the present paper, we focus on optimizing the way we solve the local eigenvalue problems depending on their size and the imposed spectral tolerance.

3 The Eigenvalue Problems in `SAAMGe`

For each agglomerated element T we form a non-overlapping subset of its degrees of freedom, and call this non-overlapping subset $\mathcal{A}$. That is, we partition each agglomerate T into degrees of freedom that it owns and degrees of freedom that it shares with its neighbors, so we can write the stiffness matrix of the agglomerated element T as (in Matlab notation) $A_T = [\ A_\mathcal{A}\ A_{\mathcal{A}\phi};\ A_{\phi\mathcal{A}}\ A_\phi]$. The local eigenvalue problems we want to solve are of the form

$$S_\mathcal{A} q = \lambda D_\mathcal{A} q \tag{1}$$

where $D_\mathcal{A}$ is a diagonal matrix (a weighted smoother obtained from $A_\mathcal{A}$) and $S_\mathcal{A}$ is the Schur complement of the agglomerate stiffness matrix A_T with respect to its subset $\mathcal{A}$, i.e., $S_\mathcal{A} = A_\mathcal{A} - A_{\mathcal{A}\phi} A_\phi^{-1} A_{\phi\mathcal{A}}$.

3.1 Direct eigensolvers

To avoid explicitly forming $S_\mathcal{A}$ (which is dense), the original approach used in `SAAMGe` for solving (1) is to define

$$A_T q = \lambda D_{T,0} q \tag{2}$$

where $D_{T,0}$ is just $D_\mathcal{A}$ extended with zeros to be the size of A_T. This form has the advantage that all the matrices involved are sparse and easy to compute, but the potential disadvantage that the right-hand side matrix $D_{T,0}$ is singular. For simplicity, from now on we drop the subindexes of A and D. We then rewrite (2) as

$$Dq = \frac{1}{1+\lambda}(A + D)q, \tag{3}$$

such that the eigenvalues $\hat{\lambda} = \frac{1}{(1+\lambda)}$ lay in the interval $[0.5, 1]$ and small λ's correspond to large $\hat{\lambda}$'s. Problem (3) can be transformed from sparse to dense and solved with LAPACK's

DSYGVX, which computes selected eigenvalues and, optionally, eigenvectors of a real generalized symmetric-eigenvalue problem $\hat{A}x = \lambda \hat{B}x$, where $\hat{B}$ is assumed to be positive definite[1]. A direct eigensolver first transforms $\hat{A}x = \lambda \hat{B}x$ into the standard form $(L^{-1}\hat{A}L^{-T})y = \lambda y$, where L is obtained from the Cholesky factorization $\hat{B} = LL^T$ and $y = L^T x$. Then, $(L^{-1}\hat{A}L^{-T})y = \lambda y$ is mapped into a tridiagonal eigenvalue problem, where the tridiagonalization of $(L^{-1}\hat{A}L^{-T})$ is typically the most time consuming part of the eigensolution.

In DSYGVX, eigenvalues (and corresponding eigenvectors) to be computed can be specified through a range of values, i.e., an interval (VL,VU], or a range of indexes (i.e., IL-th through IU-th eigenvalues). We use the interval option, setting VL to $\frac{1}{(1+\theta)}$ and VU to a number slightly bigger than 1. With the transformation in (3), the eigenvalues become closer to each other, potentially in tight clusters. DSYGVX uses bisection to compute eigenvalues and inverse iteration to compute eigenvectors [3]; it may may require additional work to reorthogonalize the computed eigenvectors associated with tight clusters of eigenvalues.

For large fine grids, problem (2) needs to be solved thousands of times by calling DSYGVX to compute from a few to tens of eigenvectors depending on the spectral tolerance θ. To illustrate, Fig. 1 shows a histogram of the number of eigenvectors (x-axis) versus the number of agglomerates that require that number of eigenvectors (y-axis), for isotropic (I) and anisotropic (A) models of the SPE10 benchmark (see Section 4), using 400 finite elements per agglomerate, and θ equal to 0.001 and 0.005. A very large fraction of the agglomerates requires just a few eigenvectors; some agglomerates require several, and ten or more, depending on θ. Additional experiments (not shown) with larger agglomerates (e.g. 1000 finite elements per agglomerate) revealed that some agglomerates required more than 20 eigenvalues for the θ's used in the Fig. 1.

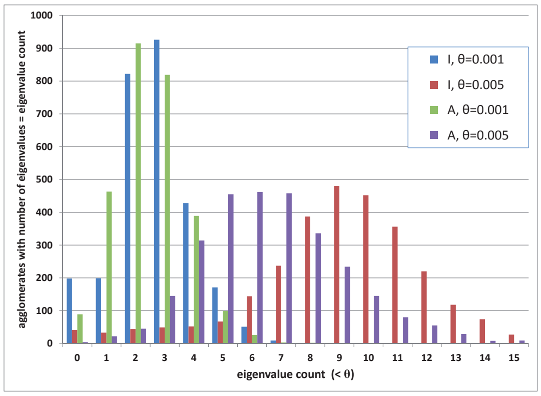


Figure 1: Number of eigenvectors per agglomerate for the isotropic (I) and anisotropic (A) models of the SPE10 benchmark, with 400 finite elements per agglomerate, and θ equal to 0.001 and 0.005. More than 90% of the agglomerates require less than 5 eigenvectors if $\theta = 0.001$.

3.2 Iterative eigensolvers

Alternatively to the mapping (3), one can solve (2) through a projection-based method, e.g. through Arnoldi, Lanczos or Jacobi-Davidson type algorithms [7]. In this case, a shift-and-invert is needed, i.e., (2) needs to be rewritten as

$$(A - \sigma D)^{-1} Dq = \mu q, \tag{4}$$

where $\mu = \frac{1}{\lambda - \sigma}$. If σ is near the origin, small λ's become large and better separated in μ's, which favors the convergence of projection methods. With Lanczos or Arnoldi, the effect of $(A - \sigma D)^{-1}$ can be realized through a LU or LDL^T factorization of $(A - \sigma D)$ and subsequent solutions of systems of equations. With Jacobi-Davidson, the effect of $(A - \sigma D)^{-1}$ can be realized with iterative solvers and preconditioners [15].

[1]For solving (3), we invoke DSYGVX with $\hat{A} = D$ and $\hat{B} = A + D$.

3.3 Direct versus iterative eigensolvers at a glance

Iterative eigensolvers can take advantage of the sparsity of the matrices in Equation (2). Given the size of the eigenproblems that need to be solved in `SAAMGe`, the question is whether an iterative solver could outperform a direct solver. To help finding the answer, we took two agglomerates of dimensions 612 and 1373 from the SPE10 benchmark, and computed an increasing number of eigenvalues and eigenvectors using Matlab's `eigs` and LAPACK's DSYGVX (through a Matlab's MEX file). We recall that `eigs` is an interface for `ARPACK`, with provides an implementation of the Arnoldi Algorithm with implicit restarts [13]. Although not shown here, the timings for these experiments revealed that for the small problem the break-even point is about 13, while for the large problem `ARPACK` is consistently faster. (The timings for `ARPACK` include the transformation discussed in the previous subsection.) This simple experiment encouraged us to investigate the potential impact of an iterative eigensolver in the overall performance of `SAAMGe`. Hence, we started by adding an interface for `JADAMILU` in `SAAMGe`. `JADAMILU` provides an implementation of the Jacobi-Davidson method that uses a general preconditioner or a multilevel incomplete LU preconditioner [4][2]. Given a preconditioner, `JADAMILU` requires only matrix-vector multiplications with A and D. We then added an interface for `ARPACK`, using `SuperLU` [14] to factor and solve systems of equations involving the shifted operator in (4).

3.4 Early termination for iterative eigensolvers

While DSYGVX allows us to use a cutoff (the spectral tolerance θ) for the number of eigenvectors to be taken into account in each agglomerate, a similar feature is not available in `JADAMILU` and `ARPACK`, which take as input a fixed number of eigenpairs, *neig*, to be computed. In fact, this is the case of most implementations of projection-based methods. However, in our interface for `ARPACK`, we added a mechanism that mimics a cutoff, by monitoring the convergence *externally* to `ARPACK`. We recall that for symmetric matrices Arnoldi behaves like Lanczos: it projects the original eigenvalue problem into a smaller one, involving a symmetric tridiagonal matrix T. The accuracy of approximate solutions for the original problem can be measured through the bottom entries of the normalized eigenvectors of T [17]. In other words, for an approximate solution $(\hat{\lambda}, \hat{q})$ of (2) by means of (4), at the j-the step of the Lanczos algorithm we have $T_j s = \mu s$, $\hat{q} = P_j s$, $P_j^T D_L P_j = I$, where $P_j = [p_1, p_2, \ldots p_j]$ contains the vectors generated by the Lanczos algorithm. It can be shown that $\|A_L \hat{q} - \hat{\lambda} D_L \hat{q}\| = \beta_{j+1} |s^{(j)}|$, where β_{j+1} is a normalization factor related to p_{j+1} (the vector that would expand P_j) and $s^{(j)}$ is the bottom entry of s. This means that at step j we can check the accuracy of j pairs $(\hat{\lambda}, \hat{q})$. It is also possible to use more sophisticated ways of monitoring convergence in the Lanczos algorithm [18] but in our implementation we have adopted a modified version of the implicit QL method for tridiagonals implemented in [9], as also discussed in [10], to compute only the bottom entries of the eigenvectors s of T_j. (The plane rotations used in the implicit QL method can be rearranged to compute only those entries.) We have incorporated this strategy externally to `ARPACK` to monitor T_j, check whether there are eigenvalues below the cutoff (θ), whether the corresponding eigenpairs have converged, and then trigger an *early termination* of `ARPACK`. This also allows us to use a tolerance for convergence different than the one set for `ARPACK` at the beginning.

The caveat of the early termination strategy is that *neig* (the fixed number of eigenpairs) may conflict with θ, i.e., we may set *neig* to be smaller than the number of eigenvalues below θ. Based on our experiments and the results shown in Fig. 1, we estimate that *neig* $= 10$ should suffice for most cases. A more robust strategy could consist of allowing for restarts of

[2]The version of `JADAMILU` prepared to solve generalized eigenvalue problems is available only in binary form.

eigensolver	problem and approach
DSYGVX	(3)): $(\lambda_i, q_i), \lambda_i < \theta$; bisection plus inverse iteration [3]
`JADAMILU (diag)`	(4): $(\lambda_i, q_i), i = 1, 2, \dots neig, \lambda_i \leq \lambda_{i+1}$; `JADAMILU` with diag. precond.
`JADAMILU (ILU)`	(4): $(\lambda_i, q_i), i = 1, 2, \dots neig, \lambda_i \leq \lambda_{i+1}$; `JADAMILU` with ILU precond.
`ARPACK (1)`	(4): $(\lambda_i, q_i), i = 1, 2, \dots neig, \lambda_i \leq \lambda_{i+1}$; `ARPACK+SuperLU`
`ARPACK (2)`	(4): $(\lambda_i, q_i), i = 1, 2, \dots neig, \lambda_i \leq \lambda_{i+1} < \theta$; `ARPACK+SuperLU`

Table 1: Summary of eigensolvers used in the tests; `ARPACK (1)-(2)` in shift-invert mode.

the eigensolver, to compute additional eigenvectors, if *neig* is determined not to be enough.

4 Numerical experiments

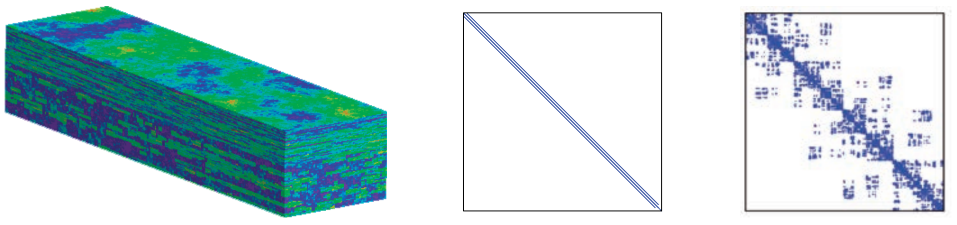
In this section, we show performance results using the eigensolvers summarized in Table 1: DSYGVX, `JADAMILU` with a diagonal preconditioner, `JADAMILU` with incomplete LU preconditioner, `ARPACK` in shift-invert mode, and `ARPACK` in shift-invert mode with the early termination strategy discussed in section 3.4 We show results related to a real problem[3], on a Cray XC system, a parallel system based on 2.3 GHz 16-core Haswell processors per node, and 128 GB of memory per node. On this system we use the Intel compiler suite, and -O2 as the level of optimization for the compiler. We use `LAPACK` 3.5.0, `SuperLU` 4.3, and the most recent version of `ARPACK`. We use $\sigma = -0.0001$ and `SuperLU` in symmetric mode to perform the factorization $(A - \sigma D) = LU$ in problem (4).

SPE10 benchmark

This is an oil reservoir simulation benchmark from the SPE Comparative Solution Project [2]. The benchmark is related to fluid flow in porous media described by the Darcy equation (in primal form), $-\nabla \cdot (\kappa(x)\nabla p(x)) = f(x)$, where $p(x)$ is the pressure field and $\kappa(x)$ is the permeability of the medium. The challenge arises when the coefficient $\kappa(x)$ admits a wide range of variation. The particular test case we use is the SPE10 (model 2) [6], which is a 3D waterflood of a 1.1-million-cell geostatistical model. At the fine geological model scale, the model is described on a regular Cartesian grid, with dimensions $1200 \times 2200 \times 170$ cubic feet. The top 70 feet (35 layers) represent the Tarbert formation, and the bottom 100 feet (50 layers) represent Upper Ness. The fine-scale cell size is $20 \times 10 \times 2$ cubic feet, with a total of $60 \times 220 \times 85$ cells. The porosity of the model is shown in Fig. 2a. The model admits jumps of several orders of magnitude between distinct horizontal layers of the medium. Using the finite-element discretization library MFEM [1], we obtained a fine-grid matrix with dimension 1.16M and 30.6M nonzero entries. The model has an isotropic variant, in which the permeability is a scalar, and an anisotropic one, in which it is a tensor. The matrix A has the same structure in both cases, with an average of 26.4 nonzero entries per row (see Fig. 2(b). The dimensions and the sparsity pattern of the coarse-space matrix depend on the `SAAMGe` parameters (see Section 2). An example of a coarse-grid matrix is shown in Fig. 2(c).

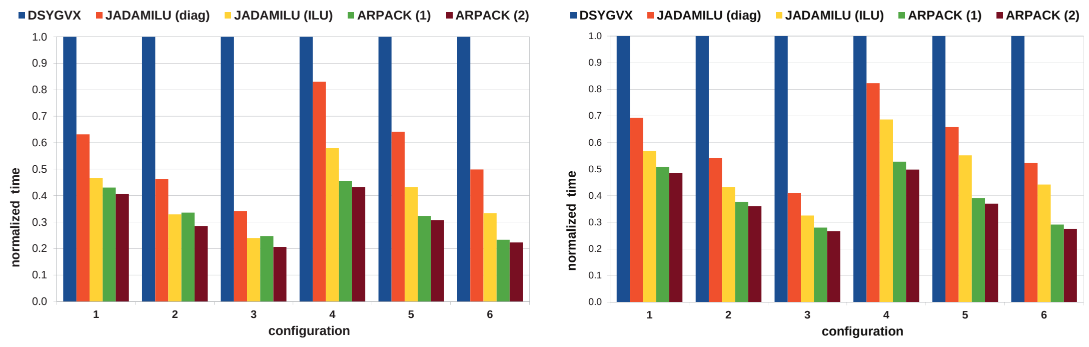
Figure 3 shows timings for the eigensolvers in Table 1 for 6 configurations of the algorithm `SAAMGe` for SPE10. For configurations 1-3 we used 300, 400, and 500 finite elements per agglomerate, respectively; similarly for configurations 4-6. These values lead to 3740 agglomerates of average size 483 for configuration 1, 2805 agglomerates of average size 619 for configuration

[3]The results are also representative for a variety of meshes that we have experimented with.



(a) porosity model (b) fine-grid matrix (c) coarse-grid matrix

Figure 2: (a) porosity of model 2 in SPE10 (source: http://www.spe.org); (b) sparsity pattern of the fine-grid matrix; (c) example of a coarse-grid matrix.



(a) isotropic model (b) anisotropic model

Figure 3: Timings for eigenvalue calculations for 6 configurations of SPE10: DSYGVX (blue), `JADAMILU (diag)` (red), `JADAMILU (ILU)` (yellow), `ARPACK (1)` (green), and `ARPACK (2)` (purple). The timings are normalized with respect to DSYGVX in each configuration.

2, and 2244 agglomerates of average size 752 for configuration 3. For configurations 1-3 we used $\theta = 0.001$ and for configurations 4-6 we used $\theta = 0.005$. For these θ's, DSYGVX computes between 2.3 to 3 eigenvectors (in average) per agglomerate for configurations 1-3, and 7.5 to 10.5 for configurations 4-6. Given these numbers, we set *neig* to 5 for the iterative solvers in configurations 1-3, and to 10 for configurations 4-6. The maximum number of matrix vector multiplications for `JADAMILU` was set to 10 times the number of DOFs of the agglomerate, with a tolerance for the eigenvector residual (tolerance for convergence) equal to 10^{-10}. The maximum basis size to be computed by `ARPACK` before an eventual restart was set to $(neig+10) \times 2$, and the tolerance for convergence to machine epsilon, $\varepsilon \approx 1.11 \times 10^{-16}$. As can be seen in the figure, the reduction in time ranges from 50% to almost 80% for the iterative solvers, depending on the size of the agglomerates.

Finally, Table 2 shows the total time to solution (setup phase, eigenvalue calculations plus AMG cycles) for the anisotropic model of SPE10 using the eigensolvers listed in Table 1. For some configurations, the use of DSYGVX leads to a smaller time to solution than `JADAMILU (diag,ILU)` and `ARPACK (1)`. This is because the coarse grids obtained with these iterative eigensolvers are bigger, since they compute a fixed number of eigenvectors per agglomerate, without necessarily reducing the number of AMG cycles. In contrast, `ARPACK (2)` consistently leads to the smaller time to solution, from 26% to 58% gains, with coarse grid sizes and number

config.	DSYGVX	`JADAMILU (diag)`	`JADAMILU (ILU)`	`ARPACK (1)`	`ARPACK (2)`
1	655	820	531	467	365
2	790	507	607	563	395
3	838	422	725	452	359
4	1219	1810	1135	780	512
5	998	952	1354	1297	739
6	1203	762	1769	883	623

Table 2: Total time to solution (seconds) for the six configurations of SPE10, anisotropic model. The largest times in each configuration is in red; the smallest is in blue, i.e., `ARPACK (2)`.

of AMG cycles (not shown) that are similar to the ones obtained with DSYGVX.

5 Conclusions

In this paper we showed that iterative eigensolvers can lead to significant time savings in the computation of subsets of eigenvectors that are needed for the computation of the prolongator in a TG algebraic multigrid solver. Given the size of the eigenvalue problems to be solved, a shift-and-invert strategy for a Krylov-based eigensolver outperformed a preconditioned eigensolver for all cases examined. We showed that additional savings can be obtained with a strategy that allows for an earlier stop of the Krylov-based eigensolver. Overall, the eigenvalue calculation phase has been sped up more than 4x in comparison with the original baseline implementation, depending on the problem and configuration. This speed up has been reflected in the total time to solution in the TG solver, i.e., setup phase, eigenvalue calculations plus AMG cycles: the total time to solution can be up to 2x faster.

Although not shown in this paper, preliminary experiments with a parallel multi-level AMG have also shown very promising savings when using a Krylov-based eigensolver. However, in this version of the solver the agglomerates become denser and potentially larger as we compute a sequence of denser and smaller coarse grids (i.e., as a consequence of $A_c = P^T AP$, as discussed in Section 2). Our experiments have revealed that some of the larger agglomerates may require a disproportionate large number of eigenvectors (a trend that can be observed in Fig. 1 if we increase θ), without significantly improving the convergence of the multi-level solver, but with a negative effect in terms of load balancing. As future work, we plan to investigate a strategy for obtaining agglomerates of similar sizes (in each level), which in turn will lead to a better distribution of the time required for the computation of eigenvectors and a better overall performance of the multi-level solver.

Also as future work, we plan to investigate how the tolerance for convergence in iterative eigensolvers might affect the number of AMG cycles. In [16, 19], the authors propose the use of alternative sets of vectors (Ritz or Lanczos), cheaper to compute than eigenvectors, for the dynamic analysis of structures. In these approaches, a "participation factor", based on the loading the structures are subject to, is used to determine how many vectors to be taken into consideration in the analyses. Although such a factor may not make sense in AMG, it may still be possible to use alternative sets of vectors, not so strongly dependent on a fixed spectral tolerance, in the construction of the prolongator.

References

[1] MFEM. http://mfem.org.

[2] SPE comparative solution project. http://www.spe.org/web/csp.

[3] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, Jack J. Dongarra, J. Du Croz, S. Hammarling, A. Greenbaum, A. McKenney, and D. Sorensen. *LAPACK Users' Guide (Third Ed.)*. SIAM, Philadelphia, PA, USA, 1999.

[4] Matthias Bollhöfer and Yvan Notay. JADAMILU: a software code for computing selected eigenvalues of large sparse symmetric matrices. *Computer Physics Communications*, 177:951–964, 2007.

[5] Marian Brezina and Panayot S. Vassilevski. Smoothed aggregation spectral element agglomeration AMG: SA-ρAMGe. In Ivan Lirkov, Svetozar Margenov, and Jerzy Waśniewski, editors, *Large-Scale Scientific Computing*, volume 7116 of *LNCS*, pages 3–15. Springer Berlin Heidelberg, 2012.

[6] M. A. Christie and M. J. Blunt. Tenth SPE comparative solution project: Comparison of upscaling techniques. *SPE Reserv. Eval. Eng.*, 4(4):308–317, 2001.

[7] James Demmel, Jack Dongarra, Axel Ruhe, and Henk van der Vorst. *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*. SIAM, Philadelphia, PA, USA, 2000.

[8] Alex Druinsky, Peter Ghyels, Xiaoye Li, Osni Marques, Samuel Williams, Andrrew Barker, Delyan Kalchev, and Panayot Vassilevski. Comparative performance analysis of coarse solvers for algebraic multigrid on leading multicore architectures. pages 1–2, 2015.

[9] Burton Garbow, James Boyle, Jack Dongarra, and Cleve Moler. *Matrix Eigensystem Routines - EISPACK Guide Extension*, volume 51 of *Lecture Notes in Computer Science, Volume 6*. Springer Verlag, 1977.

[10] R. G. Grimes, J. G. Lewis, and H. D. Simon. A shifted block Lanczos algorithm for solving sparse symmetric eigenvalue problems. *SIAM J. Matrix Anal. Appl.*, 15:228–272, 1994.

[11] D. Kalchev, C. Ketelsen, and P. S. Vassilevski. Two-level adaptive algebraic multigrid for a sequence of problems with slowly varying random coefficients. *SIAM J. Sci Comput.*, 35(6):B1215–B1234, 2013.

[12] Delyan Kalchev. Adaptive algebraic multigrid for finite element elliptic equations with random coefficients. Master's thesis, Sofia University, Bulgaria, 2012.

[13] Richard Lehoucq, Danny Sorensen, and Chao Yang. *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*. SIAM, 1998.

[14] Xiaoye Li, Jim Demmel, John Gilbert, Laura Grigori, Meiyue Shao, and Ichitaro Yamazaki. SuperLU Users' Guide. Technical Report LBNL-44289, Lawrence Berkeley National Laboratory, September 1999. http://crd.lbl.gov/~xiaoye/SuperLU/. Last update: August 2011.

[15] Y. Notay. Combination of Jacobi-Davidson and conjugate gradients for the partial symmetric eigenproblem. *Numerical Linear Algebra with Applications*, 9:21–44, 2002.

[16] Bahram Nour-Omid and Ray W. Clough. Dynamic analysis of structures using Lanczos coordinates. *Earthquake Engineering & Structural Dynamics*, 12:565–577, 1984.

[17] Beresford Parlett. *The Symmetric Eigenvalue Problem*. SIAM (Classics in Applied Mathematics), Philadelphia, USA, 1998.

[18] Beresford Parlett and Bahram Nour-Omid. The use of a refined error bound when updating eigenvalues of tridiagonals. *Linear Algebra and its Applications*, 68:179–219, 1985.

[19] Edward L. Wilson, Ming-Wu Yuan, and John M. Dickens. Dynamic analysis by direct superposition of Ritz vectors. *Earthquake Engineering & Structural Dynamics*, 10:813–821, 1982.