Parallel Aggregation Based on Compatible Weighted Matching for AMG

Ambra Abdullahi[1(✉)], Pasqua D'Ambra[2], Daniela di Serafino[3], and Salvatore Filippone[4]

[1] Università degli Studi di Roma "Tor Vergata", Roma, Italy
ambra.abdullahi@uniroma2.it
[2] Istituto per le Applicazioni del Calcolo "Mauro Picone", CNR, Naples, Italy
pasqua.dambra@cnr.it
[3] Università degli Studi della Campania "Luigi Vanvitelli", Caserta, Italy
daniela.diserafino@unicampania.it
[4] Cranfield University, Cranfield, UK
salvatore.filippone@cranfield.ac.uk

Abstract. We focus on the extension of the MLD2P4 package of parallel Algebraic MultiGrid (AMG) preconditioners, with the objective of improving its robustness and efficiency when dealing with sparse linear systems arising from anisotropic PDE problems on general meshes. We present a parallel implementation of a new coarsening algorithm for symmetric positive definite matrices, which is based on a weighted matching approach. We discuss preliminary results obtained by combining this coarsening strategy with the AMG components available in MLD2P4, on linear systems arising from applications considered in the Horizon 2020 Project "Energy oriented Centre of Excellence for computing applications" (EoCoE).

Keywords: AMG · Parallel aggregation · Weighted matching

1 Introduction: AMG Based on Compatible Weighted Matching

Algebraic MultiGrid (AMG) methods have proved to be very promising in preconditioning general sparse linear systems of equations when no information on the origin of the problem is available. Their goal is to define automatic coarsening processes depending only on the coefficient matrix, without using any a priori information or characterization of the *algebraically smooth error*, that is, the error not reduced by the relaxation method. Recent theoretical developments provide general approaches to the construction of coarse spaces for AMG that has optimal convergence, i.e., convergence independent of the problem size, in the case of general linear systems [14,22]. However, despite these theoretical developments, almost all currently available AMG methods and software

I. Lirkov and S. Margenov (Eds.): LSSC 2017, LNCS 10665, pp. 563–571, 2018.
https://doi.org/10.1007/978-3-319-73441-5_62

rely on heuristics to drive the coarsening process among variables; for example the *strength of connection* heuristics is derived from a characterization of the algebraically smooth vectors that is theoretically well understood only for M-matrices.

A new approach for coarsening sparse symmetric positive definite (s.p.d.) matrices has been proposed in recent papers [10,11], following some theoretical and algorithmic developments [5,14]: *coarsening based on compatible weighted matching*. It defines a pairwise aggregation of unknowns where each pair is the result of a maximum weight matching in the matrix adjacency graph. Specifically, the aggregation scheme uses a maximum product matching in order to enhance the diagonal dominance of a matrix representing the hierarchical complement of the resulting coarse matrix, thereby improving the convergence properties of a corresponding compatible relaxation scheme. The matched nodes are aggregated to form coarse index spaces, and piecewise constant or smoothed interpolation operators are applied for the construction of a multigrid hierarchy. No reference is made to any a priori knowledge on the matrix origin and/or any definition of strength of connection; however, information about the smooth error may be used to define edge weights assigned to the original matrix graph. More aggressive coarsening can be obtained by combining multiple steps of the pairwise aggregation.

This method has been implemented in a C-language software framework called *BootCMatch: Bootstrap AMG based on Compatible Weighted Matching*, which incorporates the coarsening method in an adaptive algorithm that computes a composite AMG for general s.p.d. matrices with a prescribed convergence rate through a bootstrap process [10,12]. The main computational kernel of this coarsening approach is the computation of a maximum product matching, accounting for the largest fraction of the time needed to build the AMG preconditioner.

Let $G = (V, E, C)$ be a weighted undirected graph associated with a symmetric matrix A, where V is the set of vertices, E the set of edges and $C = (c_{ij})_{i,j=1,\ldots,n}$ is the real positive matrix of edge weights. A *matching* in G is a subset of edges $M \subseteq E$ such that no two edges share a vertex. The number of edges in M is the cardinality of the matching; a *maximum cardinality matching* contains the largest possible number of edges. A *maximum product weighted matching* is a matching maximizing the product of the weights of the edges in the matching. By a simple weight transformation it is possible to formulate the computation of a maximum product weighted matching in terms of a general maximum weight matching, that is, a matching which maximizes the sum of the weights for each edge in the matching; this is also called an *assignment problem*. Such problems are widely studied in combinatorial optimization and are often used in sparse direct linear solvers for reordering and scaling of matrices [13,17].

Three algorithms for maximum product weighted matching are used in BootCMatch:

MC64: the algorithm implemented in the MC64 routine of the HSL library (http://www.hsl.rl.ac.uk). It works on bipartite graphs, i.e. graphs where the vertex set is partitioned into two subsets V_r and V_c (for example, the rows and columns indices of A) and $(i, j) \in E$ connects $i \in V_r$ and $j \in V_c$; it finds optimal matchings with a worst-case computational complexity $\mathcal{O}(n(n+nnz)\log n)$, where n is the matrix dimension and nnz the number of nonzero entries.

Half-approximate: the algorithm described in [19]. It is a greedy algorithm capable of finding, with complexity $\mathcal{O}(nnz)$, a matching whose total weight is at least half the optimal weight and whose cardinality is at least half the maximum cardinality.

Auction-type: a version of the near-optimal auction algorithm, based on a notion of approximate optimality called ε-complementary slackness, first proposed by Bertsekas [3] and implemented in the SPRAL Library as described in [17]. The original auction algorithm has a worst case computational complexity $\mathcal{O}(n \cdot nnz \log(cn))$, in the case of integer weights, where $c = \max_{ij} |c_{ij}|$; however, its SPRAL version reduces the cost per iteration and the average number of iterations, producing a near-optimal matching at a much lower cost than that of the (optimal) MC64.

Here we discuss a parallel version of the coarsening algorithm described above and its integration into the library of parallel AMG preconditioners MLD2P4, to obtain preconditioners that are robust and efficient on large and sparse linear systems arising from anisotropic elliptic PDE problems discretized on general meshes. The algorithm is implemented with a decoupled approach, where each parallel process performs matching-based coarsening on the part of the matrix owned by the process itself. This requires an extension of the MLD2P4 software framework, in order to efficiently interface the BootCMatch functions implementing the coarsening and to combine the new functionality with the other AMG components of MLD2P4. Computational experiments have been performed on linear systems coming from applications in the Horizon 2020 Project "Energy oriented Centre of Excellence for computing applications" (EoCoE, http://www.eocoe.eu).

2 Interfacing MLD2P4 with BootCMatch

MLD2P4 is a package of parallel AMG and domain decomposition preconditioners, designed to provide scalable and easy-to-use preconditioners [6–9], which has been successfully exploited in different applications (see, e.g., [2,4]). It is based on the PSBLAS computational framework [16] and can be used with the PSBLAS Krylov solvers. MLD2P4 is written in Fortran 2003 using an object-oriented approach [15]; thanks to its layered software architecture, its classes and functionalities can be easily extended and reused. It is freely available from https://github.com/sfilippone/mld2p4-2.

Two classes are used to represent AMG preconditioners in MLD2P4: the *base preconditioner* and the *multilevel preconditioner*. The multilevel preconditioner

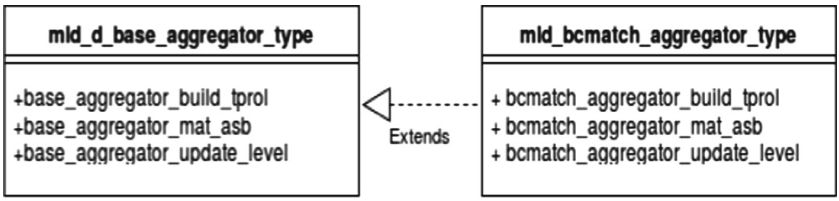

Fig. 1. Extending the aggregator class in MLD2P4 (only the relevant methods are displayed)

holds the coarse matrices of the AMG hierarchy, the corresponding smoothers (stored as base preconditioners), and prolongation and restriction operators. An *aggregator* object, encapsulated in the base preconditioner structure, hosts the aggregation method for building the AMG hierarchy. The basic aggregation available in MLD2P4 is the decoupled smoothed aggregation algorithm presented in [20], where the final prolongator is obtained by applying a suitable smoother to a tentative prolongator. In particular, the method *base_aggregator_build_tprol* builds the tentative prolongator and stores other information about it and the corresponding restriction operator.

The MLD2P4 *basic_aggregator_type* class can be extended to define new aggregation strategies, possibly adding attributes and methods, or overriding the basic ones. In order to interface the aggregation algorithm implemented in BootCMatch, the class *bcmatch_aggregator_type* was defined, extending *basic_aggregator_type* and providing the *bcmatch_aggregator_build_tprol* method, which ovverrides the basic method and performs matching-based aggregation; a schematic picture of the new class is given in Fig. 1.

When implementing the interface, we had to take into account that MLD2P4 is written in Fortran while BootCMatch is written in C, and MLD2P4 and BootCMatch use different data structures for storing sparse matrices and vectors. Fortran 2003 provides a standardized way for creating procedures, variables and types that are interoperable with C. We exploited this feature by creating in MLD2P4 two derived types that are interoperable with C and correspond to the *bcm_CSRMatrix* and *bcm_vector* structures used in BootCMatch.

Furthermore, BootCMatch is sequential and thus needed a strategy to be applied in the parallel MLD2P4 environment. Therefore, BootCMatch was interfaced with the decoupled parallel aggregation scheme available in MLD2P4. This was implemented by passing to the aggregator the local part of the matrix held by the current process, so that the aggregation algorithm could be run on that submatrix, obtaining a local tentative prolongator. Furthermore, a subroutine called *mld_base_map_to_tprol* was developed to create a global tentative prolongator from the local ones. Of course, the resulting parallel matching-based aggregation is generally different from the sequential one.

The *base_aggregator_mat_asb* method performs several actions: it applies a smoother to the tentative prolongator (if required), computes the restriction operator as $R = P^T$, and creates the coarse-level matrix A_c using the Galerking

approach, i.e. $A_c = RAP$. This method was overridden to take into account that the BootCMatch tentative prolongator has a more general form than the basic tentative prolongator implemented in MLD2P4. The *base_aggregator_update_level* method was also overriden by *bcmatch_aggregator_ update_level*, which takes care of projecting on the next coarse level information needed by the matching-based aggregation.

3 Computational Experiments

In order to illustrate the behaviour of the parallel AMG preconditioners described in the previous sections, we show the results obtained on linear systems derived from two applications in the framework of the Horizon 2020 EoCoE Project.

A first set of linear systems comes from a groundwater modelling application developed at the Jülich Supercomputing Centre (JSC); it deals with the numerical simulation of the filtration of 3D incompressible single-phase flows through anisotropic porous media. The linear systems arise from the discretization of an elliptic equation with no-flow boundary conditions, modelling the pressure field, which is obtained by combining the continuity equation with Darcy's law. The discretization is performed by a cell-centered finite volume scheme (two-point flux approximation) on a Cartesian grid [1]. The systems considered in this work have s.p.d. matrices with dimension 10^6 and 6940000 nonzero entries distributed over seven diagonals. The anisotropic permeability tensor in the elliptic equation is randomly generated from a lognormal distribution with mean 1 and three standard deviation values, i.e., 1, 2 and 3, corresponding to the three systems M1-Filt, M2-Filt and M3-Filt. The systems were generated by using a Matlab code implementing the basics of reservoir simulations and can be regarded as simplified samples of systems arising in ParFlow, an integrated parallel watershed computational model for simulating surface and subsurface fluid flow, used at JSC.

A second set of linear systems comes from computational fluid dynamics simulations for wind farm design and management, carried out at the Barcelona Supercomputing Center (BSC) by using Alya, a multi-physics simulation code targeted at HPC applications [21]. The systems arise in the numerical solution of Reynolds-Averaged Navier-Stokes equations coupled with a modified $k - \varepsilon$ model; the space discretization is obtained by using stabilized finite elements, while the time integration is performed by combining a backward Euler scheme with a fractional step method, which splits the computation of the velocity and pressure fields and thus requires the solution of two linear systems at each time step. Here we show the results concerning two systems associated with the pressure field, which are representative of systems arising in simulations with discretization meshes of different sizes. The systems, denoted by M1-Alya and M2-Alya, have s.p.d. matrices of dimensions 790856 and 2224476, with 20905216 and 58897774 nonzeros, respectively, corresponding to up to 27 entries per row.

All the systems were preconditioned by using an AMG Kcycle [18] with decoupled unsmoothed double-pairwise matching-based aggregation. One forward/backward Gauss-Seidel sweep was used as pre/post-smoother with systems M1-Filt, M2-Filt and M3-Filt; one block-Jacobi sweep, with ILU(0) factorization of the blocks, was applied as pre/post-smoother with M1-Alya and M2-Alya. UMFPACK (http://faculty.cse.tamu.edu/davis/suitesparse.html) was used to solve the coarsest-level systems, through the interface provided by MLD2P4. The experiments were performed by using the three matching algorithms described in Sect. 1. The truncated Flexible Conjugate Gradient method FCG(1) [18], implemented in PSBLAS, was chosen to solve all the systems, according to the variability introduced in the preconditioner by the Kcycle. The zero vector was chosen as starting guess and the iterations were stopped when the 2-norm of the residual achieved a reduction by a factor of 10^{-6}. A generalized row-block distribution of the matrices was used, computed by using the Metis graph partitioner (http://glaros.dtc.umn.edu/gkhome/metis/metis/overview).

The experiments were carried out on the yoda linux cluster, operated by the Naples Branch of the CNR Institute for High-Performance Computing and Networking. Its compute nodes consist of 2 Intel Sandy Bridge E5-2670 8-core processors and 192 GB of RAM, connected via Infiniband. Given the size and the sparsity of the linear systems, at most 64 cores, running as many parallel processes, were used; 4 cores per node were considered, according to the memory bandwidth requirements of the linear systems. PSBLAS 3.4 and MLD2P4 2.2, installed on the top of MVAPICH 2.2, were used together with a development version of BootCMatch and the version of UMFPACK available in SuiteSparse 4.5.3. The codes were compiled with the GNU 4.9.1 compiler suite.

We first discuss the results for the systems M1-Filt, M2-Filt and M3-Filt. In Table 1 we report the number of iterations performed by FCG with the preconditioners based on the different matching algorithms, as the number of processes, np, varies. In Table 2 we report the corresponding execution times (in seconds) and speedup values. The times include the construction of the preconditioner and the solution of the preconditioned linear system. In general the preconditioned FCG solver shows reasonable algorithmic scalability, i.e. the number of iterations

Table 1. M1-Filt, M2-Filt and M3-Filt: number of iterations

	M1-Filt			M2-Filt			M3-Filt		
np	MC64	Half-app	Auction	MC64	Half-app	Auction	MC64	Half-app	Auction
1	13	11	12	18	29	18	46	58	52
2	15	11	14	20	35	21	79	68	65
4	15	11	13	20	32	21	62	83	64
8	15	11	13	20	31	22	69	77	71
16	13	11	15	19	29	19	75	69	101
32	15	11	15	21	37	21	79	68	82
64	15	11	13	26	32	21	86	76	78

Table 2. M1-Filt, M2-Filt and M3-Filt: execution time and speedup

	M1-Filt						M2-Filt						M3-Filt					
	MC64		Half-app		Auction		MC64		Half-app		Auction		MC64		Half-app		Auction	
np	time	sp	time	sp	time	sp	time	sp	time	sp	time	sp	time	sp	time	sp	time	sp
1	18.58	1.0	8.72	1.0	8.90	1.0	19.54	1.0	12.46	1.0	9.73	1.0	25.15	1.0	18.11	1.0	15.61	1.0
2	9.67	1.9	4.43	2.0	4.71	1.9	11.16	1.8	6.61	1.9	5.58	1.7	16.24	1.5	10.07	1.8	9.48	1.6
4	5.30	3.5	2.68	3.2	2.75	3.2	5.55	3.5	4.05	3.1	3.14	3.1	8.29	3.0	7.17	2.5	5.63	2.8
8	2.66	7.0	1.42	6.1	1.32	6.7	2.79	7.0	2.02	6.2	1.63	6.0	4.41	5.7	3.66	5.0	3.24	4.8
16	1.05	17.7	0.71	12.2	0.73	12.2	1.18	16.6	1.07	11.7	0.77	12.6	1.88	13.4	1.43	12.6	2.05	7.6
32	0.67	27.9	0.52	16.8	0.52	17.2	0.75	26.2	0.82	15.3	0.60	16.1	1.25	20.2	1.16	15.7	1.07	14.6
64	0.43	43.0	0.43	20.4	0.39	22.9	0.51	38.5	0.60	20.7	0.40	24.1	0.82	30.6	0.82	22.2	0.83	18.7

for all systems does not vary too much with the number of processes. A larger variability in the iterations can be observed with M3-Filt, due to the higher anisotropy of this problem and its interaction with the decoupled aggregation strategy. With a more in-depth analysis (not shown here for the sake of space) we find the coarsening ratio between consecutive AMG levels ranges between 3.6 and 3.8 for the MC64 and auction algorithms, while it is between 3.0 and 3.3 for the half-approximation one, except with 64 processes where it reduces to 2.8 for M2-Filt and M3-Filt. None of the three matching algorithms produces preconditioners that are clearly superior in reducing the number of FCG iterations; indeed, for these systems there is no advantage in using the optimal matching algorithm implemented in MC64, since the non-optimal ones appear very competitive. The times corresponding to the half-approximation and auction algorithms are generally smaller, mainly because the time needed to build the AMG hierarchies is reduced. The speedup decreases as the anisotropy of the problem grows due to the larger number of FCG iterations and the well-known memory-bound nature of our computations. The largest speedups are obtained with MC64, because of the larger time required by MC64 on a single core.

In the case of M1-Alya and M2-Alya, only MC64 is able to produce an effective coarsening (with a ratio greater than 3.8), hence avoiding a significant increase of the number of iterations when the number of processes grows. Therefore, in Table 3, we only report results (iterations, time and speedup) for this case. The preconditioned solver shows good algorithmic scalability in general; the number of iterations on a single core is much smaller because in this case the AMG smoother reduces to an ILU factorization. A speedup of 42.8 is achieved for M1-Alya, which reduces to 29.3 for the larger matrix M2-Alya; in our opinion, this can be considered satisfactory. Note that we did not achieve convergence to the desired accuracy by using the basic classical smoothed aggregation algorithm available in MLD2P4.

In conclusion, the results discussed here show the potential of parallel matching-based aggregation and provide an encouraging basis for further work in this direction, including the application of non-decoupled parallel matching algorithms and the design of ad-hoc coarsening strategies for some classes of smoothers, according to the principles of compatible relaxation [14,22].

Table 3. M1-Alya and M2-Alya: number of iterations, execution time and speedup, using weighted matching based on MC64.

	M1-Alya			M2-Alya		
np	it	time	sp	it	time	sp
1	8	51.22	1.0	8	76.00	1.0
2	39	25.39	2.0	40	81.90	0.9
4	40	11.69	4.4	44	39.26	1.9
8	50	5.96	8.6	43	19.24	3.9
16	57	2.89	17.7	48	10.84	7.0
32	84	2.18	23.5	52	5.25	14.5
64	58	1.20	42.8	48	2.60	29.2

Acknowledgments. This work has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No. 676629 (Project EoCoE). The authors wish to thank Herbert Howen (BSC, Barcelona), Stefan Kollet and Wendy Sharples (JSC, Jülich) for making available the test problems.

References

1. Aarnes, J.E., Gimse, T., Lie, K.-A.: An introduction to the numerics of flow in porous media using matlab. In: Hasle, G., Lie, K.-A., Quak, E. (eds.) Geometric Modelling, Numerical Simulation, and Optimization, pp. 265–306. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-68783-2_9
2. Aprovitola, A., D'Ambra, P., Denaro, F.M., di Serafino, D., Filippone, S.: SParC-LES: enabling large eddy simulations with parallel sparse matrix computation tools. Comput. Math. Appl. **70**, 2688–2700 (2015)
3. Bertsekas, D.P.: The auction algorithm: a distributed relaxation method for the assignment problem. Ann. Oper. Res. **14**, 105–123 (1988)
4. Borzì, A., De Simone, V., di Serafino, D.: Parallel algebraic multilevel Schwarz preconditioners for a class of elliptic PDE systems. Comput. Vis. Sci. **16**, 1–14 (2013)
5. Brannick, J., Chen, Y., Kraus, J., Zikatanov, L.: Algebraic multilevel preconditioners for the graph Laplacian based on matching in graphs. SIAM J. Numer. Anal. **51**, 1805–1827 (2013)
6. Buttari, A., D'Ambra, P., di Serafino, D., Filippone, S.: Extending PSBLAS to build parallel Schwarz preconditioners. In: Dongarra, J., Madsen, K., Waśniewski, J. (eds.) PARA 2004. LNCS, vol. 3732, pp. 593–602. Springer, Heidelberg (2006). https://doi.org/10.1007/11558958_71
7. Buttari, A., D'Ambra, P., di Serafino, D., Filippone, S.: 2LEV-D2P4: a package of high-performance preconditioners for scientific and engineering applications. Appl. Algebra Eng. Commun. Comput. **18**, 223–239 (2007)
8. D'Ambra, P., di Serafino, D., Filippone, S.: On the development of PSBLAS-based parallel two-level Schwarz preconditioners. Appl. Numer. Math. **57**, 1181–1196 (2007)

9. D'Ambra, P., di Serafino, D., Filippone S.: MLD2P4: a package of parallel algebraic multilevel domain decomposition preconditioners in Fortran 95. ACM Trans. Math. Softw. **37**(3), Article No. 30 (2010). https://github.com/sfilippone/mld2p4-2
10. D'Ambra, P., Vassilevski, P.S.: Adaptive AMG with coarsening based on compatible weighted matching. Comput. Vis. Sci. **16**, 59–76 (2013)
11. D'Ambra, P., Vassilevski, P.S.: αAMG based on weighted matching for systems of elliptic PDEs arising from displacement and mixed methods. In: Russo, G., Capasso, V., Nicosia, G., Romano, V. (eds.) ECMI 2014. MI, vol. 22, pp. 1013–1020. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-23413-7_142
12. D'Ambra, P., Filippone, S., Vassilevski, P.S.: BootCMatch: a software package for bootstrap AMG based on graph weighted matching. Submitted
13. Duff, I.S., Koster, J.: On algorithms for permuting large entries to the diagonal of a sparse matrix. SIAM J. Matrix Anal. Appl. **22**, 973–996 (2001)
14. Falgout, R.D., Vassilevski, P.S.: On generalizing the algebraic multigrid framework. SIAM J. Numer. Anal. **42**, 1669–1693 (2004)
15. Filippone, S., Buttari, A.: Object-oriented techniques for sparse matrix computations in Fortran 2003. ACM Trans. Math. Softw. **38**(4), Article No. 23 (2012)
16. Filippone, S., Colajanni, M.: PSBLAS: a library for parallel linear algebra computation on sparse matrices. ACM Trans. Math. Softw. **26**, 527–550 (2000)
17. Hogg, J., Scott, J.: On the use of suboptimal matchings for scaling and ordering sparse symmetric matrices. Numer. Linear Algebra Appl. **22**, 648–663 (2015)
18. Notay, Y., Vassilevski, P.S.: Recursive Krylov-based multigrid cycles. Numer. Linear Algebra Appl. **15**, 473–487 (2008)
19. Preis, R.: Linear time 1/2-approximation algorithm for maximum weighted matching in general graphs. In: Meinel, C., Tison, S. (eds.) STACS 1999. LNCS, vol. 1563, pp. 259–269. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-49116-3_24
20. Vaněk, P., Mandel, J., Brezina, M.: Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems. Computing **56**, 179–196 (1996)
21. Vásquez, M., Houzeaux, G., Koric, S., Artigues, A., Aguado-Sierra, J., Arís, R., Mira, D., Calmet, H., Cucchietti, F., Owen, H., Taha, A., Burness, E.D., Cela, J.M., Valero, M.: Alya: multiphysics engineering simulation toward exascale. J. Comput. Sci. **14**, 15–27 (2016)
22. Xu, J., Zikatanov, L.: Algebraic multigrid methods. Acta Numerica **26**, 591–721 (2017). https://doi.org/10.1017/S0962492917000083