AN EXPERIMENTAL STUDY OF THE ALGEBRAIC MULTIGRID STRATEGIES

Marcelo Carrion

Maria Claudia Boeres

Lucia Catabriga

marcelo.tperc@gmail.com

boeres@inf.ufes.br

luciac@inf.ufes.br

Federal University of Espírito Santo

Av. Fernando Ferrari, 514, Goiabeiras, 29075-910, ES, Vitória, Brazil

Abstract. *This work presents a study of Algebraic Multigrid strategies, evaluating the performance gain that can be obtained by using them over traditional relaxation methods. The main idea behind Multigrid is to relax on successively smaller problems in a hierarchy of grids to accelerate the solution of the fine-grid problem; in the Algebraic Multigrid (AMG), no explicit knowledge of the problem geometry is needed, and the solver operates directly on the coefficient matrix of the linear system. In the present work, the SOR algorithm is employed as smoother for the AMG method, and coarse-grid level construction is implemented through three different schemes, each one combining particular coarsening and interpolation procedures. Two sets of matrices are considered in the numerical tests: one consists of stencil matrices originated from the discretization of a three-dimensional problem, while the second group has general matrices relating to miscellaneous applications. The computational experiments conducted allow for a detailed comparative analysis, showing the extent to which the Multigrid strategies can reduce the time to convergence of a classical iterative technique, with impactful results achieved in significant cases.*

Keywords: *Multigrid methods, Algebraic multigrid, Iterative methods*

1 INTRODUCTION

Fast numerical methods for solving linear systems are required in several real-world applications, that compute the solution to large sparse problems. Among those, Multigrid strategies have become quite popular (Wagner et al., 2012; Park et al., 2015), as they are employed to increase the effectiveness of relaxation techniques, which can suffer from slow convergence.

The goal of the present work is to conduct an experimental study on Multigrid methods and to measure the performance gain that can be obtained by using them over classical iterative solvers. The key idea is relaxing on successively smaller problems, in a multilevel hierarchy of so-called grids, to accelerate the solution of the fine-grid problem, with intergrid operators defined to transfer approximations between fine and coarse grids.

Originally, Multigrid was applied in limited geometric cases (Fedorenko, 1962; Briggs et al., 2000), until Algebraic Multigrid (Brandt et al., 1982, 1984; Stüben, 2001) emerged and extended the benefits enjoyed by such scheme to more general linear systems, by relying on algebraic properties inferred from the underlying system of equations. In this algebraic approach, the multilevel hierarchy has to be explicitly constructed, which includes defining coarsening and interpolation strategies (Yang, 2010). The former determines the coarse-level variables, while the latter describes an operator for mapping vectors from a coarse to a fine level.

In this context, an Algebraic Multigrid method is implemented, which considers three distinct schemes for coarse-level construction and is used directly as a solver. Several numerical experiments are carried out to test this solver, with the results collected to allow for a detailed comparative analysis.

This paper is structured in four principal sections. Section 2 introduces the Algebraic Multigrid and its central components, also explaining significant parts of the code implementation. Section 3 presents the numerical experiments that were executed, along with an analysis of the results. Finally, in Section 4, we draw our main conclusions and ideas for future work.

2 ALGEBRAIC MULTIGRID

In the original Multigrid method, the main idea to obtain a solution v for the system $Au = f$ was applying coarse-grid correction to remove smooth error, which is not eliminated by relaxation. That meant solving the residual equation $Ae = r$ (where $r = f - Av$) on a coarser grid, interpolating the error back to the fine grid and using it to correct the approximation there.

As defined by Stüben (2000) and Falgout (2006), Algebraic Multigrid (AMG) can be considered a generalization of standard geometric Multigrid: it extends in a purely algebraic manner the fundamental Multigrid principles to general sparse linear systems, by determining coarse variables, transfer operators and coarse-level equations based solely on information contained in the underlying matrix. In this approach, no explicit knowledge of the problem geometry is needed; the problem does not even have to be defined on a grid. Such feature enables the treatment of more complex problems, for which geometric techniques are either not very effective or are difficult/impossible to apply, such as equations posed on more complicated domains and unstructured grids. In short, AMG has a more general range of applicability, being able to act in the manner of a black-box solver.

This inherent flexibility comes with a cost: since no multilevel hierarchy is known a priori – derived from some underlying problem grid –, AMG has to construct a problem-dependent

hierarchy of linear systems, which is done in a *setup phase*. The setup procedure is fully automatic, building the whole coarse level by exploring the algebraic information extracted from the given system of equations, as mentioned before. This phase may also be called a preprocessing, that is required to enter the actual *solve phase*: in it, V-cycles are performed, combining smoothing and coarse-level correction to obtain an approximated solution for the problem.

The basic proceedings of a (two-level) V-cycle iteration are stated below:

- First, the original problem at the fine level is relaxed a couple of times. The residual of this current solution is then calculated and multiplied by a restriction matrix, which moves it to a coarser level.
- At such level, a different linear system is handled, one where the coarsened residual is set as the right-hand side vector and the coefficient matrix is a "coarser version" of the original matrix. This system is called a residual equation, which has as solution the error of the approximation on the top level.
- The V-cycle proceeds by applying some smoothing to this residual problem, and then interpolates – i.e., multiplies by an interpolation matrix – the error back to the finer level, using it to correct the approximation there. Finally, relaxation is repeated at the fine level.

Restriction and interpolation matrices, and a "coarser version" of the original matrix were cited, and they basically define the multilevel hierarchy AMG operates on. Their construction is based on the Galerkin (Hemker, 1982) approach, which is described next for two levels – which are easily extendable to more levels by recursion.

A sparse linear system is given by

$$A^0 u^0 = f^0 \quad \text{or} \quad \sum_{j \in \Omega^0} a_{ij}^0 u_j^0 = f_i^0 \ (i \in \Omega^0) \ , \tag{1}$$

where $\Omega^0 = \{1, 2, ..., N\}$ denotes the index set of variables of the problem. The use of the superscript 0 (zero) on the matrix and vectors specifies the finest level of the hierarchy upon which the equation is defined.

A coarse problem derived from Eq. (1) can be defined as:

$$A^1 u^1 = f^1 \quad \text{or} \quad \sum_{l \in \Omega^1} a_{kl}^1 u_l^1 = f_k^1 \ (k \in \Omega^1) \ , \tag{2}$$

where the index set of coarse-level variables Ω^1 is a subset of Ω^0 obtained by a procedure known as C/F-splitting: the fine-level set Ω^0 is split into two disjoint subsets C and F, with C accounting for the variables retained in the coarse level ($\Omega^1 = C$), called *C-variables*, and F representing the complementary *F-variables*.

Having coarsened the size of the original problem, the coarse-level matrix A^1 in Eq. (2) is given by the Galerkin operator

$$A^1 = R^0 A^0 P^0 \ , \tag{3}$$

where P^0 and R^0 are the interpolation and restriction operators, respectively. They move vectors between levels, as previously noted, and restriction is actually selected here as the transpose of interpolation:

$$R^0 = (P^0)^T \ . \tag{4}$$

Considering the preceding, there are essentially two components of the AMG method one needs to explicitly define to set up a two-level (or multilevel) hierarchy:

- a coarsening scheme, or C/F-splitting, to determine the set of coarse variables;
- an interpolation operator, for mapping coarse-level vectors into fine-level ones.

Their design is closely related, taking in consideration the nonzero entries in the problem matrix, and will be addressed in the following sections.

2.1 Coarsening and interpolation

Before explaining our coarsening and interpolation strategies, we define a few concepts that will be used subsequently.

Considering Eq. (1), a variable u_i^0 is said to be (directly) connected to another variable u_j^0 ($i, j \in \Omega^0$) if $a_{ij}^0 \neq 0$. Also, the neighborhood of such variable u_i^0 is given by:

$$N_i = \{j \in \Omega^0 \mid j \neq i, a_{ij}^0 \neq 0\} \quad (i \in \Omega^0) \; . \tag{5}$$

In the process of determining coarse levels, one should consider only matrix coefficients that are significantly large. Thus strong connections between variables are defined. A variable u_i^0 is strongly connected to u_j^0 if

$$-a_{ij}^0 \geq \varepsilon_{str} \max_{k \neq i}(-a_{ik}) \; , \tag{6}$$

where $0 < \varepsilon_{str} < 1$ is a strength threshold (Yang, 2006). Note that positive connections are considered to be weak and ignored. AMG originally assumed that A^0 is a symmetric M-matrix, i.e., a matrix that is positive definite and off-diagonally nonpositive; the definition above, however, can be applied to more general matrices, that do not stray too far from these requirements.

A second definition for strong connection between variables will also be employed in this work, which takes positive entries into account and modifies Eq. (6) by simply adopting the absolute values of the matrix coefficients:

$$|a_{ij}^0| \geq \varepsilon_{str} \max_{k \neq i}(|a_{ik}|) \; . \tag{7}$$

Correspondingly, two important sets are defined:

$$S_i = \{j \in N_i \mid u_i^0 \text{ strongly connected to } u_j^0\} \quad \text{and} \quad S_i^T = \{j \in \Omega^0 \mid i \in S_j\} \; , \tag{8}$$

which are the sets of strong connections and strong transpose connections of variable u_i^0, respectively. The first one consists of all variables u_j^0 to which u_i^0 is strongly connected, while S_i^T has the variables u_j^0 which are strongly connected to u_i^0.

A good coarsening scheme is a balancing act: it should promote a nice reduction of problem size (with Ω^1 being a reasonably small subset of Ω^0), which positively influences setup and cycling costs as well as memory requirement, but also guarantee a sufficiently strong F-to-C connectivity. Ensuring that F-variables have strong connections to C-variables means that

algebraically smooth error e_i^0 ($i \in \Omega^0$) will be well approximated by the interpolation operator, which leads to fast convergence. It is assumed that

$$(A^0 e^0)_i = 0 \quad \text{or} \quad e_i^0 = -\left(\sum_{j \in N_i} a_{ij}^0 e_j^0 \right) \Big/ a_{ii}^0 \,, \tag{9}$$

and the general idea is to obtain an interpolation formula that reasonably approximates such algebraically smooth error (Stüben, 2001).

Given the set of fine-level (variable) indices Ω^0, with $|\Omega^0|$[1] $= N$, the coarsening procedure determines two disjoint sets C and F ($\Omega^0 = C \cup F$), as mentioned before. C is selected as the set of coarse-level variables: $\Omega^1 = C$, with $|\Omega^1| = N_c$.

The interpolation operation for each fine-level variable is given by

$$e_i^0 = (P^0 e^1)_i = \begin{cases} e_i^1 & \text{if } i \in C \\ \sum_{k \in C_i} w_{ik} e_k^1 & \text{if } i \in F \,, \end{cases} \tag{10}$$

where $C_i \subset C$ is the set of *interpolatory variables* of i, i.e., the variables whose values will be used to interpolate a value at i.

Notice that C-variables interpolate their values directly, since they also exist in Ω^1; only the interpolation weights w_{ik} for the F-variables in Ω^0 need to be determined. Interpolation operator P^0, which is an $N \times N_c$ matrix, holds these weights in their corresponding positions, so that the operation performed in transferring e^1 to Ω^0 will actually be a matrix-vector multiplication:

$$e^0 = P^0 e^1 \,. \tag{11}$$

For the present work, three different strategies for building a coarse level are considered, each one combining particular coarsening and interpolation procedures. They are described in the following sections.

2.2 First coarse-level setup strategy

The first strategy is based upon direct connections between variables. C-variables are chosen such that the problem is coarsened in directions of strong connections of the matrix, and interpolation requires only immediate neighbors. Strong connections in this approach consider only negative coefficients, as in Eq. (6).

Standard coarsening. The standard C/F-splitting algorithm is proposed by Ruge & Stüben (1987). With the notion of dividing the variables $i \in \Omega^0$ into C-variables and F-variables, they all start as undecided, and have an associated weight λ_i which is initially the number of their strong transpose connections:

$$\lambda_i = |S_i^T| \,. \tag{12}$$

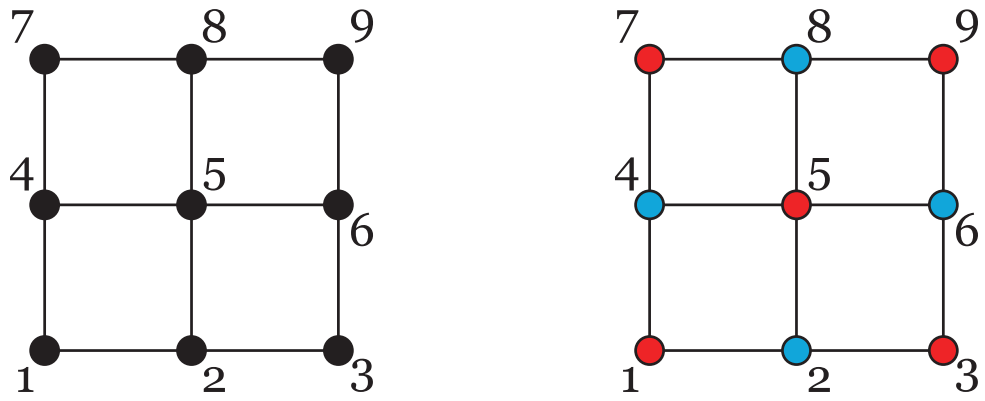
The coarsening scheme works in an iterative fashion, by selecting at each iteration an undecided variable i with maximum λ_i to become a C-variable, and turning the undecided variables

[1] $|\Omega^0|$ is the cardinality of Ω^0, i.e., the number of elements in the set.

j that are strongly connected to i into F-variables. The weight λ_k for each undecided variable k to which these new F-variables are strongly connected is then updated, increased by the number of those strong connections. This process is repeated until all variables have been decided on, assigned as either C- or F- variables.

Visual interpretation can offer a more intuitive understanding of Multigrid and this particular procedure. Suppose A^0 is a pentadiagonal matrix of order 9×9, which has the same negative value in all off-diagonal nonzero entries. Its adjacency graph can play the part of the grid Ω^0 of the algebraic problem, as seen in Fig. 1(a): Ω^0 is a directed graph whose nodes i represent the associated variables u_i^0 – or the matrix rows –, and whose edges (i, j) correspond to nonzeros a_{ij}^0 and indicate connected variables (edge orientation is omitted because, for this graph, if a directed edge (i, j) exists, then so does (j, i)).

Accordingly, for illustration, Fig. 1(b) provides the resulting C/F-splitting if the coarsening algorithm were applied to the grid previously described. Variable 5 becomes a C-variable first, since it has four strong transpose connections, as opposed to only three or two (in that grid, all connections between variables are considered strong). Such decision would turn variables labeled 2, 4, 6 and 8 into F-variables, because they are strongly connected to 5. Variables 1, 3, 7 and 9 would later be selected as C-variables, ending the iterative process.



(a) before C/F-splitting **(b) after C/F-splitting**

Figure 1: The problem grid, i.e., the adjacency graph, associated with a pentadiagonal coefficient matrix, before and after the application of the C/F-splitting procedure: red points represent C-variables, and blue points the F-variables.

Direct interpolation. In this approach, that should be used when standard coarsening is employed, the interpolation formula for each $i \in F$ is given by

$$e_i^0 = \sum_{j \in C_i} w_{ij} e_j^1 \quad \text{with} \quad w_{ij} = -\frac{a_{ij}^0}{a_{ii}^0 + \sum_{k \in N_i^+} a_{ik}^0} \frac{\sum_{k \in N_i^-} a_{ik}^0}{\sum_{k \in C_i} a_{ik}^0}, \tag{13}$$

where the set of interpolatory variables is chosen as $C_i = C \cap S_i$, i.e., the C-variables to which i is strongly connected, and N_i^+ and N_i^- correspond to positive and negative neighbors k of i (with respect to a_{ik}), respectively.

This is a very simple interpolation and often not very accurate (De Sterck et al., 2008), but it can be appropriate provided that potentially existing positive off-diagonal entries in the matrix are relatively small.

2.3 Second coarse-level setup strategy

This second strategy operates in a 'looser' manner than the first one, as coarsening will no longer require direct strong F-to-C connectivity: aggressive coarsening is applied, which allows F-variables to be strongly (negatively) connected to C-variables not directly, but via their neighbors. Direct interpolation is not suitable to this case, so multipass interpolation is used instead.

Complexity can be significantly reduced by this approach, as Ω^1 will be a smaller subset of Ω^0 than previously, but convergence is worse. It can be quite beneficial, however, to employ this scheme once, at the top level of the hierarchy (Stüben, 2000).

Aggressive coarsening. We first need to introduce the notion of distance-2 strong connections: a variable i is said to be distance-2 strongly connected to another variable j if there is a path of length less or equal to 2 connecting them, i.e., if either i is (distance-1) strongly connected to j or there exists a variable k such that $k \in S_i$ and $j \in S_k$.

Our procedure for aggressive coarsening considers strong connectivity to be of distance-2, and is implemented by employing the standard coarsening algorithm twice: the first time just as described earlier, considering only direct strong negative connections. The second one, it is applied to Ω^0 restricted to the C-variables derived from the previous step, and now exploring strong (negative) connectivity of distance-2 between these variables – via neighboring F-variables:

$$S_i^2 = \{j \in C \mid i \text{ distance-2 strongly connected to } j\} \ (i \in C) \ . \tag{14}$$

The resulting C-variables from this second step of the process will be selected as the coarse-level set of variables Ω^1.

Multipass interpolation. Multipass interpolation starts by using direct interpolation to derive interpolation formulas for all F-variables $i \in \Omega^0$ which are strongly negatively connected to C-variables ($C \cap S_i \neq \emptyset$). Not every F-variable is able to obtain an interpolation formula this way, since aggressive coarsening allowed for C-variables to be further apart.

Thus, for the remaining F-variables, the scheme proceeds in passes or steps: in each pass, interpolation weights are evaluated for those variables that are strongly negatively connected to F-variables for which interpolation weights have been determined previously. This process is repeated until every F-variable has finally derived its own interpolation formula (Stüben, 2000).

2.4 Third coarse-level setup strategy

A third approach for building the AMG coarse level uses Eq. (7) for determining strong connections between variables: both negative and positive coefficients of the matrix are taken into consideration.

The coarsening algorithm applied here is the same as standard coarsening, only now the definition of strong connections is modified so that they can also be positive. The idea is to employ this strategy for a general (matrix) case, along with the extended+i interpolation formula (De Sterck et al., 2008).

Extended+i interpolation. This particular interpolation uses an extended neighborhood for determining interpolation weights, i.e., it considers not only the strongly influencing neighbors

of each variable $i \in F$ but also their respectively strongly influencing neighbors. The interpolation formula is given by:

$$e_i^0 = \sum_{j \in C_i} w_{ij} e_j^1 \quad \text{with} \quad w_{ij} = -\frac{1}{\tilde{a}_{ii}^0}\left(a_{ij}^0 + \sum_{k \in F_i^s} a_{ik}^0 \frac{\bar{a}_{kj}^0}{\sum_{l \in C_i \cup \{i\}} \bar{a}_{kl}^0}\right),$$
$$\tilde{a}_{ii}^0 = a_{ii}^0 + \sum_{n \in N_i^w \setminus C_i} a_{in}^0 + \sum_{k \in F_i^s} a_{ik}^0 \frac{\bar{a}_{ki}^0}{\sum_{l \in C_i \cup \{i\}} \bar{a}_{kl}^0}, \tag{15}$$
$$\bar{a}_{kl}^0 = \begin{cases} 0 & \text{if sign}(a_{kk}) = \text{sign}(a_{kl}) \\ a_{kl}^0 & \text{otherwise}, \end{cases}$$

where $F_i^s = F \cap S_i$, $C_i^s = C \cap S_i$, $N_i^w = N_i \setminus (F_i^s \cup C_i^s)$ and the extended interpolatory set is defined as

$$C_i = C_i^s \cup \bigcup_{j \in F_i^s} C_j^s . \tag{16}$$

In the preceding sections, three different strategies have been defined for constructing the AMG coarse level. The first two consider only strong negative connections, with one combining standard coarsening and direct interpolation and the other using aggressive coarsening along with multipass interpolation. The third strategy, which considers all strong connections, couples standard coarsening with extended+i interpolation.

Next, the algorithm for the AMG solver is presented.

2.5 Multigrid algorithm

The Algebraic Multigrid method consists of two separate phases, as already discussed: a setup or preprocessing phase, which constructs the multilevel hierarchy of linear systems, with L coarse levels; and a solution phase, that uses the components resulting from the previous phase to perform a sequence of V-cycles, until the error is reduced below a specified tolerance.

A pseudocode for the AMG is shown in Algorithm 1, where the linear system being solved is given by Eq. (1).

In the *setup* phase, which is stated in lines 1 through 7 of the algorithm, the three main operations are: selecting the coarse-level variables, determining the interpolation operator and computing the coarse-level matrix. The sets Ω^l there indicated contain the indices of the variables that are present on each level l, $l \in \{0, 1, ..., L\}$.

The *solve* phase comprises lines 8 through 20 of Algorithm 1: during each V-cycle iteration of this phase, the main operations executed are relaxation sweeps – controlled by parameters ν_1 and ν_2 – and matrix-vector products, that transfer vectors between the multiple levels (restriction is done in line 12 and interpolation in line 17 of the code).

Note that, instead of fixing the number of coarse levels L in the method, a different approach to construct the multilevel hierarchy could be to successively coarsen the original problem ($\Omega^0 \supset \Omega^1 \supset ... \supset \Omega^L$) until Ω^L is small enough, i.e., until $|\Omega^L|$ drops below some specified threshold.

Algorithm 1: AMG(f^0, v^0, A^0)

for $l = 0$ **to** $L - 1$ **do**
  Partition Ω^l into disjoint sets C^l and F^l through coarsening;
  Set $\Omega^{l+1} = C^l$;
  Define interpolation operator P^l;
  Set $R^l = (P^l)^T$;
  Compute the Galerkin coarse-level operator $A^{l+1} = R^l A^l P^l$;
end
while *convergence not reached* **do**
  for $l = 0$ **to** $L - 1$ **do**
    Relax ν_1 times on $A^l u^l = f^l$ with initial guess v^l;
    Compute the residual $r = f^l - A^l v^l$;
    Set $f^{l+1} = R^l r$;
    $v^{l+1} \leftarrow 0$;
  end
  Relax $\nu_1 + \nu_2$ times on $A^L u^L = f^L$ with initial guess v^L;
  for $l = L - 1$ **downto** 0 **do**
    Correct $v^l \leftarrow v^l + P^l v^{l+1}$;
    Relax ν_2 times on $A^l u^l = f^l$ with initial guess v^l;
  end
end

2.6 Implementation details

AMG needs to hold in memory all the matrices involved in solving a given system of linear equations: the original matrix and every one of its coarse-level versions, along with all the different restriction and interpolation operators. To save memory usage in our code implementation, the *Compressed Row Storage* (CRS) format (Barrett et al., 1994) is employed for storing matrix data.

CRS is a compact data structure for sparse matrices that does not store any unnecessary elements, by putting the subsequent nonzeros of the matrix rows in contiguous memory locations. This format uses three one-dimensional arrays val, col_ind and row_ptr to represent a sparse matrix A. The first two arrays hold the numerical values and column indices, respectively, of the matrix nonzero elements, which are traversed in a row-wise fashion. The array row_ptr is a compound of row pointers, whose i-th element gives the position in both val and col_ind of the first nonzero element of the i-th row of A (the last nonzero is pointed by the $(i+1)$-th value of row_ptr decremented by one). A matrix with m rows containing nnz nonzero elements needs only $2nnz + (m + 1)$ storage locations in the CRS format.

In terms of memory space, the AMG framework – with L coarse levels – employed in solving a linear system of dimension N requires the following (on top of the original matrix A^0, right-hand side f^0 and solution vector v^0):

- L triplets of matrices A^{i+1}, P^i and R^i, $i \in \{0, ..., L-1\}$ (R^i is explicitly stored despite being the transpose of P^i);

- L pairs of solution vector v^i and right-hand side f^i, $i \in \{1, ..., L\}$;
- a vector r, of size N, which holds the solution residual computed at each fine level.

Although economical memory-wise, using CRS introduces some difficulties in the AMG algorithm. During the coarsening procedure, to establish the transpose connections of a given variable, one has to check for nonzeros in the matrix column, which is not an efficient operation under said storage scheme. Our approach to sidestep this issue is to hold a transpose of the matrix for executing those calculations, and then discard it when that task is finished. We are temporarily increasing memory usage but the reduction in computational time makes up for it. Also, for selecting – at each iteration of the coarsening process – the variable with maximum associated weight to become a C-variable, a Fibonacci heap (Cormen et al., 2009) is used to maintain those undecided variables.

In constructing the interpolation matrices, they are computed row by row and initially stored using linked lists, that hold the nonzero elements of each row. After the whole matrix is determined, the data in the linked lists is converted to the CRS format.

The triple matrix multiplication $R^l A^l P^l$, which is performed in the AMG setup for obtaining each matrix A^{l+1}, can be quite time consuming if not properly executed; then, an optimization is employed. Using an auxiliary matrix W, the product A^{l+1} is obtained in two steps:

$$W = R^l A^l \,, \tag{17a}$$

$$A^{l+1} = W P^l \,. \tag{17b}$$

The two separate matrix-matrix products above can each be computed using the near optimal sparse matrix-matrix multiplication algorithm proposed by Gustavson (1978). This algorithm very efficiently computes the $p \times r$ product matrix C of two sparse matrices A and B – of dimensions $p \times q$ and $q \times r$, respectively – by obtaining each row i of C in one step, as the linear combination of those rows j of B for which $a_{ij} \neq 0$:

$$c_i = \sum_{a_{ij} \neq 0} a_{ij} b_j \quad (i \in \{1, ..., p\}) \,. \tag{18}$$

Using this approach, one has only to traverse the nonzeros of matrix rows (for both A and B), which is routine in CRS, and the most prevalent operation being done is the repeated addition of a scalar multiple of one sparse vector to another sparse vector.

This procedure is basically implemented through a loop, which fills each row of matrix $C_{p \times r}$ by actually constructing an array of p linked lists, each one pertaining to a row and containing its nonzero elements (i.e., their numerical values and column indices). It utilizes two auxiliary vectors xb and x, both of size r: at each iteration of the main loop, xb is used for marking the detection of a new nonzero in a given row of C, while x holds the updated values for the nonzeros in the row being computed. The algorithm ultimately produces a matrix C in the form of linked lists, which will later be stored in the appropriate CRS scheme.

3 EXPERIMENTAL RESULTS

Every code implemented for the present work was written in the C programming language, and compiled using gcc version 4.8.2 with *-O3* optimization level. Real numbers are always

represented using double precision, and every solution method starts with zero initial guess. All numerical experiments were conducted under Ubuntu 14.04 LTS on a 64-bit machine with 64GB RAM and a four-core Intel Xeon CPU E5-46030 @ 2.00GHz processor.

For testing our Algebraic Multigrid implementation, two separate sets of experiments were performed: the first one concerns stencil matrices arising from the discretization of a three-dimensional problem, while the second batch of tests considers general matrices, i.e., matrices not (necessarily) associated with a grid.

3.1 Testing stencil matrices

As preliminary tests, we considered the three-dimensional Laplace equation

$$-\nabla^2 u(x, y, z) = 0 \,, \tag{19}$$

posed on a cubic domain, with $u(x, y, z) = 1$ on the boundaries of the cube.

Using finite differences, the domain is discretized into a uniform grid with $N = n^3$ unknowns, i.e., with n points in each spatial direction. The linear system $Au = f$ derived from this FD discretization has a heptadiagonal or 7-point stencil matrix A, with nonzero sparsity structure given by $(a_{i,i-n^2}, a_{i,i-n}, a_{i,i-1}, a_{i,i}, a_{i,i+1}, a_{i,i+n}, a_{i,i+n^2})$.

The test instances in our experiments comprise five different problem sizes: $N = 100^3$, $150^3, 200^3, 250^3, 300^3$. The AMG method is run for all of them, set to have one up to four coarse levels ($L = 1, 2, 3, 4$). To construct the multilevel hierarchy, we adopt $\varepsilon_{str} = 0.25$, a common value in the literature. The SOR algorithm is chosen as smoother for the V-cycles.

Regarding coarsening and interpolation schemes, the second coarse-level setup strategy described in Section 2 is applied exclusively at the top resolution level, to generate the first coarse level; all other coarse levels are built using the first setup strategy.

For a comparison with AMG, the test instances are also solved using the traditional SOR algorithm (Hadjidimos, 2000), configured with $\omega = 1.8$ (selected after some calibration) and convergence tolerance fixed at 10^{-8}. The results from these computations are shown in Table 1, where the first two columns define the problem being treated – N is the system dimension and nnz is the number of nonzeros in the coefficient matrix –, while the columns labeled *iter* and *time* provide, respectively, the number of iterations and CPU time, in seconds, to obtain a solution. Notice that the iterations count increases significantly with problem size and that, for the largest instance ($N = 300^3$), the method does not even converge in the maximum number of iterations fixed here, which is ten thousand.

Table 1: Computational results for the SOR method, with $\omega = 1.8$, $tol = 10^{-8}$.

N	nnz	iter	time (s)
100^3	6,940,000	1,511	90.68
150^3	23,490,000	3,194	579.36
200^3	55,760,000	5,411	2,342.05
250^3	109,000,000	8,132	6,687.23
300^3	188,460,000	10,000	14,310.14

Tables 3 and 4 present the computational results from the tests using the AMG solver. Information on the size of coarse levels is shown below in Table 2 for every problem instance (N), with the dimension N_c and number of nonzeros nnz_c given for each coarse-level matrix.

Table 2: AMG configuration of coarse levels, obtained by using $\varepsilon_{str} = 0.25$ in the setup phase; the first one is constructed using the second coarse-level setup strategy, while the others result from the first setup scheme.

N	1st coarse level		2nd coarse level		3rd coarse level		4th coarse level	
	N_c	nnz_c	N_c	nnz_c	N_c	nnz_c	N_c	nnz_c
100^3	83,333	1,417,179	15,071	534,921	2,531	135,197	400	20,328
150^3	281,255	4,784,603	50,499	1,824,193	8,319	465,897	1,202	67,404
200^3	666,674	11,341,382	140,334	5,971,186	26,006	2,123,072	3,674	324,894
250^3	1,302,082	22,149,146	232,431	8,557,135	38,178	2,281,926	5,485	355,523
300^3	2,250,006	38,271,614	400,921	14,788,681	64,260	3,742,270	9,087	584,161

In all tests using the AMG method, which may be called AMG-SOR as it employs SOR for smoothing, the following applies: $\omega = 1.8$ (for fair comparison with the SOR results), $\nu_1 = \nu_2 = 2$ and the desired tolerance is again fixed at 10^{-8}. One-two and three-four coarse levels are considered in solving each test instance, with the results for each case (L) shown in Tables 3 and 4, respectively. In those tables, the CPU times – measured in seconds – taken in the *setup* and *solve* phases of the algorithm are discriminated, and the column *total* brings the sum of these two (and *iter* gives the number of V-cycles executed in the solve phase). The columns labeled % provide the percentage time reduction achieved by a given AMG *total* with respect to the SOR *time* in Table 1.

Table 3: Computational results for the AMG-SOR, with $L = 1$ and 2, $\omega = 1.8$, $\nu_1 = \nu_2 = 2$, $tol = 10^{-8}$.

N	**One coarse level** ($L = 1$)					**Two coarse levels** ($L = 2$)				
	setup (s)	iter	solve (s)	total (s)	%	setup (s)	iter	solve (s)	total (s)	%
100^3	5.39	96	18.64	24.03	73.51	7.51	42	10.31	17.82	80.35
150^3	19.67	202	132.29	151.95	73.77	35.41	63	50.03	85.43	85.25
200^3	47.83	338	529.61	577.44	75.34	65.86	101	187.91	253.77	89.16
250^3	100.16	501	1,545.12	1,645.28	75.40	125.90	157	535.18	661.08	90.11
300^3	180.84	692	4,073.48	4,254.31	70.27	218.86	214	1,441.13	1,659.99	88.40

Table 4: Computational results for the AMG-SOR, with $L = 3$ and 4, $\omega = 1.8$, $\nu_1 = \nu_2 = 2$, $tol = 10^{-8}$.

N	**Three coarse levels** ($L = 3$)					**Four coarse levels** ($L = 4$)				
	setup (s)	iter	solve (s)	total (s)	%	setup (s)	iter	solve (s)	total (s)	%
100^3	8.77	42	15.02	23.79	73.76	6.38	42	8.83	15.21	83.22
150^3	24.36	42	30.98	55.33	90.45	22.84	42	39.39	62.23	89.26
200^3	70.65	44	78.46	149.12	93.63	58.18	42	72.89	131.07	94.40
250^3	128.79	60	224.75	353.54	94.71	128.46	42	152.79	281.25	95.79
300^3	224.25	80	494.16	718.41	94.98	228.56	44	272.99	501.55	96.50

A careful analysis of the results obtained by AMG, in solving the stencil matrix problems, allows for some interesting observations. First, setup time does not rise dramatically with the number of coarse levels: the coarsening and interpolation procedures, as well as the triple matrix product to determine coarse matrices, are being called more times, but they operate on increasingly smaller matrices.

Moreover, although significant, setup time accounts for only a portion of total computation time, which is dominated by solve time. And the convergence gain achieved in the solve phase is evident, and clearly worth the preprocessing time – even with a V-cycle being more expensive than one SOR iteration. In short, the setup phase is not a hindrance to AMG-SOR, which shows a noticeable improvement over standard SOR, reducing the latter's time by more than 70% in all test cases considered here.

We point out that AMG set with just one coarse level already beats SOR in both iterations count and processing time – to better visualize the behavior of both algorithms when the problem size increases, see Fig. 2 –, but adding more levels makes the V-cycles count shown in the tables decrease and get more 'constant'. Comparing the number of V-cycles performed (for the five test instances) by setting one and four coarse levels in the AMG method, they drop from 96; 202; 338; 501; 692 to 42; 42; 42; 42; 44, respectively.

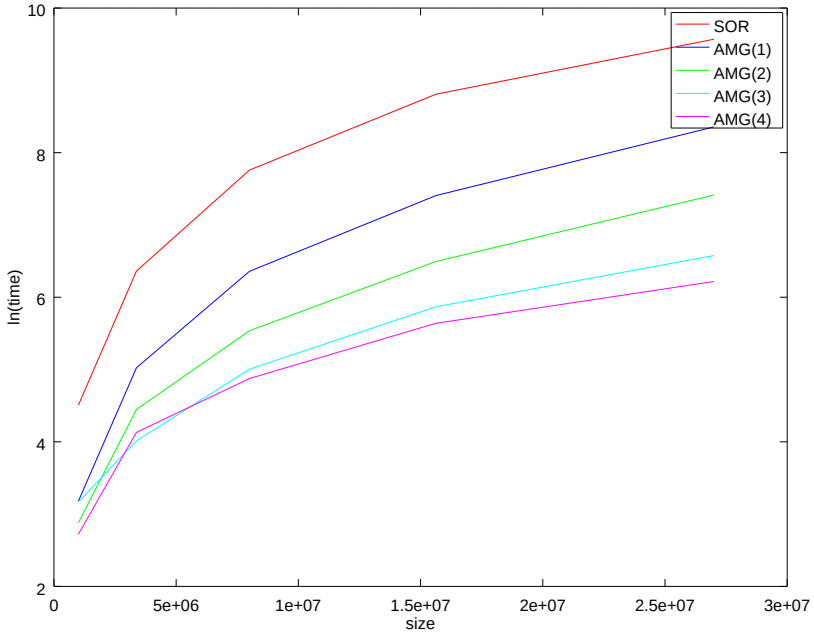


Figure 2: The natural logarithm of the CPU time, in seconds, for computing the solution using SOR and the four AMG(L) methods ($L = 1, 2, 3, 4$), plotted against problem size N.

For the AMG execution considering $L = 4$, one can practically say convergence is independent of problem size (iterations count is near constant), which is the goal with Multigrid schemes, that is reached here algebraically. The experiments aptly display how Multigrid enables a simple relaxation method to accelerate its convergence with very impactful results.

3.2 Testing general matrices

In our second set of experiments with the Algebraic Multigrid, computational tests are performed for 14 matrices: 11 of them obtained from the matrix collection of the University of Florida (Davis & Hu, 2011) – relating to miscellaneous applications – and the other three in-house produced, arising from finite element problems. They are described in Table 5, where the dimension N and number of nonzeros nnz are given for each of them, with an asterisk * marking the ones which were generated in-house.

These test matrices will each serve as the coefficient matrix for the linear system $Au = f$, of dimension N, to be solved here. The right-hand side vector f will be given by:

$$f_i = \sum_{a_{ij} \neq 0} a_{ij} \quad (i \in \{1, ..., N\}) \,, \tag{20}$$

which means the exact solution u for the problem is the vector with all ones.

The intent of the experiments in this section is to measure the performance of AMG-SOR – i.e., AMG using SOR as smoother – against the SOR method, in solving the problems associated with some general matrices, as proposed above. The coarse-level setup strategy employed in this case, for all fine levels, is the third scheme defined in Section 2.

Table 5 shows the results from using SOR as a solver. The parameter ω and the convergence tolerance tol, empirically chosen, are indicated for each test matrix; tolerance was increased for

matrices Cavity1 and Cavity2 because they were difficult to converge. The columns labeled *iter* and *time* give the resulting CPU time (in seconds) and number of iterations, respectively. It is worth noting that all problems converged to a solution, except for the one – highlighted in red – involving matrix Cavity2, that reached the maximum number of 20 thousand iterations without convergence.

Table 5: Computational results for the SOR method, with ω and tol specified for each test matrix.

Matrix	N	nnz	ω	tol	iter	time (s)
cage10	11,397	150,645	1.5	10^{-8}	36	0.04
FEM_3D_thermal1	17,880	430,740	1.5	10^{-8}	83	0.19
Cavity1*	25,033	485,041	0.5	10^{-4}	10,863	23.50
cage11	39,082	559,722	1.5	10^{-8}	36	0.16
Cavity2*	67,203	1,176,073	0.5	10^{-4}	20,000	110.64
cage12	130,228	2,032,536	1.5	10^{-8}	37	0.55
FEM_3D_thermal2	147,900	3,489,300	1.5	10^{-8}	95	1.51
CoupCons3D	416,800	17,277,420	0.5	10^{-8}	668	49.36
cage13	445,315	7,479,343	1.5	10^{-8}	37	1.54
atmosmodd	1,270,432	8,814,880	1.5	10^{-8}	4,023	255.91
atmosmodj	1,270,432	8,814,880	1.5	10^{-8}	6,289	399.93
atmosmodl	1,489,752	10,319,760	1.5	10^{-8}	1,227	90.90
cage14	1,505,785	27,130,349	1.5	10^{-8}	38	5.95
pudding*	3,379,006	23,634,744	0.5	10^{-8}	8,628	4,468.88

The AMG-SOR is likewise executed for the matrices defined above. The method is initially configured as: $L = 1$, $\varepsilon_{str} = 0.25$, $\nu_1 = \nu_2 = 2$, and reusing the values for ω and tol determined in Table 5. The results from these experiments are shown in Table 6, where the columns N_c and nnz_c give, respectively, the dimension and the number of nonzeros of the coarse-level matrix built for each test case. The CPU times, in seconds, spent in the *setup* and in the *solve* phases, and the *total* AMG computation time are presented along with the corresponding number of V-cycles (*iter*).

Table 6: Computational results for the AMG-SOR method, with $L = 1$, $\varepsilon_{str} = 0.25$, $\nu_1 = \nu_2 = 2$ and ω, tol the same as before.

Matrix	N_c	nnz_c	setup (s)	iter	solve (s)	total (s)
cage10	2,547	160,541	0.12	9	0.05	0.17
FEM_3D_thermal1	4,737	288,833	0.30	15	0.18	0.47
Cavity1	6,225	385,597	0.45	1,313	15.43	15.89
cage11	8,389	657,021	0.53	9	0.15	0.67
Cavity2	11,399	453,285	0.62	3,903	90.31	90.93
cage12	26,930	2,679,810	2.12	9	0.58	2.70
FEM_3D_thermal2	35,359	2,503,795	2.21	20	1.57	3.78
CoupCons3D	61,109	8,261,431	11.30	129	41.36	52.66
cage13	88,438	10,491,278	8.93	10	2.46	11.38
atmosmodd	635,216	16,534,896	7.03	282	110.82	117.85
atmosmodj	635,216	16,534,896	6.77	444	175.14	181.90
atmosmodl	744,876	13,828,000	9.05	89	37.84	46.89
cage14	288,861	41,188,005	37.38	10	10.11	47.48
pudding	1,410,346	19,788,216	53.12	761	1,759.00	1,812.12

AMG-SOR using one coarse level was able to obtain a solution for all matrices tested, including Cavity2 which had previously failed to converge. The six results highlighted in blue are cases where AMG successfully reduced the computation time of SOR, i.e., where the corresponding *total* in Table 6 is smaller than *time* in Table 5.

After this initial batch of experiments, more tests were performed using the AMG-SOR. The method was configured to employ one up to four coarse levels, aiming to achieve better time reductions relative to SOR. Table 7 organizes the findings from this investigation, showing for each of the six matrix cases highlighted above, combined with a number of coarse levels ($L = 1, 2, 3, 4$): the number of V-cycle iterations *iter* and the percentage time reduction (%), with respect to SOR, attained by the AMG solution.

Table 7: Computational results comparison for the AMG-SOR method, with $\varepsilon_{str} = 0.25$, $\nu_1 = \nu_2 = 2$ and ω, tol the same as before: one vs. two vs. three vs. four coarse levels ($L = 1, 2, 3, 4$).

Matrix	One coarse level		Two coarse levels		Three coarse levels		Four coarse levels	
	iter	%	iter	%	iter	%	iter	%
Cavity1	1,313	32.39	1,094	36.30	1,024	38.05	995	40.93
Cavity2	3,903	17.81	†		†		†	
atmosmodd	282	53.95	66	75.01	13	80.46	13	77.70
atmosmodj	444	54.52	103	78.18	21	86.22	13	85.67
atmosmodl	89	48.41	16	58.70	12	43.59	12	38.82
pudding	761	59.45	421	72.25	331	75.72	316	75.93

In Table 7, cells filled with † refer to experiments that failed to converge. The results highlighted in blue indicate, for each matrix considered, the test case which generated the best computational result (regarding the time reduction).

A good metric to evaluate the improvement of AMG over SOR is comparing the number of iterations for both methods. Since one V-cycle, with $\nu_1 = \nu_2 = 2$, performs four (top-level) SOR relaxations while just one is executed in a SOR iteration, the number of V-cycles run by AMG-SOR must be smaller than $\frac{1}{4}$ the number of SOR iterations, in order for the Multigrid strategy to even begin gaining performance over standalone SOR.

Five of the six previously successful AMG cases improved their computation times by adding coarse levels, with only Cavity2 benefiting exclusively from using just one coarse level. Cavity1, on the other hand, consistently and gradually improved its performance through the increment of coarse levels, achieving a time reduction of almost 41% with $L = 4$.

The three atmosmod matrices – atmosmodd, atmosmodj, atmosmodl – started, respectively, with 282, 444 and 89 V-cycles executed to obtain a solution, and ultimately converged in 13, 13 and 12 iterations, when employing four coarse levels. For the first two matrices, $L = 3$ seems enough to reach the best AMG execution, with time reduction above 80% for both cases, while for atmosmodl (which was the 'easiest' problem of the three) two coarse levels suffice: adding more levels further reduces the number of iterations to convergence, but only from 16 to 12 V-cycles, and entails computational work which increases final computation time.

The pudding matrix, which had the slowest associated problem to solve, achieves the best AMG time reduction with four coarse levels: originally, in the SOR method, the problem took 8,628 iterations and almost 4,500 seconds to converge; then, using AMG-SOR configured with $L = 4$, it obtained a solution in under 1,100 seconds (in 316 V-cycles).

Thus far, we have only discussed the AMG results where computation time was improved compared to SOR. Looking closely at Table 6, however, there are eight test cases for which the use of Multigrid was not beneficial. This can be explained by either setup time already being higher than SOR time, or the reduced number of iterations not being sufficient to offset setup

and cycling costs (remember that V-cycles are more expensive than single SOR iterations).

The case with matrices cage10, FEM_3D_thermal1, cage11 and cage12, for instance, is that they are too rapidly solved by the SOR method – all in under one second. Matrix CoupCons3D, on the other hand, has quite a significant solve time under SOR (of almost 50 seconds), which AMG is able to reduce, but not enough to make up for setup taking more than 11 seconds. This example comes the closest to reducing SOR time of the unsuccessful AMG experiments.

Finally, the problem associated with matrix cage14 obtained the worst result in AMG-SOR compared to SOR: the time spent in the setup and solve phases are, even separately, much higher than the time taken by the SOR method. There is just too much computational work being done for such a simple problem – while SOR computation time is under six seconds, AMG setup time alone is above 37 seconds.

One final observation can be made regarding these experiments with general matrices. For every successful AMG test case but Cavity1, the computation time of SOR was higher than 90 seconds (the unsuccessful cases all ran under 50 seconds). It might be obvious to say, but a Multigrid technique should be applied when it is actually necessary, i.e., when there is time to be gained and the need to accelerate solution convergence.

4 CONCLUSIONS AND FUTURE WORK

In this work, we studied the multilevel technique known as Multigrid, which is applied to accelerate the convergence of iterative methods used for solving linear systems. The central idea is that smooth error, i.e., error that is not eliminated by a relaxation scheme, must be removed by coarse-grid correction. Our discussion focused on Algebraic Multigrid, which determines coarse levels not based on problem geometry but explicitly, using information that is inferred from the coefficient matrix of the linear system.

The Algebraic Multigrid algorithm, being of general purpose, demands significant computational work to be done as preprocessing. Since it does not know the multilevel hierarchy a priori, the method builds a coarse level by performing a number of operations involving the coefficient matrix. We considered different variations of AMG, which was applied initially to solve a 'geometric' problem (associated with a stencil matrix). This AMG solver, using SOR as smoother and four coarse levels, reduced the computation time of standard SOR by more than 80% in the tests executed. We are also able to, in an algebraic context, reach the theoretical property expected from Multigrid, which is convergence independent of problem size.

A second phase of AMG experiments, that involved general matrices, allowed us to evaluate the performance of the method for distinct problems. The Multigrid strategy displayed a varied behavior in this case, with several matrices not benefiting from its application. Nonetheless, the AMG solver managed to significantly improve the convergence in six out of 14 problems considered, with time reduction ranging from 40% to 86% in comparison to SOR.

Our study of Algebraic Multigrid is still in the early stages, though. For future work, a main goal is applying AMG as a preconditioner, that we expect to produce good results for Krylov methods. We want to also explore and compare different strategies for coarsening and interpolation, and are looking to further optimize our AMG algorithm – by using interpolation truncation (Stüben, 2000) and employing parallelism, for example.

REFERENCES

Barrett, R., Berry, M., Chan, T. F., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C. & der Vorst, H. V., 1994. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, 2nd edn, SIAM, Philadelphia, PA.

Brandt, A., McCormick, S. F. & Ruge, J., 1982. Algebraic multigrid (AMG) for automatic multigrid solution with application to geodetic computations. Submitted to: SIAM Journal on Scientific and Statistical Computing.

Brandt, A., McCormick, S. F. & Ruge, J., 1984. *Algebraic multigrid (AMG) for sparse matrix equations*, Cambridge University Press, Cambridge, pp. 257–284.

Briggs, W. L., Henson, V. E. & McCormick, S. F., 2000. *A Multigrid Tutorial (2nd Ed.)*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C., 2009. *Introduction to Algorithms, Third Edition*, 3rd edn, The MIT Press.

Davis, T. A. & Hu, Y., 2011. The University of Florida Sparse Matrix Collection, *ACM Transactions on Mathematical Software* 38(1), 1–25.
URL: *http://www.cise.ufl.edu/research/sparse/matrices*

De Sterck, H., Falgout, R. D., Nolting, J. W. & Yang, U. M., 2008. Distance-two interpolation for parallel algebraic multigrid, *Numerical Linear Algebra with Applications* 15(2–3), 115–139.

Falgout, R. D., 2006. An Introduction to Algebraic Multigrid, *Computing Science and Engineering* 8(6), 24–33.

Fedorenko, R. P., 1962. A relaxation method for solving elliptic difference equations, *USSR Computational Mathematics and Mathematical Physics* 1(4), 1092–1096.

Gustavson, F. G., 1978. Two Fast Algorithms for Sparse Matrices: Multiplication and Permuted Transposition, *ACM Transactions on Mathematical Software* 4(3).

Hadjidimos, A., 2000. Successive overrelaxation (SOR) and related methods, *Journal of Computational and Applied Mathematics* 123(1–2), 177–199. Numerical Analysis 2000. Vol. III: Linear Algebra.

Hemker, P. W., 1982. A note on defect correction processes with an approximate inverse of deficient rank, *Journal of Computational and Applied Mathematics* 8(2), 137–139.

Park, J., Smelyanskiy, M., Yang, U. M., Mudigere, D. & Dubey, P., 2015. High-performance Algebraic Multigrid Solver Optimized for Multi-core Based Distributed Parallel Systems, *in* 'Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis', SC '15, ACM, New York, NY, USA, pp. 54:1–54:12.

Ruge, J. W. & Stüben, K., 1987. *Algebraic multigrid (AMG)*, Vol. 3 of *Frontiers in Applied Mathematics*, Society for Industrial and Applied Mathematics, Philadelphia, PA, pp. 73–130.

Stüben, K., 2000. *Algebraic multigrid (AMG): an introduction with applications*, Academic Press, New York.

Stüben, K., 2001. A review of algebraic multigrid, *Journal of Computational and Applied Mathematics* 128(1–2), 281–309.

Wagner, M., Rupp, K. & Weinbub, J., 2012. A Comparison of Algebraic Multigrid Preconditioners using Graphics Processing Units and Multi-Core Central Processing Units, *in* 'Proceedings of the 2012 Symposium on High Performance Computing', HPC '12, Society for Computer Simulation International, San Diego, CA, USA, pp. 2:1–2:8.

Yang, U. M., 2006. *Parallel Algebraic Multigrid Methods - High Performance Preconditioners*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 209–236.

Yang, U. M., 2010. On Long Range Interpolation Operators for Aggressive Coarsening, *Numerical Linear Algebra with Applications* 17(2–3).