2023 31st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP) | 979-8-3503-3763-1/23/$31.00 ©2023 IEEE | DOI: 10.1109/PDP59025.2023.00017

AMG Preconditioners based on Parallel Hybrid Coarsening and Multi-objective Graph Matching

Pasqua D'Ambra
Institute for Applied Computing "M. Picone" (IAC)
National Research Council (CNR)
Napoli, Italy
0000-0003-2047-4986

Fabio Durastante
Department of Mathematics
University of Pisa
Pisa, Italy
0000-0002-1412-8289

S M Ferdous
Physical and Computational Sciences
Pacific Northwest National Laboratory
Richland, WA, USA
0000-0002-2323-4753

Salvatore Filippone
Dept. of Civil and Computer Engineering
University of Rome Tor-Vergata
Rome, Italy
0000-0002-5859-7538

Mahantesh Halappanavar
Physical and Computational Sciences
Pacific Northwest National Laboratory
Richland, WA, USA
0000-0002-2323-4753

Alex Pothen
Computer Science Department
Purdue University
West Lafayette, IN, USA
0000-0002-3421-3325

***Abstract*—We describe preliminary results from a multi-objective graph matching algorithm, in the coarsening step of an aggregation-based Algebraic MultiGrid (AMG) preconditioner, for solving large and sparse linear systems of equations on high-end parallel computers. We have two objectives. First, we wish to improve the convergence behavior of the AMG method when applied to highly anisotropic problems. Second, we wish to extend the parallel package `PSCToolkit` to exploit multi-threaded parallelism at the node level on multi-core processors. Our matching proposal balances the need to simultaneously compute high weights and large cardinalities by a new formulation of the weighted matching problem combining both these objectives using a parameter λ. We compute the matching by a parallel $2/3-\varepsilon$-approximation algorithm for maximum weight matchings. Results with the new matching algorithm show that for a suitable choice of the parameter λ we compute effective preconditioners in the presence of anisotropy, i.e., smaller solve times, setup times, iterations counts, and operator complexity.**

***Index Terms*—Sparse solvers, AMG, Matching, MPI, OpenMP, Scalability**

I. Introduction

We describe an improved iterative method to solve a large linear system of the form $A\mathbf{x} = \mathbf{b}$, where $A \in \mathbb{R}^{n\times n}$ is a sparse matrix, i.e., a matrix with $O(n)$ nonzero entries and $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$, when A is obtained from an anisotropic problem, on current pre-exascale parallel computers. More precisely, we describe a new bi-objective matching algorithm to compute the coarsening step of an aggregation-based Algebraic Multigrid (AMG) preconditioner to accelerate the convergence of a Krylov-type linear solver [1]. A fast and potentially parallel approximation algorithm is used to compute the matching.

The new matching balances the need to obtain large weights as well as high cardinalities in the weighted adjacency graph of the system matrix; this is to ensure that the AMG preconditioner is effective in reducing solve time and memory overhead for increasing numbers of parallel cores while leaving at each level as few unmatched vertices as possible. The final AMG shows improved node-level efficiency and scalability when compared with other available parallel iterative linear solvers on linear systems with up to 192×10^6 unknowns on up to 1024 computing cores of the *Marconi 100* Supercomputer.

In Section II, we describe the AMG preconditioner and the main challenges to improving the quality and parallel efficiency of its setup. In Section III, we formulate the Lambda Matching to compute a bi-objective graph matching and demonstrate its *pareto optimality*. Furthermore, we describe an approximation algorithm for solving the Lambda Matching problem. In Section IV, we present our approach in applying the approximated Lambda Matching for parallel hybrid shared/distributed-memory implementation of the AMG setup. Section V includes a discussion of the preliminary results obtained on benchmark test cases arising from anisotropic diffusion problems.

II. AMG preconditioner based on graph matching

The Algebraic MultiGrid method considered here applies to the case of a symmetric and positive-definite matrix A, and is a general stationary iterative method:

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + B(b - A\mathbf{x}^{(k-1)}), \quad k \geq 1 \text{ for } \mathbf{x}^{(0)} \in \mathbb{R}^n,$$

where $B \in \mathbb{R}^{n\times n}$ is defined recursively as follows. Let A_l be a hierarchy of coarse matrices computed by the triple-matrix Galerkin product:

$$A_{l+1} = P_l^T A_l P_l, \quad l = 0, \ldots, \ell - 1, \tag{1}$$

with $A_0 = A$ (the fine matrix) and P_l a sequence of prolongation matrices of size $n_l \times n_{l+1}$, with $n_{l+1} < n_l$ and $n_0 = n$. Here ℓ is the number of hierarchy levels. Let M_l be an A-convergent smoother for A_l, i.e., an operator for which $\|I_l - M_l^{-1}A_l\|_{A_l} < 1$, where I_l is the identity matrix of size n_l, and $\|\cdot\|_{A_l}$ indicates the weighted A_l norm. The preconditioner matrix B for the $V(1,1)$ cycle, where one sweep of pre– and post-smoothing step is applied, is the linear operator corresponding to the

2377-5750/23/$31.00 ©2023 IEEE
DOI 10.1109/PDP59025.2023.00017

multiplicative composition of the following error propagation matrices:

$$I - B_l A_l = (I - (M_l)^{-T} A_l)(I - P_l B_{l+1} (P_l)^T A_l)$$
$$(I - M_l^{-1} A_l) \; \forall l < \ell,$$

assuming that $B_\ell \approx A_\ell^{-1}$ is an approximation of the inverse of the coarsest-level matrix.

Our AMG method relies on an aggregation strategy for the setup of the coarse matrices: at each level it uses disjoint aggregates of unknowns to define the sequence of prolongation and coarse matrices, and exploits the properties of maximum weight matching in undirected edge-weighted graphs, as detailed in [2], [3]. Parallel versions of the method have been recently proposed [4], [5] for hybrid architectures embedding Graphics Processing Units (GPUs) both in a single node and in multiple node settings; they benefit from efficient parallel algorithms for the approximate computation of a maximum weight matching in a graph, with implementations available as source code [6], [7].

In this paper, we extend our previous work to improve the quality and parallel efficiency of the AMG setup. More specifically, we pursue a twofold objective: on the one hand, we wish to improve the convergence behavior of our AMG method when applied as a preconditioner of a Krylov solver for highly anisotropic problems, while on the other hand, we wish to extend our previous parallel implementations to exploit multi-threaded parallelism at the node level on multi-core processors (both in the setup and in the application of the AMG method).

A. Prolongators from compatible weighted matching

An earlier version of the AMG method [2], [3] used *a coarsening based on compatible weighted matching* for setup of the AMG hierarchy (1). This is a recursive procedure starting from the weighted adjacency (undirected) graph $G = (\mathcal{V}, \mathcal{E}, C)$ associated with the sparse matrix A, where the vertex set $\mathcal{V}$ consists of the row/column indices of A and the edge set $\mathcal{E}$ corresponds to the index pairs (i, j) of the nonzero entries in A. We consider a weight matrix C computed from the matrix A and an arbitrary vector $\mathbf{w}$, as follows:

$$(C)_{i,j} = c_{i,j} = 1 - \frac{2a_{i,j} w_i w_j}{a_{i,i} w_i^2 + a_{j,j} w_j^2}, \tag{2}$$

where $a_{i,j}$ are the entries of A and $\mathbf{w} = (w_i)_{i=1}^n$ is a given vector.

A *matching* $\mathcal{M}$ in a graph G is a subset of edges such that no two edges are incident on the same vertex. By applying a matching algorithm to the adjacency graph G of A, we can define the aggregates $\{\mathcal{G}_j\}_{j=1}^{n_p}$ for the row/column indices $\mathcal{I}$ of matrix A to consist of matched pairs of indices, where $n_p = |\mathcal{M}|$ is the cardinality of the graph matching $\mathcal{M}$. Equivalently, we are decomposing the index set as

$$\mathcal{I} = \bigcup_{i=1}^{n_p} \mathcal{G}_i, \quad \mathcal{G}_i \cap \mathcal{G}_j = \emptyset \text{ if } i \neq j.$$

A *perfect matching* is one in which all rows and columns of A (vertices of G) are matched. In the case of a non-perfect matching (a graph may not have a perfect matching, or the algorithm may have computed a matching with less than maximum cardinality) we will have *unmatched* vertices. In this case, each unmatched vertex corresponds to a singleton G_i; we denote by n_s the number of unmatched vertices. For each edge $e = (u, v) \in \mathcal{M}$, we can identify the vectors:

$$\mathbf{w}_e = \frac{1}{\sqrt{w_i^2 + w_j^2}} \begin{bmatrix} w_i \\ w_j \end{bmatrix}.$$

Given the above vectors, and assuming an ordering of the indices that moves all the unknowns corresponding to unmatched vertices at the bottom, we can formally define a *prolongator*:

$$P = \begin{bmatrix} \tilde{P} & O \\ O & W \end{bmatrix} = [\mathbf{p}_1, \ldots, \mathbf{p}_{n_c}], \tag{3}$$

where $\tilde{P} = \text{blockdiag}(w_{e_1}, \ldots, w_{e_{n_p}})$, $W = \text{diag}(w_l/|w_l|)$, $l = 1, \ldots, n_s$, and the number of vertices at the coarse level is $n_c = n_p + n_s$.

The matrix P we have just built is a piecewise constant interpolation operator whose range includes, by construction, the vector $\mathbf{w}$. In large-scale applications we generally combine multiple steps of the basic pairwise aggregation by computing the product of m consecutive prolongators of type (3), to obtain a more aggressive coarsening; the resulting aggregates merge multiple pairs and have size 2^m, thus producing a coarsening ratio at each level equal to $n/n_c \approx 2^m$.

A way of improving this strategy is represented by the use of a *smoothed aggregation* procedure: we post-process the prolongator P obtained from the matching strategy, to obtain an operator with higher regularity than the piecewise constant. At level l we perform such an operation by applying a weighted Jacobi smoother to the matrix P_l

$$P_l^s = (I - \omega D_l^{-1} A_l) P_l, \tag{4}$$

where A_l is the coarse matrix of the corresponding level, D_l denotes its diagonal, and ω is a damping parameter approximating the spectral radius of $D_l^{-1} A_l$.

A quantitative measure of the memory footprint of the multigrid hierarchies and an estimated cost of the application of a V-cycle is given by the *operator complexity*

$$\text{opc} = \frac{\sum_{l=0}^{\ell-1} \text{nnz}(A_l)}{\text{nnz}(A_0)} > 1,$$

that gives a measure of the arithmetic intensity of the matrix-vector and matrix-matrix products involved in building and applying the multigrid algorithm.

III. The multi-objective matching strategy

The original version of the *coarsening based on compatible weighted matching* used a *maximum product matching* in the weighted graph G, i.e., $\mathcal{M} = \arg\max_{\mathcal{M}'} \prod_{(i,j)\in\mathcal{M}'} c_{ij}$, where $\mathcal{M}'$ denotes a matching in G. This idea has been inspired by the so-called compatible relaxation principle, originally introduced

in [8] as a general way to obtain good-quality coarsening in AMG. A simple logarithmic weight transformation allows us to formulate the computation of a maximum product matching in terms of the maximum weight sum matching, that is, a matching that maximizes the sum of the weights of the edges in the matching. An exact algorithm, i.e., an algorithm that computes the maximum value of the objective function, is impractical for very large graphs with billions of edges; therefore, parallel approximation algorithms have been designed [9]. The `PSCToolkit` software framework[1] [10] relies on a parallel distributed-memory algorithm, included in the `MATCHBOX-P` software library [6], which computes a $1/2$-approximate matching, i.e., a matching which realizes at least half of the optimal weight for all input graphs.

While the above approximation algorithm works well for isotropic problems (see [5]), in the case of anisotropic problems it is often unable to detect the direction of the strong coupling among unknowns in the first level of the AMG hierarchy, creating a large number of unmatched nodes and thus hampering the compression properties of the coarsening strategy; the bad quality of the aggregates affects negatively the convergence properties of the AMG preconditioner.

Is it possible to attain high cardinalities in the matching while also obtaining large weights so that the aggregation has better quality, and the AMG algorithm has better convergence properties? We demonstrate that the answer is yes for anisotropic problems, by considering results from experiments with a parallel bi-objective approximate algorithm for matching.

A. Lambda matching

Consider a graph $G(\mathcal{V}, \mathcal{E}, C)$, where $\mathcal{V}$ and $\mathcal{E}$ are the vertex and edge sets respectively, and C is a positive weight function defined on the edges. We denote the number of vertices $|\mathcal{V}|$ by n and the number of edges $|\mathcal{E}|$ by m throughout this section.

The classic formulation of maximum weight matching maximizes only the weight function and does not consider the cardinality of the matching. We can reformulate the matching problem to account for the cardinality by introducing a non-negative parameter λ, and modifying the formulation to

$$\max \sum_{e\in\mathcal{E}} C(e)x(e) + \lambda \sum_{e\in\mathcal{E}} x(e) \qquad (5)$$
$$\text{subject to} \quad \sum_{e\in\delta(v)} x(e) \le 1, \forall v \in V, \text{ and } x(e) \in \{0,1\}.$$

Here $\delta(v)$ is the set of edges incident on vertex v and x is the characteristic vector of the matching. The formulation (5), called LAMBDA MATCHING, has two terms in its objective function: the first term maximizes the weight of matching, while the second term maximizes its cardinality. The parameter $\lambda \ge 0$ incorporates the trade-off between these two objectives. If $\lambda = 0$, the algorithm produces the classic maximum weight matching. It is intuitive that increasing λ would also increase the cardinality of the matching and possibly decrease the weight of the matching in terms of the original edge weights.

[1]See https://psctoolkit.github.io for the code.

B. Pareto Optimality of Weight and Cardinality

Let $\mathcal{M}_*^\lambda$ be an optimum solution (i.e. a matching), and W_λ be the optimum objective value of 5. Let also W_0 be the total weight of a matching (say, $\mathcal{M}^\lambda$) on the original weight of the graph, i.e., $W_0(M^\lambda) = \sum_{e\in\mathcal{M}^\lambda} C(e)$. To show the Pareto optimality between weight and cardinality, we have to prove that:

1) the weight $W_0(\mathcal{M}_*^\lambda)$ is largest among matchings with cardinality $|\mathcal{M}_*^\lambda|$, and
2) the cardinality of $\mathcal{M}_*^\lambda$ is the highest among matchings with weight $W_0(\mathcal{M}_*^\lambda)$.

Note that if $\lambda = 0$, the second condition may not always be satisfied. For example, consider a path graph with four vertices $\{a, b, c, d\}$ with edge weights $(a, b) = (c, d) = 1$ and $(b, c) = 2$. This has two maximum weight matchings, namely $(a, b), (c, d)$ and (b, c). For $\lambda = 0$ the LAMBDA MATCHING could return the latter matching, thus violating the second condition.

Lemma 1: For $\lambda > 0$, the matching computed by solving LAMBDA MATCHING is Pareto optimal w.r.t weight and cardinality.

Proof: (Proof of weight:) By contradiction. Suppose there exists another matching $\mathcal{M}_1^\lambda$ with $|\mathcal{M}_1^\lambda| = |\mathcal{M}^\lambda|$ but $W_0(\mathcal{M}_1^\lambda) > W_0(\mathcal{M}^\lambda)$. Then we have $W_\lambda(\mathcal{M}_1^\lambda) > W_\lambda(\mathcal{M}^\lambda)$, and $\mathcal{M}_*^\lambda$ cannot be an optimum matching.

(Proof of cardinality:) By contradiction. Suppose we have a matching $\mathcal{M}_1^\lambda$ with weight $W_0(\mathcal{M}_1^\lambda) = W_0(\mathcal{M}_*^\lambda)$ but $|\mathcal{M}_\lambda^1| > |\mathcal{M}_*^\lambda|$. As $\lambda > 0$, we see $\lambda|\mathcal{M}_\lambda^1| > \lambda|\mathcal{M}_*^\lambda|$. Again $W_\lambda(\mathcal{M}_1^\lambda) > W_\lambda(\mathcal{M}_*^\lambda)$, which is a contradiction. ■

C. Choosing a suitable value of λ

Given a matching $\mathcal{M}$ of a graph, a path or cycle P is called *alternating* if it consists of edges chosen alternatively from $\mathcal{M}$ and $\mathcal{E} \setminus \mathcal{M}$. An *augmenting path* with respect to the current matching is an alternating path that begins and ends with unmatched vertices; by switching matching edges to non-matching edges and *vice versa* we can increase the cardinality of the matching. Now we can develop some guiding principles for choosing the value of λ.

Lemma 2: Let γ be the maximum weight and δ be the minimum weight of the edges in G. If $\lambda = \max\{\frac{(k-1)}{2}\gamma - \frac{(k+1)}{2}\delta, 0\} + \varepsilon$, where $\varepsilon > 0$, then $\mathcal{M}_*^\lambda$ obtained from LAMBDA MATCHING has the maximum weight among all matchings such that there exists no augmenting path of length $k(\ge 3)$ or less w.r.t the matching.

Proof: Due to space limitations, we provide a sketch of the proof. Suppose choosing λ as mentioned in the lemma does not lead to a maximum weight matching with the augmenting path length guarantee. Then there is an augmenting path, say P, of length k w.r.t to the optimal matching $\mathcal{M}_*^\lambda$. In P, there are $\frac{k-1}{2}$ matched edges and $\frac{k+1}{2}$ unmatched edges. Let Δ be the change of weight if we augment along P. The proof is completed by showing that $\Delta \ge 0$, which leads to a contradiction since we began with a maximum weight matching. ■

D. $2/3 - \varepsilon$ Lambda matching algorithm

The theoretical analysis of LAMBDA MATCHING technique is tailored to the optimal solution of the corresponding matching problem. The optimal algorithms are tedious to implement due to the complex data structures needed to maintain blossoms (odd cycles in the graph with the maximum number of matching edges on the cycle) efficiently, and have little to no concurrency. For the large problems that we need to solve during the setup of AMG hierarchies, it is of the utmost importance to compute matchings exploiting the large memories available on parallel computers; the matchings must also be computed fast. In practice, approximation algorithms compute matchings with weights quite close to the optimal matchings; see the survey article [9] for a thorough discussion of the advantages of approximate matchings over exact matchings. For these reasons, we choose to employ approximate weighted matching in this application.

A *weight increasing path (cycle)* is an alternating path (cycle) with equal numbers of matching and non-matching edges such that the sum of the weights of the non-matching edges is greater than the sum of the weights of the matching edges. By switching matching and non-matching edges on the path (cycle), we can increase the weight of the current matching. An *augmentation* is an augmenting path or a weight increasing path or cycle; if P is an augmentation, then $\mathcal{M} \oplus P = (\mathcal{M} \setminus P) \cup (P \setminus \mathcal{M})$ is also a matching. The *gain* of an alternating path or cycle P is $g(P) = W(P \setminus \mathcal{M}) - W(P \cap \mathcal{M})$. For $k \geq 1$, a *k-augmentation* is an augmentation containing *exactly* k edges not in $\mathcal{M}$.

A matching that does not admit any positive gain l-augmentation, where $1 \leq l \leq k$, is $1 - 1/(k+1)$-approximate (see a proof in [9]). Pettie and Sanders [11] extended this result by introducing vertex-disjoint 2-augmentations. Using these they developed a $2/3 - \varepsilon$ approximate randomized algorithm that runs in $O(m \log \frac{1}{\varepsilon})$ time. This iterative algorithm works by choosing a random vertex v and augmenting the current matching with the largest gain 1- and 2-augmentations centered at v in each iteration. A variant of this random algorithm was implemented by Maue and Sanders [12]. This new algorithm works in phases, where in each phase, the algorithm successively selects all the vertices in some random order, and the current matching is augmented with a highest-gain 2-augmentation *centered at the selected vertex*. Berge and Manne have a parallel multi-threaded implementation [13] of this new algorithm, which we have used with modifications as a building block for setting up the AMG hierarchies.

IV. PARALLEL COARSENING BASED ON LAMBDA MATCHING

In extending `AMG4PSBLAS` software for using the parallel multi-threaded version of the LAMBDA MATCHING discussed in the above sections, we applied a *decoupled approach* for running in a hybrid shared/distributed-memory programming model based on the OpenMP and MPI libraries. This means that every MPI rank applies the matching to the local sub-graph corresponding to the local matrix, ignoring edges which involve graph nodes owned by different MPI ranks. We observe that this decoupled approach introduces a further approximation in the parallel hybrid AMG setup, indeed the union of local matchings on local subgraphs is in general not a $(2/3 - \varepsilon)$-approximate matching for the global graph. The setup of the prolongator in (3) and the corresponding restrictor is local on each MPI rank, but the triple matrix Galerkin products to compute the resulting coarse matrices involve data communications among MPI ranks.

Using the parallel matching algorithm, it is possible to run each MPI task with multiple threads, thus speeding up the setup phase, even though it is not fully parallelized at this time because of improvements needed in the parallel sparse matrix by sparse matrix product kernel; a fully multi-threaded version is currently under development. The current version of our AMG software provides OpenMP parallelization for the main vector operations and sparse matrix by vector products, so that the solve phase can also run in a multi-threading setting when using highly parallel smoothers. This is the case of versions of the weighted-Jacobi and block-Jacobi methods with approximate inverses on diagonal blocks, as discussed in [4], [5]. We observe that this hybrid implementation of the AMG setup would also be useful in the current CUDA-enabled version of the library; indeed, the solve phase runs on the CUDA-enabled device, whilst the setup phase can be sped up by employing OpenMP support on multi-core CPUs commonly found in GPU-enabled compute nodes.

V. NUMERICAL EXPERIMENTS

All the experiments are performed using a Poisson benchmark with axial anisotropy: we solve the boundary value problem

$$\begin{cases} -\nabla(K\nabla u) = f, & \in \Omega \\ u \equiv 0, & \in \partial\Omega, \end{cases} \tag{6}$$

in 2D and 3D. We consider two cases. In the first case, $K = \mathrm{diag}(k_{ii})$ is a diagonal diffusivity tensor. For the second case, we consider only a 2D problem with $K = \mathrm{diag}([1, \delta])$ rotated by an angle θ. Here Ω is the unit square or the unit cube for the axial anisotropies, and the $[-1, 1] \times [-1, 1]$ square for the rotated ones. We use the standard second-order finite-difference scheme to discretize the problem, obtaining well-known matrices with 5 and 7 diagonals in 2D and 3D, respectively for the axially oriented case, while we use the $\mathbb{Q}_1$-Lagrangian elements on a regular Cartesian grid for the rotated anisotropy.

In the following discussion, we will focus on the two major steps in the solution process. On the one hand we consider the *setup time* of multigrid hierarchies; this comprises the construction of the auxiliary graph, the graph matching, the construction of the coarse matrix hierarchy together with the projection/restriction operators, and of the smoothers at each level of the hierarchy. On the other hand we consider the *solve time*, that is, the time necessary to apply a Krylov iteration using the multigrid hierarchy as a preconditioner. In the application

of sparse linear solvers it is usually the case that a single setup phase is reused over many applications/linear system solutions.

A. The effect of the λ parameter

In this section, we show results when the Poisson equation in 2D is solved in a sequential setting. We use axial anisotropy corresponding to two diffusivity tensors $K_1 = \text{diag}(1000, 1)$ and $K_2 = \text{diag}(1, 1000)$, and analyze the impact of the λ value on the aggregates generated by the *coarsening based on compatible weighted matching*. To achieve this we consider a small number of discretization mesh points, 10 per direction for a total of $n = 100$ unknowns (dofs, degrees of freedom). In Table I we show the main parameters of the 2-level AMG hierarchy, i.e., operator complexity (opc) and coarsening ratio cr $= n/n_c$, where n is the dimension of the original matrix and n_c is the dimension of the coarse-level matrix; aggregates of size at most 2 are built and no smoothing is applied to the resulting prolongator. We also report the number of iterations, nit, of the preconditioned Conjugate Gradient (PCG) method for solving the system, when the AMG hierarchy is applied as a V-cycle with one iteration of the Gauss-Seidel method as pre-/post-smoother and LU factorization as coarsest solver. The linear solver is stopped when a relative residual norm is less than 10^{-6}.

In our experiments we varied the λ value in the objective function, following the discussion in Section III-C for optimal matchings, with the augmentation length $k = 3$. We chose λ by using the following formula, which depends on an input real parameter s:

$$\lambda = \begin{cases} s\varepsilon + (1-s)\Lambda & \text{if } s \in]0,1] \ \wedge \ \Lambda > 0; \\ s & \text{if } s = 0 \ \vee \ s > 1; \\ \varepsilon & \text{otherwise,} \end{cases} \tag{7}$$

where ε is the machine precision and

$$\Lambda = (\max(\max(c_{ij}) - 2.0\min(c_{ij}), 0.0)) + \varepsilon,$$

with c_{ij} denoting graph edge weights as in (2) after a logarithmic transformation of the weights. As input parameter, we used values of $s = ih \in [0, 2], \ i = 0, 1 \ldots$, with $h = 0.25$. We note that the value $s = 0$ corresponds to $\lambda = 0$, so the resulting matching is a $2/3 - \epsilon$-approximate matching in the graph with no cardinality considerations. If $s \in]0, 1]$, we have $\lambda \in [\varepsilon, \Lambda]$, and $\lambda = s$ if $s > 1$.

We observed similar convergence behavior of the solver for all values of input $s \in [0, 1]$, while, when we fix $s = \lambda > 1$ we observed worse behavior for all values except for $\lambda = 1.75$, where better coarsening parameters are observed for both diffusivity tensors, and better convergence behavior is seen for tensor K_1. In Table I we show parameters for three different representative values of λ, as obtained by (7). In the case of $K = K_1$ with $\lambda = 1.75$, we observe that the resulting AMG hierarchy requires a smaller number of iterations in the solve phase, even though it has a smaller operator complexity. In the case of $K = K_2$, for $\lambda = 1.75$, we observe the smallest operator complexity and the best coarsening ratio, which corresponds to a cheaper hierarchy, although the number of iterations in the solve phase does not change with respect to the case of $\lambda = 1.25$. The smallest number of iterations (only one iteration fewer) is observed when $\lambda = 0$, although for that choice we observe the smallest coarsening ratio.

TABLE I: AMG parameters and number of iterations for several values of λ in the objective function (5).

$K = K_1$			
λ	opc	cr	nit
0	1.552	1.923	9
1.25	1.559	1.961	10
1.75	1.489	1.961	7

$K = K_2$			
λ	opc	cr	nit
0	1.589	1.887	6
1.25	1.604	1.923	7
1.75	1.489	1.961	7

B. Weak scaling analysis

The experiments presented here and in the next section were performed on some nodes of the CINECA Marconi 100 machine, equipped with 2×16 cores IBM POWER9 16C AC922 CPUs, 256GB of RAM per node, and Dual-rail Mellanox EDR Infiniband connection network. Taking into account this computing unit layout, we perform a weak scaling analysis by associating 16 threads with each MPI process while assigning $\sim$ 3M dofs to each MPI task and using from 1 to 64 MPI tasks for a total of 1024 computing cores and 192M total dofs.

We perform a weak scalability analysis on the model problem (6) in 3D, with an axial anisotropy along z corresponding to a diagonal diffusivity tensor $K = \text{diag}(1, 1, 1000)$. We use the PCG method coupled with a V-cycle where four iterations of ℓ_1-Jacobi are used as pre- and post-smoother. This is a modification of the highly parallel weighted Jacobi method in which the weights are computed by looking at the ℓ_1-norm of the off-diagonal entries of the matrices. It usually has better smoothing properties than the weighted Jacobi method; see, e.g., the discussion in [14] and the experiment in [5]. The solver on the coarse grid is again the ℓ_1-Jacobi method of which we do 30 iterations. The tentative prolongator P is built as in (3) by using in a recursive way the multi-objective matching discussed in Section III and applied as described in Section IV. The formula in (7) is applied at each new iteration of the matching algorithm to the weighted graph associated with the current level coarse matrix. To have a more aggressive aggregation we compose together three consecutive prolongators, making the target size for the aggregates equal to 8. To obtain the final prolongator P_l^s at each level a single sweep of damped Jacobi is applied to obtain (4). The linear solver is stopped when a relative residual norm is less than 10^{-6}.

Following the analysis in Section V-A, we consider three cases: $\lambda = 0$, $\lambda = 1.25$, and $\lambda = 1.75$. First of all, in Fig. 1 we observe that the hierarchy built employing the hybrid matching ($\lambda = 1.75$) has a lower operator complexity, as it was in the previous case. This means that we have an effective coarsening at each level in which fewer unmatched nodes and lower weight matchings appear. The more evident effect is then a reduction in the setup-time for the preconditioner with respect to the other cases, as seen in Fig. 2. Fewer singletons means that

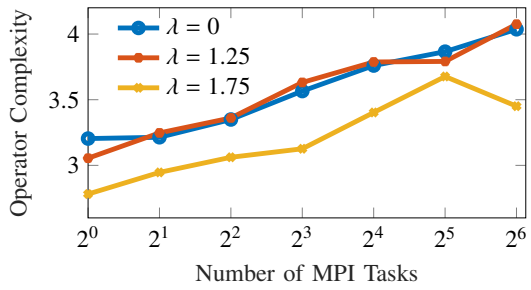


Fig. 1: Weak scaling. Operator Complexity.

the graphs at succeeding levels are smaller; thus the matching algorithm has fewer operations to perform and is then faster.

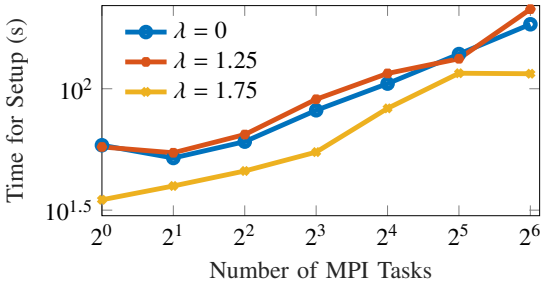


Fig. 2: Weak scaling. Setup time

Clearly, from the point of view of the problem that we want to solve, the other requirement that we need to satisfy is the quality of such an aggregation procedure, and we can measure it by looking at the number of iterations needed by the PCG method to reach a tolerance of 10^{-6} on the residual in Fig. 3. We

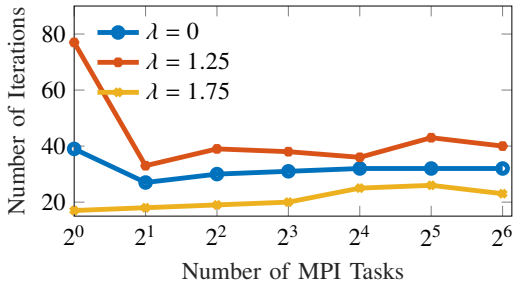


Fig. 3: Weak scaling. Number of iterations for the preconditioned CG.

observe that reducing the number of singletons by increasing the cardinality of the matching while simultaneously obtaining a good weight for the matching leads to good algorithmic scalability for the proposed preconditioner. Finally, we observe that such good properties are reflected in the solution time reported in Fig. 4. Here increasing cardinality in the multi-objective matching results in an AMG preconditioner that outperforms the original approach based on maximum weight matching alone.

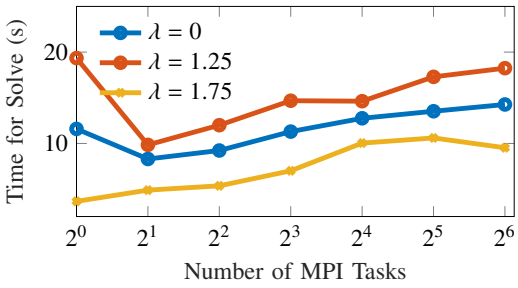


Fig. 4: Weak scaling. Solve time

C. Strong scaling analysis

For the strong scalability analysis we consider again the same preconditioner and application case used for the weak scalability case in Section V-B. The total number of dofs we use is $288^3 \approx 11M$. We vary the number of MPI tasks from 2^k for $k = 0, \ldots, 5$, and employ 16 OpenMP threads for task. With this choice in the coarser case with 64 MPI tasks, we end up having $\approx 185,000$ dofs per task. To evaluate the performance of the matching algorithms we look again at the complexity of the operator in the different cases. From Fig. 5 we observe as before that the case with $\lambda = 1.75$ attains the smallest operator complexity. This corresponds in turn in a smaller setup-time for

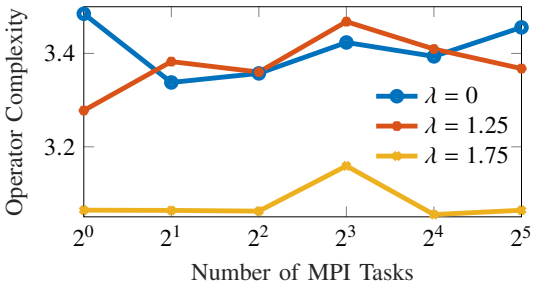


Fig. 5: Strong scaling. Operator complexity

the preconditioner, as shown in Fig. 6, attaining a speed up of 6.4 with $\lambda = 1.75$. The speedup is computed relative to a single MPI task. Along with the algorithmic and scalability properties of matching, the approximation capabilities of the multigrid hierarchy as a preconditioner are also preserved. Indeed, we observe a stable number of iterations for the linear PCG solver (Fig. 7) and a corresponding solution time (Fig. 8). The parallel algorithm attains a speedup of 13.4 for the $\lambda = 1.75$ choice, relative to a single MPI task.

D. Node-level analysis and some comparison

In this section, we focus on the 2D version of problem (6) with a rotated anisotropy, with $\theta = 18°$ and $\varepsilon = 100$, and compare the obtained results with several algorithms, including some state-of-the-art libraries through PETSc, while running

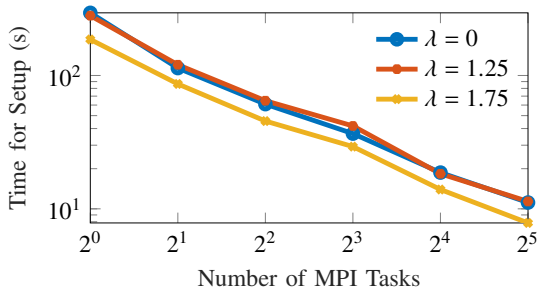


Fig. 6: Strong scaling. Setup time

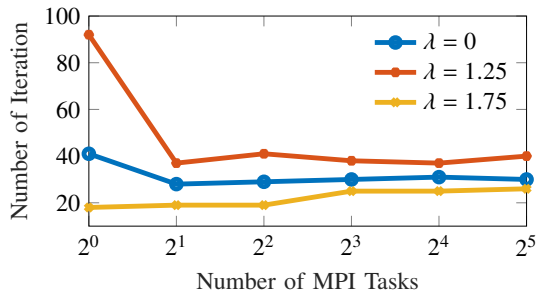


Fig. 7: Strong scaling. Number of iterations for the preconditioned CG.

the example on a single node with different load balancing between the number of OpenMP threads and MPI tasks. For this analysis we use a single node with 48 Intel® Xeon® Gold 6238R CPUs at 2.20GHz, and we use up to 32 cores. We divide them into either pure MPI tasks or pure OpenMP threads and consider a weak scaling framework with around 512k dofs per core.

The algorithms we compare are:

- LAMBDA MATCHING with 4 iterations of ℓ_1-Jacobi as smoother, a smoothed V-cycle, and `MUMPS` as coarse solver. For the aggregation, we consider aggregates of size 8. For this algorithm we test both the pure MPI task and the pure OpenMP thread settings;
- **Parmatch** with 4 iterations of ℓ_1-Jacobi as smoother, a smoothed V-cycle, and `MUMPS` as coarse solver. For the aggregation, we consider aggregates of size 8. In this case, they are obtained with the pure MPI version of the Matching algorithm from [5];
- **VBM** with 4 iterations of ℓ_1-Jacobi as smoother, a smoothed V-cycle, and `MUMPS` as coarse solver. For the aggregation procedure we use the decoupled aggregation based on the algorithm from [15] as implemented;

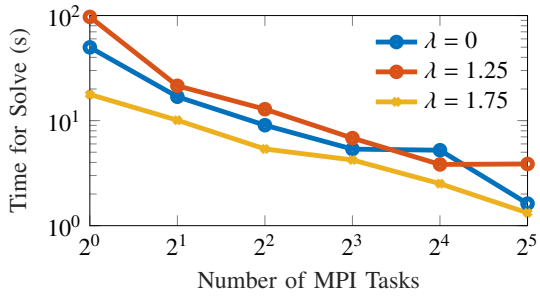


Fig. 8: Strong scaling. Solve time

and the following algorithms from `Hypre` and `Trilinos/ML` through the PETSc interface

- **Falgout** the AMG preconditioner from the `Hypre/Boomeramg` package using 4 iterations of ℓ_1-Jacobi as smoother, the Falgout coarsening scheme with classical interpolation, and Gaussian-elimination as coarse solver;
- **HMIS1** the AMG preconditioner from the `Hypre/Boomeramg` package using 4 iterations of ℓ_1-Jacobi as smoother, the HMIS coarsening scheme with one level of aggressive interpolation, and Gaussian-elimination as coarse solver;
- **ML** the AMG preconditioner from the `Trilinos` library using as smoother 4 iterations of the combination of one Chebyshev iteration preconditioned by a single sweep of Jacobi, the aggregation algorithm from [15], and Gaussian-elimination as coarse solver;

We stress that for PETSc there is no available OpenMP version with optimized performances[2], thus to avoid unfair comparisons we use the optimized compiled version with pure MPI tasks.

In Fig. 9 we compare the results in terms of operator complexities and the number of iterations. The information regarding the LAMBDA MATCHING is in the leftmost and central panels. On the left, we have the cases in which we scale with pure MPI tasks, on the central the ones in which we scale by using threads.

To complete this analysis we turn again to *weak-scaling* setting. We use up to $N = 16$ nodes of Marconi-100 with two MPI tasks per node. Leveraging the results obtained on a single node, we employ for the LAMBDA MATCHING 16 threads per MPI task corresponding to the processor layout of the machine. The load per MPI task is set to have around 512k dofs. We test again all the discussed strategies and use them as measures of the building times for the multigrid hierarchies, the solve times, the time per iteration, and the number of iterations. To avoid cluttering the plots, we restrict ourselves to the cases of $\lambda = 0$, and $\lambda = 1.56$ that were delivering the two best results in the central panels of Figure 9. All the results are given in Fig. 10. We observe a better solve time of our LAMBDA MATCHING with respect to the Hypre HMIS1 algorithm having operator complexity which is comparable with our AMG hierarchies, and a marked improvement with respect to **VBM** and **Parmatch**. In the face of a very high

[2]"The core PETSc team has come to the consensus that pure MPI using neighborhood collectives and the judicious using of MPI shared memory (for data structures that you may not wish to have duplicated on each MPI process due to memory constraints) will provide the best performance for HPC simulation needs on current generation systems, next-generation systems and exascale systems."; see https://petsc.org/release/miscellaneous/threads/

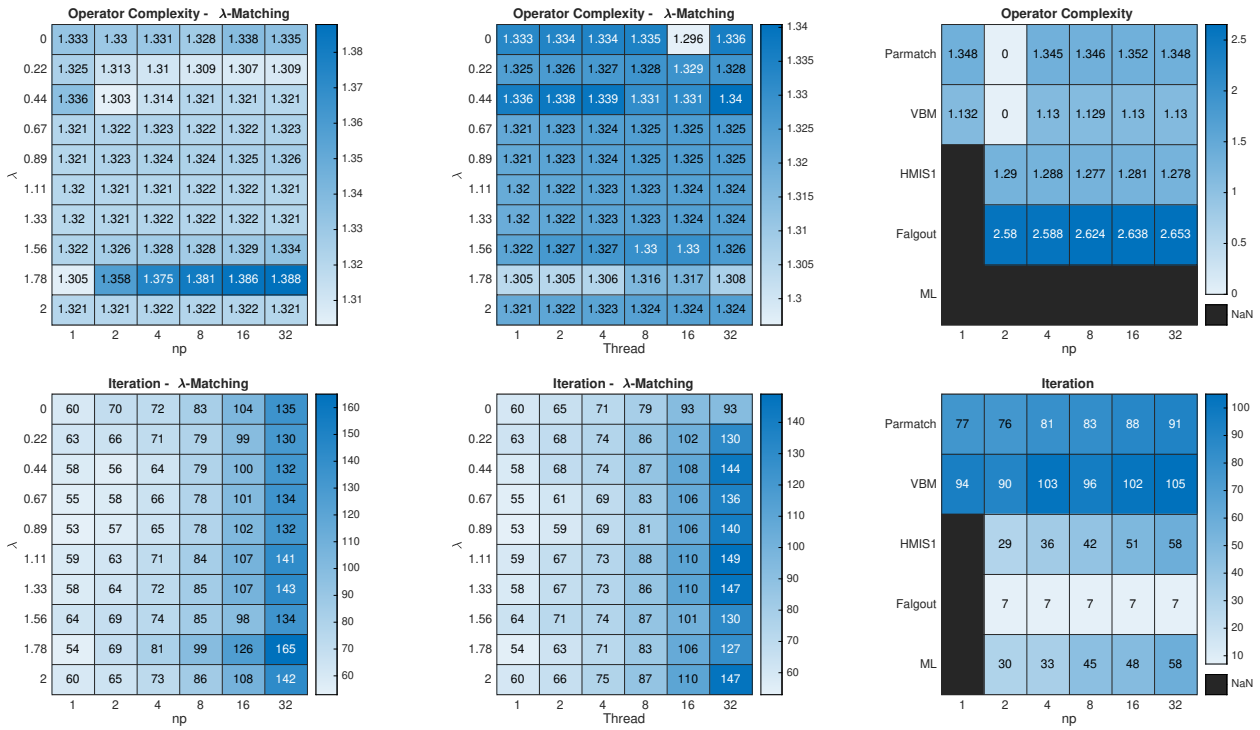
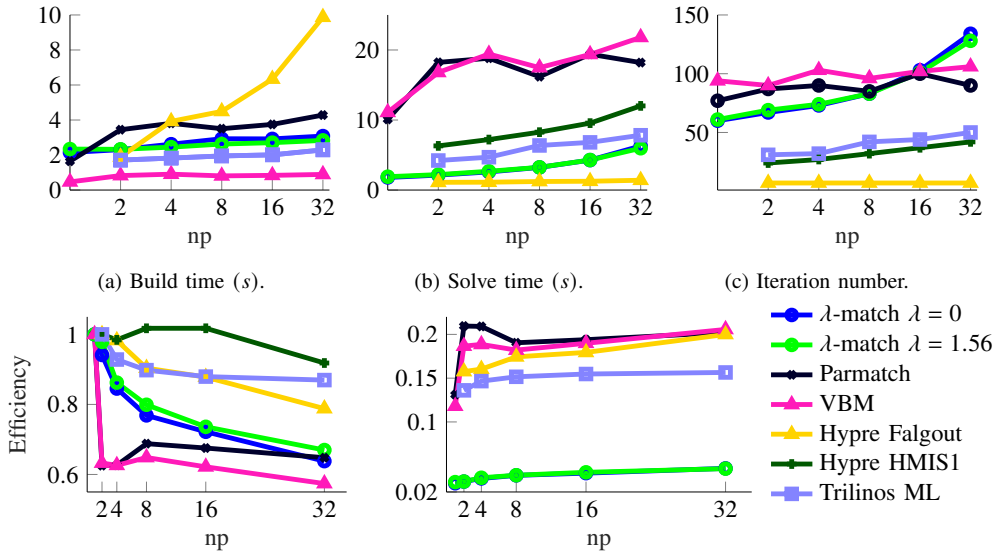


Fig. 9: 2D rotated anisotropy with $\theta = 18°$ and $\varepsilon = 100$. Comparisons of the operator complexities (top panels), and of the number of iterations (bottom panels). The sequential runs with the algorithms from PETSc hang indefinitely, and we have terminated them. The operator complexities for ML are not computed by the interface. Size of the aggregates for LAMBDA MATCHING and Parmatch is 8.



(a) Build time (s). (b) Solve time (s). (c) Iteration number.

(d) Time per iteration (Efficiency) (e) Time per iteration (s).

Fig. 10: Weak scaling - 512k dofs per MPI task. Rotated anisotropy with $\theta = 18°$, $\varepsilon = 100$. We use up to $N = 16$ nodes of Marconi-100, with $np = 2$ MPI tasks per node, and 16 OpenMP threads per task for the multigrid based on the LAMBDA MATCHING algorithm. The efficiency for the case of the algorithms from Hypre and from Trilinos is computed with the baseline time per iteration the run on two processes, since the run on a single process does not terminate its execution.

construction cost Hypre Falgout has the best solve time and iteration count.

VI. Conclusions

In this paper we discussed some preliminary results on using a new matching algorithm in the setup of an AMG preconditioner for sparse linear solver. This new matching algorithm is an approximation strategy balancing maximum weight and maximum cardinality, its principal aim is to accelerate the coarsening procedure, i.e., decreasing the operator complexity of the AMG hierarchy, while retaining a good iteration count in the solve phase. The new matching algorithm is implemented by exploiting a shared memory programming model based on OpenMP. It was integrated in the MPI-based library for AMG preconditioners inside a decoupled aggregation scheme for coarsening. This reduces accuracy in the approximate matching on the overall sparse matrix while allowing the exploitation of a hybrid programming model aimed to obtain high-performance both at the node level and in the distributed setting. We plan to use this hybrid implementation of the AMG setup to enhance the current CUDA-enabled version of the library so that on the one hand we run the solve phase on the available CUDA-enabled devices of each node; on the other hand, we equally distribute the work in building the AMG hierarchy among the cores on the nodes on CUDA-enabled devices.

Acknowledgment

The research is supported by the EU Horizon 2020 Project, Towards EXtreme scale Technologies and Accelerators for euROhpc hw/Sw Supercomputing Applications for exascale (TEXTAROSSA, ID 956831), as part of the EuroHPC initiative; the U.S. DOE Exascale Computing Project's (ECP) (17-SC-20-SC) ExaGraph codesign center and Laboratory Directed Research and Development Program at Pacific Northwest National Laboratory (PNNL); and DOE grant DE-SC-0022260 at Purdue. S M Ferdous is grateful for the support of the Linus Pauling Distinguished Postdoctoral Fellowship program. We also thank the anonymous reviewers for detailed comments and suggestions to improve the manuscript.

References

[1] Y. Saad, *Iterative Methods for Sparse Linear Systems*. Boston: PWS Publishing Company, 1996. [Online]. Available: http://www-users.cs.umn.edu/~saad/books.html

[2] P. D'Ambra and P. S. Vassilevski, "Adaptive AMG with coarsening based on compatible weighted matching," *Comput. Vis. Sci.*, vol. 16, no. 2, pp. 59–76, 2013. [Online]. Available: https://doi.org/10.1007/s00791-014-0224-9

[3] P. D'Ambra, S. Filippone, and P. S. Vassilevski, "BootCMatch: a software package for bootstrap AMG based on graph weighted matching," *ACM Trans. Math. Software*, vol. 44, no. 4, pp. Art. 39, 25, 2018. [Online]. Available: https://doi.org/10.1145/3190647

[4] M. Bernaschi, P. D'Ambra, and D. Pasquini, "AMG based on compatible weighted matching for GPUs," *Parallel Comput.*, vol. 92, pp. 102 599, 13, 2020. [Online]. Available: https://doi.org/10.1016/j.parco.2019.102599

[5] P. D'Ambra, F. Durastante, and S. Filippone, "AMG preconditioners for linear solvers towards extreme scale," *SIAM J. Sci. Comput.*, vol. 43, no. 5, pp. S679–S703, 2021. [Online]. Available: https://doi.org/10.1137/20M134914X

[6] U. V. Catalyürek, F. Dobrian, A. Gebremedhin, M. Halappanavar, and A. Pothen, "Distributed-memory parallel algorithms for matching and coloring," in *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, 2011, pp. 1971–1980.

[7] F. Manne and M. Halappanavar, "New Effective Multithreaded Matching Algorithms," in *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, 2014, pp. 519–528.

[8] A. Brandt, "General highly accurate algebraic coarsening," in *Electron. Trans. Numer. Anal. Multilevel methods (Copper Mountain, CO, 1999)*, 2000, vol. 10, pp. 1–20.

[9] A. Pothen, S. M. Ferdous, and F. Manne, "Approximation algorithms in combinatorial scientific computing," *Acta Numerica*, vol. 28, pp. 541–633, 2019. [Online]. Available: https://doi.org/10.1017/s0962492919000035

[10] P. D'Ambra, F. Durastante, and S. Filippone, "Parallel Sparse Computation Toolkit," *Software Impacts*, vol. 15, p. 100463, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2665963822001476

[11] S. Pettie and P. Sanders, "A simpler linear time $2/3 - \epsilon$ approximation for maximum weight matching," *Inform. Process. Lett.*, vol. 91, no. 6, pp. 271–276, 2004. [Online]. Available: https://doi.org/10.1016/j.ipl.2004.05.007

[12] J. Maue and P. Sanders, "Engineering algorithms for approximate weighted matching," in *WEA*, vol. 7. Springer, 2007, pp. 242–255.

[13] A. Berge, "A parallel version of the Random Order Augmentation Matching Algorithm," Master's thesis, University of Bergen, 2020.

[14] A. H. Baker, R. D. Falgout, T. V. Kolev, and U. M. Yang, "Multigrid smoothers for ultraparallel computing," *SIAM J. Sci. Comput.*, vol. 33, no. 5, pp. 2864–2887, 2011. [Online]. Available: https://doi.org/10.1137/100798806

[15] P. Vaněk, J. Mandel, and M. Brezina, "Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems," in *Computing. International GAMM-Workshop on Multi-level Methods (Meisdorf, 1994)*, 1996, vol. 56, no. 3, pp. 179–196. [Online]. Available: https://doi.org/10.1007/BF02238511