Proceedings of the Seventh International Conference on
Parallel, Distributed, GPU and Cloud Computing for Engineering
Edited by: P. Iványi, F. Magoulès and B.H.V. Topping
Civil-Comp Conferences, Volume 4, Paper 1.1
Civil-Comp Press, Edinburgh, United Kingdom, 2023
doi: 10.4203/ccc.4.1.1

Accurate Coarse Residual for Two-Level Asynchronous Domain Decomposition Methods

G. Gbikpi-Benissan[1] and F. Magoulès[1,2]

[1]**Laboratory of Mathematics in Interaction with Computer Science, CentraleSupélec, Paris-Saclay University, Gif-sur-Yvette, France**

[2]**Faculty of Engineering and Information Technology University of Pécs, Hungary**

Abstract

Recently, asynchronous coarse-space correction has been achieved within both the overlapping Schwarz and the primal Schur frameworks. Both additive and multiplicative corrections have been discussed. In this paper, we address some implementation drawbacks of the proposed additive correction scheme. In the existing approach, each coarse solution is applied only once, leaving most of the iterations of the solver without coarse-space information while building the right-hand side of the coarse problem. Moreover, one-sided routines of the Message Passing Interface (MPI) standard were considered, which introduced the need for a sleep statement in the iterations loop of the coarse solver. This implies a tuning of the sleep period, which is a non-discrete quantity. In this paper, we improve the accuracy of the coarse right-hand side, which allowed for more frequent corrections. In addition, we highlight a two-sided implementation which better suits the asynchronous coarse-space correction scheme. Numerical experiments show a significant performance gain with such increased incorporation of the coarse space.

Keywords: parallel computing, domain decomposition methods, overlapping Schwarz, asynchronous iterations, coarse-space correction, global residual

1 Introduction

As the amount of parallelism in modern computing platforms is tremendously increasing, numerical iterative methods have to cope with resource failures, heterogeneity and costly communication. Domain decomposition methods [1, 2] are so far among the approaches providing highest levels of parallelism. Still, their classical mathematical design relies on iterative schemes which hardly support failures and heterogeneity. Asynchronous iterations [3] early arose as a parallel mathematical scheme where computation and communication can occur concurrently, and are now gaining a particular attention as a viable option for overcoming the limits of classical iterative methods. For an overview of the asynchronous iterations theory and applications, we refer to, e.g., the review papers [4, 5]. Recent results have shown promising performances against Krylov methods within a framework of non-overlapping domain decomposition (see, e.g., [6, 7]). These results were obtained while applying asynchronous coarse-space correction [8, 9], a recent breakthrough in the asynchronous computing field.

In [8], an asynchronous additive coarse-space correction was proposed in the framework of the restricted additive Schwarz (RAS) method, however, an in-depth analysis by [9] in case of multiplicative correction showed that more attention should be paid to the accuracy of the computed coarse residual, which determines the effectiveness of the correction. A connection can therefore be made between the asynchronous coarse-space correction issue and the asynchronous convergence detection one, both linked by the fact that it seems not trivial to compute a consistent global residual without pausing the ongoing computation. The convergence detection issue has been largely discussed in the literature of asynchronous methods (see, e.g., [10–14]), however, most of the proposed solutions rely on local residuals which, in the context of asynchronous iterations, are not local components of a global residual. In [8, 15], an approximation approach was used in a centralized form, which only needs a reduction operation over the local residuals (also see, e.g., Fig. 3 or Fig. 4 in [14] or Algorithm 4.5 in [9] for the distributed form). A similar operation was performed to also construct the coarse global residual vector. While such a minimal approach can be robust enough for convergence detection, it seems natural that the accuracy of a global residual is much more critical for effective asynchronous coarse-space correction. In this paper, we apply more accurate global residual ideas from [13, 16] (also see Section 3 of [10]) to improve the quality of the coarse-space correction.

The paper is organized as follows. Section 2 recalls the iterative model of an RAS solver, its asynchronous implementation, and their current two-level counterparts with additive coarse-space correction. Our contribution is in Section 3. We first present a simpler way to manage communications between the subdomains and the coarse domain, then we propose the use of a globally consistent coarse residual to improve the accuracy of the coarse solution. Experimental investigation is conducted in Section 4, and our conclusions follow in Section 5.

2 Computational background

We consider iterative solution of a linear algebraic problem $Ax = b$ with $A \in \mathbb{R}^{n\times n}$, where A is a large sparse real coefficients matrix and x is a real unknowns vector. Consider $p \in \mathbb{N}$ discrete subdomains, and let R_i, $i \in \{1, \ldots, p\}$, denote the unknowns selection mapping to form the i-th subdomain. Derive the local matrices and vectors $A_i = R_i A R_i^\mathsf{T}$, $x_i = R_i x$ and $b_i = R_i b$, where the superscript T denotes the transpose of a matrix or vector. Consider diagonal Boolean matrices, say B_i, $i \in \{1, \ldots, p\}$, such that $\sum_{i=1}^{p} R_i^\mathsf{T} B_i R_i = I$, where I denotes the identity matrix. The RAS preconditioner [17] is then given by $M = \sum_{i=1}^{p} R_i^\mathsf{T} B_i A_i^{-1} R_i$, which leads to the RAS iterative solver,

$$x^{k+1} = x^k + M\left(b - Ax^k\right), \qquad k \in \mathbb{N}, \tag{1}$$

and its parallel form,

$$x_i^{k+1} = \sum_{j=1}^{p} R_i R_j^\mathsf{T} B_j \left(x_j^k + A_j^{-1}\left(b_j - R_j A x^k\right)\right).$$

Note that $R_j A$, $j \in \{1, \ldots, p\}$, is of the form $R_j A = (A_j, R_j A R_{\neg j}^\mathsf{T})$, where $\neg j$ denotes the complement of the j-th subdomain, which means that $R_j A R_{\neg j}^\mathsf{T}$ corresponds to the coefficients connecting the unknowns of the j-th subdomain to the ones not included in the subdomain. They can be thought of as boundary unknowns. It yields the final parallel scheme

$$x_i^{k+1} = \sum_{j=1}^{p} R_i R_j^\mathsf{T} B_j A_j^{-1}\left(b_j - R_j A R_{\neg j}^\mathsf{T} x_{\neg j}^k\right), \quad i \in \{1, \ldots, p\}, \quad k \in \mathbb{N}. \tag{2}$$

With such a parallel scheme, communication is needed for boundary components $x_{\neg i}^k$ only.

Asynchronous iterations [3] consist of not waiting for sending and receiving data before resuming to the next iteration. Theoretical modeling and convergence analysis of asynchronous iterations have been largely discussed in the literature. We refer to, e.g., the review papers [4, 5] and the RAS-related papers [18, 19]. Let N_i denote the subset of the subdomains with which the i-th subdomain shares boundary unknowns. Using the well-known Message Passing Interface (MPI) formalism, an asynchronous implementation of the RAS iterations (2) is described in Algorithm 1. The *ISynchronize* function implements necessary communication between the i-th subdomain and its neighboring subdomains, in a non-blocking way. For simplicity, we consider marking the returned requests for being automatically freed upon completion (function *FreeAll*). Efficient management of MPI communication requests was discussed in, e.g., [16, 20]. In case of an underlying network protocol supporting remote direct memory access (RDMA), one-sided MPI communication should be considered (see, e.g., [21]).

We rather focus on the simple approach analyzed in [14], where convergence can be reliably detected by only considering the non-blocking collective function *IAllReduce*. It computes an approximation of the dot product $r^\mathsf{T} r$, based on inconsistent

Algorithm 1 Asynchronous RAS solver.

Require: A_i, $R_i A R_{\neg i}^{\mathsf{T}}$, b_i, x_i, $x_{\neg i}$, B_i, N_i, ε, $\|b\|$, $k_{\max}$
1: $r_i := b_i - A_i x_i - R_i A R_{\neg i}^{\mathsf{T}} x_{\neg i}$; AllReduce($r_i^{\mathsf{T}} B_i r_i$, $r^{\mathsf{T}} r$, SUM) ; $\|r\| := \sqrt{r^{\mathsf{T}} r}$; $k := 0$
2: req_r := REQUEST_NULL
3: **while** $\|r\| \geq \varepsilon \|b\|$ **and** $k < k_{\max}$ **do**
4: $x_i := x_i$ + Solve(A_i, r_i) ; $\text{reqs}_x[\,]$:= ISynchronize(x_i, $x_{\neg i}$, N_i) ; FreeAll(reqs_x)
5: $r_i := b_i - A_i x_i - R_i A R_{\neg i}^{\mathsf{T}} x_{\neg i}$
6: **if** Test(req_r) **then**
7: $\|r\| := \sqrt{r^{\mathsf{T}} r}$; req_r := IAllReduce($r_i^{\mathsf{T}} B_i r_i$, $r^{\mathsf{T}} r$, SUM) ; $k := k + 1$
8: **end if**
9: **end while**
10: **return** x_i

local contributions $r_i^{\mathsf{T}} B_i r_i$. Upon completion of the collective operation (checked with *Test(req$_r$)*), all the processes consider a same approximation of the residual norm $\|r\| = \sqrt{r^{\mathsf{T}} r}$. While such an approximated residual can be satisfactory in practice for stopping an asynchronous iterative solver, we shall see that it prevents the solver from taking full advantage of coarse-space correction.

In [8], an asynchronous implementation of a two-level RAS solver was proposed. The considered classical coarse-space technique consists of defining a restriction mapping, say R_0, which aggregates the unknowns of each subdomain into one coarse unknown, and is used to define a coarse matrix, say $A_0 = R_0 A R_0^{\mathsf{T}}$. The iterative method (1) is then modified such as

$$x^{k+1} = x^k + \frac{1}{2}\left(M + R_0^{\mathsf{T}} A_0^{-1} R_0\right)\left(b - A x^k\right), \qquad k \in \mathbb{N},$$

which results in the two-level parallel scheme

$$\left\{ \begin{array}{rcl} x_0^k & = & A_0^{-1} \displaystyle\sum_{j=1}^{p} R_0 R_j^{\mathsf{T}} B_j \left(b_j - A_j x_j^k - R_j A R_{\neg j}^{\mathsf{T}} x_{\neg j}^k\right), \\ x_i^{k+1} & = & \displaystyle\sum_{j=1}^{p} \frac{1}{2} R_i R_j^{\mathsf{T}} B_j \left(A_j^{-1}\left(b_j - R_j A R_{\neg j}^{\mathsf{T}} x_{\neg j}^k\right) + x_j^k + R_j R_0^{\mathsf{T}} x_0^k\right), \end{array} \right. \tag{3}$$

with $i \in \{1, \ldots, p\}$ and $k \in \mathbb{N}$. A comprehensive introduction to multilevel techniques can be found, e.g., in the review books [1, 2, 22].

The asynchronous implementation proposed in [8] is based on the fact that the coarse domain is simply seen as a $(p+1)$-th subdomain handling the component x_0. The asynchronous implementation therefore consists of not waiting for an updated x_0 before updating x_i. However, the coarse solver waits for receiving each of the contributions $R_0 R_i^{\mathsf{T}} B_i r_i$ before computing x_0. Similarly, the update of x_i is corrected with x_0 only when an updated x_0 is received. It follows that a coarse solution x_0 can be rarely available, and when it is finally provided, it is applied only once. We propose here a slight modification of this implementation approach, which allowed us to more often correct x_i, whether x_0 has been updated or not.

3 New computational model

3.1 Two-level asynchronous RAS solver with two-sided MPI

The two-level asynchronous RAS solver proposed in [8,15] is based on MPI one-sided communication routines, however, coarse-space correction requires checking arrival of new data. As a drawback, the coarse domain process has to constantly poll shared Boolean variables in an active loop, waiting for subdomain contributions to the coarse residual vector. A sleep statement was therefore introduced in the loop, hence, one has to determine the optimal sleep period. This adds to the other problem of using each coarse solution only once.

Two-sided routines are particularly suitable for such data arrival checks. If a $(p+1)$-th process is dedicated to the coarse domain, a function *Wait* can be simply called on requests initialized by *IRecv* or *IGather*. If, otherwise, a i_0-th process, $i_0 \in \{1, \ldots, p\}$, handles the coarse domain, then completion is just checked with a function *Test*. The latter case is described here in Algorithm 2. As one can see, compared to all the

Algorithm 2 Two-level asynchronous RAS with non-blocking collective routines.

Require: A_i, $R_i A R_{\neg i}^\mathsf{T}$, b_i, x_i, $x_{\neg i}$, B_i, N_i, A_0, $R_0 R_i^\mathsf{T}$, i_0, ε, $\|b\|$, $k_{\max}$
1: $r_i := b_i - A_i x_i - R_i A R_{\neg i}^\mathsf{T} x_{\neg i}$; AllReduce($r_i^\mathsf{T} B_i r_i$, $r^\mathsf{T} r$, SUM) ; $\|r\| := \sqrt{r^\mathsf{T} r}$; $k := 0$
2: req_r := REQUEST_NULL ; $\text{state}_0 := 0$
3: **while** $\|r\| \geq \varepsilon \|b\|$ **and** $k < k_{\max}$ **do**
4: **if** $\text{state}_0 = 0$ **then** req_{r_0} := IReduce($R_0 R_i^\mathsf{T} B_i r_i$, r_0, i_0, SUM) ; $\text{state}_0 := 1$ **end if**
5: **if** $\text{state}_0 = 1$ **and** Test(req_{r_0}) **then**
6: **if** $i = i_0$ **then** x_0 := Solve(A_0, r_0) **end if**
7: req_{x_0} := IBcast(x_0, i_0) ; $\text{state}_0 := 2$
8: **end if**
9: **if** $\text{state}_0 = 2$ **and** Test(req_{x_0}) **then** $x_i := x_i + 0.5 \times (\text{Solve}(A_i, r_i) + R_i R_0^\mathsf{T} x_0)$; $\text{state}_0 := 0$
10: **else** $x_i := x_i + \text{Solve}(A_i, r_i)$ **end if**
11: $\text{reqs}_x[\,] :=$ ISynchronize(x_i, $x_{\neg i}$, N_i) ; FreeAll(reqs_x) ; $r_i := b_i - A_i x_i - R_i A R_{\neg i}^\mathsf{T} x_{\neg i}$
12: **if** Test(req_r) **then** $\|r\| := \sqrt{r^\mathsf{T} r}$; req_r := IAllReduce($r_i^\mathsf{T} B_i r_i$, $r^\mathsf{T} r$, SUM) ; $k := k + 1$ **end if**
13: **end while**
14: **return** x_i

shared Boolean variables managed in the one-sided implementation (at both subdomain and coarse domain sides), non-blocking collective routines allow for a more straightforward implementation without any particular performance issue. In practice, the function *IReduce* (applied to the contributions $R_0 R_i^\mathsf{T} B_i r_i$, $i \in \{1, \ldots, p\}$) can actually consist of an *IGather* where the subdomains send only data related to their coarse unknown. It must be noted that, if an RDMA-capable network device is targeted, a hybrid approach is advisable, which consists of using one-sided routines for inter-subdomains communication and non-blocking collective routines for coarse-space correction.

Beyond the type of communication routines one chooses to consider, the main performance issue of such a two-level asynchronous solver is the weak incorporation of the coarse space. As suggested by [8,15], each coarse solution is applied only once,

however, if the coarse domain can just be seen as a $(p+1)$-th subdomain, then the corrections should just be done using the latest available coarse solution x_0, without wondering if it has been updated or not. In the sequel, we propose a more effective coarse-space correction where x_0 is more often incorporated.

3.2 Accurate two-level asynchronous RAS solver

The two-level asynchronous RAS solver [8] features a one-sided non-blocking synchronization which consists of collecting all the residual contributions, $R_0 R_i^\mathsf{T} B_i r_i$, $i \in \{1, \ldots, p\}$, before computing the coarse residual, r_0. Algorithm 2 has provided a two-sided counterpart by using non-blocking collective routines. It however appears in both implementations that the computed coarse residual, r_0, is inconsistent since each fine local component r_i is based on globally inconsistent interface data $x_{\neg i}$. The effectiveness of the coarse-space correction can therefore be improved by considering a consistent coarse residual, which is achieved here by using consistent interface data $x_{\neg i}$.

Let, then, $\bar{x}$ and $\bar{r}$ respectively denote the global solution and the global residual vectors that we shall consistently build. Algorithm 3 describes the improved two-level asynchronous solver. A step is added to Algorithm 2, prior to the coarse residual com-

Algorithm 3 Accurate two-level asynchronous RAS.

Require: A_i, $R_i A R_{\neg i}^\mathsf{T}$, b_i, x_i, $x_{\neg i}$, B_i, N_i, A_0, $R_0 R_i^\mathsf{T}$, i_0, ε, $\|b\|$, $k_{\max}$
1: $r_i := b_i - A_i x_i - R_i A R_{\neg i}^\mathsf{T} x_{\neg i}$; AllReduce($r_i^\mathsf{T} B_i r_i$, $r^\mathsf{T} r$, SUM) ; $\|r\| := \sqrt{r^\mathsf{T} r}$; $k := 0$
2: req_r := REQUEST_NULL ; $\text{state}_0 := 0$; $x_0 := 0$
3: **while** $\|r\| \geq \varepsilon \|b\|$ **and** $k < k_{\max}$ **do**
4: **if** $\text{state}_0 = 0$ **then** $\bar{x}_i := x_i$; $\text{reqs}_{\bar{x}}[\,]$:= ISynchronize($\bar{x}_i$, $\bar{x}_{\neg i}$, N_i) ; $\text{state}_0 := 1$ **end if**
5: **if** $\text{state}_0 = 1$ **and** TestAll($\text{reqs}_{\bar{x}}$) **then**
6: $\bar{r}_i := b_i - A_i \bar{x}_i - R_i A R_{\neg i}^\mathsf{T} \bar{x}_{\neg i}$; req_{r_0} := IReduce($R_0 R_i^\mathsf{T} B_i \bar{r}_i$, r_0, i_0, SUM) ; $\text{state}_0 := 2$
7: **end if**
8: **if** $\text{state}_0 = 2$ **and** Test(req_{r_0}) **then**
9: **if** $i = i_0$ **then** x_0 := Solve(A_0, r_0) **end if**
10: req_{x_0} := IBcast(x_0, i_0) ; $\text{state}_0 := 3$
11: **end if**
12: **if** $\text{state}_0 = 3$ **and** Test(req_{x_0}) **then** $\text{state}_0 := 0$ **end if**
13: $x_i := x_i + 0.5 \times (\text{Solve}(A_i, r_i) + R_i R_0^\mathsf{T} x_0)$
14: $\text{reqs}_x[\,]$:= ISynchronize(x_i, $x_{\neg i}$, N_i) ; FreeAll(reqs_x) ; $r_i := b_i - A_i x_i - R_i A R_{\neg i}^\mathsf{T} x_{\neg i}$
15: **if** Test(req_r) **then** $\|r\| := \sqrt{r^\mathsf{T} r}$; req_r := IAllReduce($r_i^\mathsf{T} B_i r_i$, $r^\mathsf{T} r$, SUM) ; $k := k + 1$ **end if**
16: **end while**
17: **return** x_i

putation. Each subdomain $i \in \{1, \ldots, p\}$ records a version, $\bar{x}_i$, of its local solution x_i, and triggers synchronization with its neighbors on this particular version. The returned requests are then checked at each subsequent iteration. Once they are all completed, a globally consistent local residual, $\bar{r}_i$, can be computed using the recorded local solution, $\bar{x}_i$, and the corresponding synchronized interface solution, $\bar{x}_{\neg i}$. The remaining

non-blocking coarse-space correction procedure is then based on such consistent local residuals.

With such a more consistent coarse solution, x_0, we also now let the coarse-space correction occur at several successive iterations, simply using the latest available x_0. As we will see from numerical experiments, this can result in a significantly more effective two-level asynchronous solver.

4 Experimental results

4.1 Problem and general settings

Similarly to [15], the Poisson's equation, $-\Delta u = g$, is considered for evaluating the practical performance of the proposed asynchronous additive coarse-space correction. The three-dimensional domain $[0, 1]^3$ is considered with a uniform source $g = 4590$ and $u = 0$ on the whole boundary. The domain is decomposed on each dimension, and the local matrices and vectors, A_i, $R_i A R_{\neg i}^{\mathsf{T}}$ and b_i, $i \in \{1, \ldots, p\}$, are generated in each subdomain by P1 finite-element approximation. Both A_i and A_0 are then Cholesky-factorized.

The compute cluster is homogeneously composed of nodes consisting of 175 GB of memory and two 20-cores processors at 2.1 GHz (40 cores per node), connected through an Omni-Path Architecture (OPA) network at 100 Gbit/s. Each core runs exactly one MPI process which also handles exactly one subdomain.

In the following, we show performances of the RAS solver (2) (referred to as "1L-Sync"), its two-sided asynchronous execution from Algorithm 1 ("1L-Async"), the two-level RAS solver (3) ("2L-Sync"), and its two-sided asynchronous execution from Algorithm 3 based on the proposed consistent coarse residual. We first considered applying each coarse solution only once ("1corr-2L-Async"), based on [8,15], then we allowed for a maximum of five corrections using the same coarse solution ("5corr-2L-Async"). Overall execution times are compared for a stopping criterion $\|b - Ax\| < \varepsilon\|b\|$ with $\varepsilon = 10^{-6}$. A final check is synchronously carried out after the iterations loop.

4.2 Scaling

We consider a weak scaling experiment, where both the size of the problem and the number of subdomains are increased by the same ratio. The size of the local problems is kept constant at approximately 20000 unknowns, while the size of the global coarse problem equals the number of subdomains. One of the involved processes handles the whole coarse domain, which, up to $p = 1600$ subdomains, does not yet raise a significant overhead issue, compared to the size of the subdomain. A discrete overlap of two mesh steps is considered throughout the experiment.

Numbers of iterations and total elapsed times are reported in Figure 1. Comparing

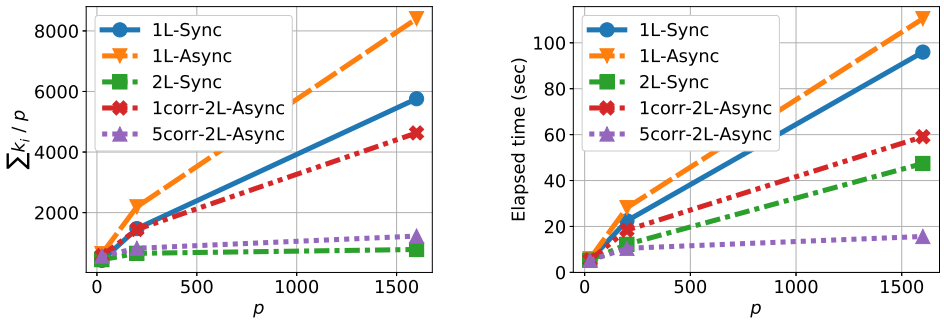

Figure 1: Weak scaling of one-level and two-level RAS solvers. "1corr-2L" corresponds to 1 correction per coarse solution as in [15], and "5corr-2L" allows for up to 5 corrections. k_i is the number of iterations on the i-th subdomain.

the synchronous solvers, there is a clear impact of the coarse-space correction. The number of iterations of the 2L-Sync solver is nearly constant, which confirms the effectiveness of the implemented coarse space. Comparison between synchronous and asynchronous solvers is relevant on elapsed time instead. For the one-level solvers, we note that, while showing the same scaling behavior, the synchronous one is still slightly faster at 1600 cores, which can be explained by the homogeneity of the cluster and the balanced workload. We observe the same result with the 2L-Sync and the 1corr-2L-Async solvers. Such a context, which is favorable to synchronous execution, strengthen the far better performance of the 5corr-2L-Async solver over the 2L-Sync and the 1corr-2L-Async solvers. This clearly confirms the benefit from allowing for more successive corrections using the same coarse solution, which is made possible here by improving the accuracy of the coarse residual. We should mention that the upper bound of 5 corrections per coarse solution became operative only at 1600 subdomains, where coarse solutions were less frequent, probably due to the cost of global communication towards the coarse domain. This therefore strongly suggests the need for bounded delays in the theoretical analysis of two-level asynchronous iterations.

5 Conclusion

The development of effective asynchronous iterative solvers within the framework of domain decomposition methods is still at an early stage, however, promising results have been recently achieved, such as two-level asynchronous domain decomposition methods, within both overlapping Schwarz and primal Schur frameworks. In this paper, we have proposed an improvement of the asynchronous additive coarse-space correction scheme applied in the overlapping Schwarz framework, which has resulted in a far more effective two-level asynchronous solver. Still, numerical experiments have also pointed out the necessity of deeper theoretical investigation of asynchronous coarse-space correction models, notably relying on bounded delays.

It turns out that effective implementation of asynchronous coarse space is an issue similar to the asynchronous convergence detection one, both of them being related to the accurate evaluation of a global residual. While accurate convergence detection is sometimes seen as of a low importance, building an accurate global residual now appears as a key ingredient for achieving effective two-level asynchronous solvers.

Acknowledgements

This work was partly funded by the French National Research Agency as part of project ADOM, under grant number ANR-18-CE46-0008.

References

[1] A. Toselli, O. Widlund, *Domain Decomposition Methods - Algorithms and Theory*, Volume 34 of *Springer Series in Computational Mathematics*, Springer-Verlag Berlin Heidelberg, 2005, page 450.

[2] V. Dolean, P. Jolivet, F. Nataf, *An Introduction to Domain Decomposition Methods – Algorithms, Theory, and Parallel Implementation*, SIAM, Philadelphia, PA, USA, 2015, pages IX, 233.

[3] D. Chazan, W. Miranker, "Chaotic Relaxation", *Linear Algebra Appl.*, 2(2): 199–222, 1969.

[4] A. Frommer, D.B. Szyld, "On asynchronous iterations", *J. Comput. Appl. Math.*, 123, 201–216, 2000.

[5] P. Spiteri, "Parallel asynchronous algorithms: A survey", *Adv. Eng. Softw.*, 149: 102896, 2020.

[6] G. Gbikpi-Benissan, M. Rynkovskaya, F. Magoulès, "Scalable asynchronous domain decomposition solvers for non-homogeneous elastic structures", *Adv. Eng. Softw.*, 174: 103299, 2022.

[7] G. Gbikpi-Benissan, F. Magoulès, "Asynchronous multisplitting-based primal Schur method", *J. Comput. Appl. Math.*, 425: 115060, 2023.

[8] C. Glusa, E.G. Boman, E. Chow, S. Rajamanickam, P. Ramanan, "Asynchronous One-Level and Two-Level Domain Decomposition Solvers", in R. Haynes, S. MacLachlan, X.C. Cai, L. Halpern, H.H. Kim, A. Klawonn, O. Widlund (Editors), *Domain Decomposition Methods in Science and Engineering XXV*, pages 134–142. Springer International Publishing, Cham, Switzerland, 2020.

[9] G. Gbikpi-Benissan, F. Magoulès, "Asynchronous Multiplicative Coarse-Space Correction", *SIAM J. Sci. Comput.*, 44(3): C237–C259, 2022.

[10] S.A. Savari, D.P. Bertsekas, "Finite termination of asynchronous iterative algorithms", *Parallel Comput.*, 22(1): 39–56, 1996.

[11] J.M. Bahi, S. Contassot-Vivier, R. Couturier, "An Efficient and Robust Decentralized Algorithm for Detecting the Global Convergence in Asynchronous Iterative Algorithms", in *High Performance Computing for Computational Science*

- *VECPAR 2008*, Volume 5336 of *Lecture Notes in Computer Science*, pages 240–254. Springer Berlin Heidelberg, 2008.

[12] J. Miellou, P. Spiteri, D. El Baz, "A new stopping criterion for linear perturbed asynchronous iterations", *J. Comput. Appl. Math.*, 219(2): 471–483, 2008.

[13] F. Magoulès, G. Gbikpi-Benissan, "Distributed Convergence Detection Based on Global Residual Error Under Asynchronous Iterations", *IEEE Trans. Parallel Distrib. Syst.*, 29(4): 819–829, 2018.

[14] G. Gbikpi-Benissan, F. Magoulès, "Protocol-free asynchronous iterations termination", *Adv. Eng. Softw.*, 146: 102827, 2020.

[15] C. Glusa, E.G. Boman, E. Chow, S. Rajamanickam, D.B. Szyld, "Scalable Asynchronous Domain Decomposition Solvers", *SIAM J. Sci. Comput.*, 42(6): C384–C409, 2020.

[16] F. Magoulès, G. Gbikpi-Benissan, "JACK2: An MPI-based communication library with non-blocking synchronization for asynchronous iterations", *Adv. Eng. Softw.*, 119: 116–133, 2018.

[17] X.C. Cai, M. Sarkis, "A Restricted Additive Schwarz Preconditioner for General Sparse Linear Systems", *SIAM J. Sci. Comput.*, 21(2): 792–797, 1999.

[18] A. Frommer, H. Schwandt, D.B. Szyld, "Asynchronous Weighted Additive Schwarz Methods", *Electron. Trans. Numer. Anal.*, 5: 48–61, 1997.

[19] A. Frommer, D.B. Szyld, "An Algebraic Convergence Theory for Restricted Additive Schwarz Methods Using Weighted Max Norms", *SIAM J. Numer. Anal.*, 39(2): 463–479, 2001.

[20] M. Chau, D. El Baz, R. Guivarch, P. Spiteri, "MPI implementation of parallel subdomain methods for linear and nonlinear convectionâ€"diffusion problems", *J. Parallel Distrib. Comput.*, 67(5): 581–591, 2007.

[21] J. Wolfson-Pou, E. Chow, "Reducing Communication in Distributed Asynchronous Iterative Methods", *Procedia Computer Science*, 80: 1906–1916, 2016.

[22] W. Hackbusch, *Multi-Grid Methods and Applications*, Volume 4 of *Springer Series in Computational Mathematics*, Springer-Verlag Berlin Heidelberg, 1985, pages XIV, 378.