SIAM J. SCI. COMPUT.
Vol. 38, No. 1, pp. C22–C52

MULTILEVEL BALANCING DOMAIN DECOMPOSITION AT EXTREME SCALES*

SANTIAGO BADIA†, ALBERTO F. MARTÍN†, AND JAVIER PRINCIPE†

Abstract. In this paper we present a fully distributed, communicator-aware, recursive, and interlevel-overlapped message-passing implementation of the multilevel balancing domain decomposition by constraints (MLBDDC) preconditioner. The implementation highly relies on subcommunicators in order to achieve the desired effect of coarse-grain overlapping of computation and communication, and communication and communication among levels in the hierarchy (namely, interlevel overlapping). Essentially, the main communicator is split into as many nonoverlapping subsets of message-passing interface (MPI) tasks (i.e., MPI subcommunicators) as levels in the hierarchy. Provided that specialized resources (cores and memory) are devoted to each level, a careful rescheduling and mapping of all the computations and communications in the algorithm lets a high degree of overlapping be exploited among levels. All subroutines and associated data structures are expressed recursively, and therefore MLBDDC preconditioners with an arbitrary number of levels can be built while re-using significant and recurrent parts of the codes. This approach leads to excellent weak scalability results as soon as level-1 tasks can fully overlap coarser-levels duties. We provide a model to indicate how to choose the number of levels and coarsening ratios between consecutive levels and determine qualitatively the scalability limits for a given choice. We have carried out a comprehensive weak scalability analysis of the proposed implementation for the three-dimensional Laplacian and linear elasticity problems on structured and unstructured meshes. Excellent weak scalability results have been obtained up to 458,752 IBM BG/Q cores and 1.8 million MPI being, being the first time that exact domain decomposition preconditioners (only based on sparse direct solvers) reach these scales.

Key words. domain decomposition, perdominant preconditioning, parallel computing, high-performance computing, finite elements, scientific software

AMS subject classifications. 65N55, 65F08, 65N30, 65Y05, 65Y20

DOI. 10.1137/15M1013511

1. Introduction.

The simulation of scientific and engineering problems governed by partial differential equations (PDEs) involves the solution of sparse linear systems. The time spent in an implicit simulation at the linear solver relative to the overall execution time grows with the size of the problem and the number of cores [25]. In order to satisfy the ever increasing demand of reality and complexity in the simulations, scientific computing must advance in the development of numerical algorithms and implementations that will efficiently exploit the largest amounts of computational resources, and a massively parallel linear solver is a key component in this process.

*Submitted to the journal's Software and High-Performance Computing section March 23, 2015; accepted for publication (in revised form) November 6, 2015; published electronically January 26, 2016. This work has been funded by the European Research Council under FP7 Programme Ideas through starting grant 258443, COMFUS: Computational Methods for Fusion Technology, and the FP7 NUMEXAS project under grant agreement 611636.

http://www.siam.org/journals/sisc/38-1/M101351.html

†Centre Internacional de Mètodes Numèrics en Enginyeria (CIMNE), Parc Mediterrani de la Tecnologia, UPC, Esteve Terradas 5, 08860 Castelldefels, Spain, and Universitat Politècnica de Catalunya, Jordi Girona 1-3, Edifici C1, 08034 Barcelona, Spain (sbadia@cimne.upc.edu, amartin@cimne.upc.edu, principe@cimne.upc.edu). The first author received support from the Catalan Government through the ICREA Acadèmia Research Program. The second author was also partially funded by the Generalitat de Catalunya under the program "Ajuts per a la incorporació, amb caràcter temporal, de personal investigador júnior a les universitats públiques del sistema universitari català PDJ 2013."

The growth in computational power passes now through increasing the number of cores in a chip, instead of making cores faster. The next generation of supercomputers, able to reach 1 exaflop/s, is expected to reach billions of cores. Thus, the future of scientific computing will be strongly related to the ability to efficiently exploit these extreme core counts [1].

Only numerical algorithms with all their components scalable will efficiently run on extreme scale supercomputers. On extreme core counts, it will be a must to reduce communication and synchronization among cores and overlap communication with computation. At the largest scales, linear solvers are based on preconditioned Krylov subspace methods. Algorithmically scalable preconditioners include algebraic multigrid (AMG) [35] and some domain decomposition (DD) algorithms [36]. However, this theoretical property is not enough for practical weak scalability, since the preconditioner itself must allow for a massively scalable implementation. Today's most scalable algorithms/implementations present practical limits of parallelism, e.g., due to the small, coarse problems to be solved in the hierarchical process for DD/AMG, and the loss of sparsity and denser communication patterns at coarser levels of AMG [7].

DD preconditioners make explicit use of the partition of the global mesh, e.g., for the finite element (FE) integration, into submeshes (subdomains), and provide a natural framework for the development of fast and robust parallel solvers tailored for distributed-memory machines. One-level DD algorithms involve the solution of local problems and nearest-neighbors communications. A (second-level) coarse correction (coupling all subdomains) is required to have algorithmic scalability, but it can also harm the practical (CPU time) weak scalability. Two-level DD algorithms include the balancing Neumann–Neumann preconditioner [26], the balancing DD by constraints (BDDC) preconditioner [15], and FETI-DP preconditioners [17]. The practical scalability limits of a two-level DD implementation is determined by the coarse solver computation, whose size increases (at best) linearly with respect to the number of subdomains. The coarse problem rapidly becomes the bottleneck of the algorithm as we increase the number of cores, since it irremediably produces a severe parallel efficiency loss (all the cores not involved in the coarse solver computation are idling).

Weak scalability can be sustained even when incurring in notable parallel efficiency loss by increasing the number of cores dealing with the coarse solver, e.g., by means of a message-passing sparse direct solver as the one in MUMPS [2]. The complexity of this type of solver is quadratic for three-dimensional (3D) problems, and a quadratic increase of the number of cores is needed to keep constant the computational time. However, the scalability of sparse direct solvers is limited to some hundreds of cores, harming the overall weak scalability of the two-level BDDC method at some point [3].

Traditionally, DD methods solve subproblems with sparse direct methods, leading to the so-called exact DD methods. A salient feature of BDDC methods is the fact that the constrained Neumann and Dirichlet local problems, as well as the coarse problem, can be computed in an inexact way, e.g., using one AMG cycle without affecting the algorithmic scalability of the method [16]. The use of inexact coarse solvers is also possible for FETI-DP methods, after some modifications [22]. Since AMG solvers maintain weak scalability much further than message-passing sparse direct methods, *inexact* versions of BDDC and FETI-DP methods exhibit improved weak scalability, even though such implementations still incur in the parallel efficiency loss commented above. Inexact FETI-DP methods, in which local problems are computed with direct

solvers and the coarse problem is approximated using AMG, have been exploited in [22].

The BDDC preconditioner has some salient properties that permit US to overcome this parallel overhead, making it an excellent candidate for extreme scale solver design:

(P1) It allows for a mathematically supported extremely aggressive coarsening and the coarse matrix has a similar sparsity pattern as the original system matrix. On memory-constrained supercomputers, the coarsening ratio is on the order of 10^5 for sparse direct methods [5] (see section 5).

(P2) Coarse and fine components can be computed in parallel [5], since the basis for the coarse space is constructed in such a way that it is orthogonal to the fine component space with respect to the inner product endowed by the unassembled system matrix [15, 28].

(P3) Due to the fact that the coarse matrix has a similar structure as the original system matrix, a multilevel extension of the algorithm is possible [37, 29].

Property (P1) is readily exploited in any BDDC implementation.

The efficient exploitation of (P2), i.e., the orthogonality between coarse and fine spaces, is not trivial. However, this property makes possible a parallel computation of coarse and fine corrections, i.e., overlapped in time. In [5], we have classified all the duties in an exact (i.e., using sparse direct solvers) BDDC-PCG algorithm into fine and coarse duties. These duties have been rescheduled to achieve the maximum degree of overlapping while preserving data dependencies. The actual implementation of this idea requires significant code refactoring, since it involves a switch from a data parallelism to a task parallelism paradigm, dividing processors into those having fine grid duties and those having coarse grid duties. This bulk-asynchronous approach reduces synchronization among cores and overlaps communications/computations, following the exascale solver paradigm [1]. It has been exploited in [5], where we have performed scalability analyses for the 3D Poisson and linear elasticity problems on a pair of state-of-the-art multicore-based distributed-memory machines (HELIOS and CURIE). Excellent weak scalability has been attained up to 27K cores for reasonably high local problem sizes; both local and coarse problems were solved by using the multithreaded sparse direct solver PARDISO [33]. Further, the clear reduction of computing time and memory requirements of inexact solvers compared to sparse direct ones made possible to get in [6] excellent weak scalability results for the inexact/overlapped implementation of the two-level BDDC preconditioner, up to 93,312 cores and 20 billion unknowns on JUQUEEN.

With regard to (P3), a multilevel BDDC (MLBDDC) algorithm has been proposed in [29], where the coarse problem at the next BDDC level is approximated by its BDDC approximation. An implementation of the MLBDDC method that does not exploit (P2) can be found in [34]. Even though the CPU cost of the coarse problem is reduced, the implementation in [29] still suffers from parallel efficiency loss, since the coarse problem is still serialized with respect to the fine component. Further, since the condition number bound slightly increases with the number of levels [29], the detailed numerical experiments in [34] are not conclusive to prove the benefit of the multilevel extension (see, e.g., Tables 4, 7, and 8 in [34]).

In this work, we extend the approach in [5] for the two-level BDDC method to MLBDDC. Exact two-level methods with serial coarse solver are effective till some tens of thousands of cores. Beyond that point, the coarse problem cannot be fully overlapped anymore by fine duties. In order to go to larger core counts, we extend here the approach in [5] for the two-level BDDC method to MLBDDC, i.e., to exploit both (P2) and (P3). We present a fully distributed, communicator-aware, recursive,

and interlevel-overlapped message-passing implementation of the MLBDDC preconditioner. Fully distributed (versus centralized) means that all data structures (and thus associated computations/communications) are distributed among cores, including the coarse-grid problem at all levels of the MLBDDC preconditioning hierarchy. Communicator-awareness refers to the fact that the codes highly rely on subcommunicators in order to achieve the desired effect of coarse-grain overlapping of computation and communication and overlapping of communication and communication among levels in the hierarchy (namely interlevel overlapping). Essentially, the main communicator is split into as many nonoverlapping subsets of message-passing interface (MPI) tasks (i.e., MPI subcommunicators) as levels in the hierarchy. Provided that specialized hardware resources (cores and memory) are devoted to each level, a careful rescheduling and mapping of all the computations and communications in the algorithm lets a high degree of overlapping be exploited among levels. Finally, recursive implementation means that all subroutines and associated data structures are expressed recursively, and therefore MLBDDC preconditioners with an arbitrary number of levels can be built while reusing recursively recurrent parts of the code (e.g., communicator creation, local solvers, interlevel data transfers).

This approach leads to excellent weak scalability results as soon as level-1 tasks can fully overlap coarser-level duties. We provide a model to indicate how to choose the number of levels and coarsening ratios between consecutive levels and determine qualitatively the scalability limits for a given choice. Finally, we present a comprehensive weak scalability analysis of the proposed implementation for the 3D Laplacian and linear elasticity problems. Excellent weak scalability results have been obtained up to 458,752 cores and 1.8 million MPI tasks. These are unprecedented results for exact DD preconditioners (equipped with sparse direct solvers) that represent more than one order of magnitude (in terms of scalability limits) improvement with respect to the best results up to now [5], showing the tremendous potential of the algorithmic approach proposed herein. Thanks to these results, the software platform FEMPAR [4] in which we have implemented the MLBDDC algorithm is in the High-Q club (since 2014) of the most scalable codes on the JUQUEEN IBM BG/Q supercomputer [12].

Related problems appear in multigrid (MG). Additive MG methods, like the Bramble–Pasciak–Xu (BPX) scheme [10], have been traditionally neglected because their converge is significantly slower than multiplicative ones [9, 13] (although the former expose a higher degree of parallelism to be exploited). However, multiplicative AMG parallel codes are nowadays facing scalability issues on state-of-the-art petascale supercomputers, mainly caused by poor communication/computation balances at the coarsest levels. These issues can certainly limit their applicability on future exascale platforms [7]. With this scenario in mind, new additive AMG schemes have been proposed in [38], the most expensive ones being mathematically equivalent to their multiplicative counterparts. It was shown for the first time in [38] that parallel additive AMG codes can strike a balance among preconditioner robustness and degree of parallelism such that they end up being significantly faster than multiplicative ones already on some thousands of cores. This outcome is even more encouraging provided that the parallel code studied in [38] solely exploits fine-grained computation and communication overlap by means of nonblocking communication. As far as we know, there are no advanced implementations of additive methods that exploit concurrent smoothing at all levels, but based on these recent results, and the expected properties of future exascale supercomputers, additive AMG schemes can certainly benefit from the coarse-grain interlevel computation-computation, computation-communication, and

communication-communication overlapping ideas proposed herein for DD methods.

Let us summarize the contributions of this work:

- Novel recursive implementation of MLBDDC methods that exploit parallelism between levels, to fully overlap the coarse-grained tasks on much larger core counts than two-level methods;
- Detailed exposition about how to efficiently exploit this novel approach on state-of-the-art supercomputers, including the exploitation of recursion in a multilevel setting for the overlapped deployment of tasks and the construction of subcommunicators;
- Comprehensive weak scalability analysis of the proposed strategy for both Laplacian and linear elasticity problems on the IBM BG/Q supercomputer, up to 458,752 cores and 1.8 million MPI tasks (subdomains).

This work is structured as follows. The MLBDDC preconditioner is presented in section 2. In section 3, we present an overlapped MLBDDC implementation of the algorithm, which overlaps multiple-level computations, and we elaborate a model to determine the scalability limits for a given choice of the number of levels and the coarsening ratios between consecutive levels. In section 4 we provide some implementation keys such as the recursive creation of MPI subcommunicators. In section 5, we report a comprehensive set of numerical experiments. Finally, in section 6, we draw some conclusions and define future lines of work.

2. Multilevel balancing domain decomposition. In this section we state the MLBDDC preconditioner. In section 2.1, we introduce some basic notation. Section 2.2 is devoted to the two-level BDDC algorithm. Finally, the MLBDDC algorithm is defined in a recursive way in section 2.3, relying on the concepts introduced in section 2.2 for the two-level algorithm. In any case, the description of the algorithms is concise, and we refer the reader to [27, 11] for a detailed exposition of two-level BDDC methods, and to [29] for the multilevel extension. Further, a thorough presentation of practical implementation details for DD methods can be found in [4].

2.1. Problem setting. Let us consider a bounded polyhedral domain $\Omega \subset \mathbb{R}^d$ with $d = 2, 3$ and a quasi-uniform partition (mesh) $\mathcal{T}_0$ with characteristic size h_0. Usually, $\mathcal{T}_0$ is a partition into tetrahedra/hexahedra for $d = 3$ or triangles/quadrilaterals for $d = 2$. We consider a quasi-uniform partition $\mathcal{T}_1$ of $\mathcal{T}_0$ into n_1^{sbd} sub-meshes, which induces a nonoverlapping DD of Ω into subdomains Ω_1^i, $i = 1, \dots, n_1^{\text{sbd}}$ (of characteristic size h_1). The interface of Ω_1^i is defined as $\Gamma_1^i := \partial\Omega_1^i \setminus \partial\Omega$ and the whole interface (skeleton) of the DD is $\Gamma_1 := \bigcup_{i=1}^{n_1^{\text{sbd}}} \Gamma_1^i$.

As a model problem, we study the Poisson problem and linear elasticity on Ω for an arbitrary forcing term and boundary conditions (as soon as the problem is well-posed). Let us consider a conforming FE space $\bar{\mathbb{V}}_1 \subset H^1(\Omega)$. We denote by $\mathbb{V}_1^i$ the restriction of $\bar{\mathbb{V}}_1$ into $\Omega_1^i \in \mathcal{T}_1$, i.e., local FE spaces. $\mathbb{V}_1 := \mathbb{V}_1^1 \times \cdots \times \mathbb{V}_1^{n_1^{\text{sbd}}}$ is the global FE space of functions that can be *discontinuous on* Γ_1. Let us also define the projection $E_1 : \mathbb{V}_1 \to \bar{\mathbb{V}}_1$ as some weighted average of interface values (see, e.g., [27]). We note that $\mathcal{T}_0$ and the FE type defines $\bar{\mathbb{V}}_1$, whereas $\mathcal{T}_1$ is also required to define the local and discontinuous spaces $\mathbb{V}_1^i$ and $\mathbb{V}_1$, respectively. The Galerkin approximation of the problem at hand (with respect to $\bar{\mathbb{V}}_1$) leads to the global linear system of equations to be solved:

$$A_1 x_1 = f_1. \tag{1}$$

The subdomain FE matrix corresponding to $\mathbb{V}_1^i$ is denoted by K_1^i. K_1 is the block-

diagonal global subassembled FE matrix on $\mathbb{V}_1$. (Throught the paper, we denote with the letter K a subassembled matrix and with A the corresponding fully assembled one.) Analogously, we define the local subassembled right-hand side g_1^i and its global counterpart g_1. The system matrix A_1 and right-hand side f_1 can be obtained after the assembly of K_1 and g_1.

The nonoverlapping partition induces a reordering of degrees of freedom (DoFs) into interior and interface DoFs, i.e., $u_1 = [u_{1_I}, u_{1_\Gamma}]^t$. We also define the interior restriction operator $R_{1_I} u_1 := u_{1_I}$. It leads to the following block structure of the global assembled, global subassembled, and local matrices:

$$A_1 = \begin{bmatrix} A_{1_{II}} & A_{1_{I\Gamma}} \\ A_{1_{\Gamma I}} & A_{1_{\Gamma\Gamma}} \end{bmatrix}, \; K_1 = \begin{bmatrix} A_{1_{II}} & K_{1_{I\Gamma}} \\ K_{1_{\Gamma I}} & K_{1_{\Gamma\Gamma}} \end{bmatrix}, \; K_1^i = \begin{bmatrix} A^i_{1_{II}} & A^i_{1_{I\Gamma}} \\ A^i_{1_{\Gamma I}} & K^i_{1_{\Gamma\Gamma}} \end{bmatrix},$$

respectively. Matrices $A_{1_{II}}$, $A_{1_{I\Gamma}}$, $A_{1_{\Gamma I}}$, and $K_{1_{\Gamma\Gamma}}$ present a block-diagonal structure (very amenable to parallelization). Matrices $K_{1_{I\Gamma}}$ and $K_{1_{\Gamma I}}$ are trivial extensions (by zeroes) of $A_{1_{I\Gamma}}$ and $A_{1_{\Gamma I}}$, respectively.

2.2. Two-level BDDC preconditioner. In what follows, we state the two-level BDDC algorithm, describing the set-up of the preconditioner and its application. The *input* required to set up the BDDC preconditioner is $(\mathcal{T}_1, \bar{\mathbb{V}}_1, K_1)$. We recall that $\mathcal{T}_1$ is a subdomain partition, $\bar{\mathbb{V}}_1$ is the global FE space, and K_1 is the global subassembled matrix.

The construction of the BDDC preconditioner requires a partition of the DoFs corresponding to $\bar{\mathbb{V}}_1$ on Γ_1 into objects, which can be corners, edges, or faces. Next, we associate to some (or all) of these objects a *coarse* DoF. The coarse DoFs can be the values of the function at the corners, or the mean values of the function on edges/faces. Three common variants of the BDDC method are referred as BDDC(c), BDDC(ce), and BDDC(cef), where we enforce continuity on only corner coarse DoFs, corner and edge coarse DoFs, and corner, edge, and face coarse DoFs, respectively. The definition of the objects and the coarse DoFs can be implemented as an automatic process for arbitrary partitions and physical problems (see [4]). It involves the use of a kernel detection mechanism in order to preserve the well-posedness of the BDDC preconditioner (see [39]). Once we have defined the coarse DoFs, we can define the BDDC FE space $\tilde{\mathbb{V}}_1$ as the subspace of functions in $\mathbb{V}_1$ that are continuous on coarse DoFs; clearly, $\bar{\mathbb{V}}_1 \subset \tilde{\mathbb{V}}_1 \subset \mathbb{V}_1$.

The BDDC preconditioner is a Schwarz-type preconditioner that combines interior corrections with corrections in the BDDC space $\tilde{\mathbb{V}}_1$ (see, e.g., [11, 36]). We define the interior correction operator as $P_1 := R_{1_I}^t A_{1_{II}}^{-1} R_{1_I}$, which involves the set-up of $(A^i_{1_{II}})^{-1}$, i.e., local Dirichlet problems. The BDDC correction is expressed as $E_1 \tilde{K}_1^{-1} E_1^t$, where $\tilde{K}_1$ is the Galerkin projection of K_1 onto $\tilde{\mathbb{V}}_1$.

The set-up of the BDDC correction requires some elaboration. Let us consider a decomposition of the BDDC space $\tilde{\mathbb{V}}_1$ into a *fine space* $\tilde{\mathbb{V}}_{1_f}$ of vectors that vanish on coarse DoFs and the K-orthogonal complement $\tilde{\mathbb{V}}_{1_c}$, denoted as the *coarse space*. As a result, the BDDC FE problem can be decomposed into fine and coarse components, i.e., $\tilde{x}_1 = \tilde{K}_1^{-1} E_1^t r_1 = x_{1_f} + x_{1_c}$. Since fine and coarse spaces are K_1-orthogonal by definition, they can be computed in parallel.

The fine space functions in $\tilde{\mathbb{V}}_{1_f}$ vanish on coarse DoFs (which are the only DoFs that involve continuity among subdomains). Due to the K_1-orthogonality, the fine component can be defined as $x_{1_f} := E_1 K_{1_f}^{-1} E_1^t$, where K_{1_f} is the Galerkin projection of K_1 onto $\tilde{V}_{1_f}$. Let us note that, since the coarse DoFs are fixed (to zero) in the fine correction, and these are the only DoFs that couple subdomains, the set-up of $K_{1_f}^{-1}$ only involves the solution of local Neumann problems with constrained values at the corresponding subdomain coarse DoFs, denoted by $(K_{1_f}^i)^{-1}$. For a comprehensive exposition of the implementation issues regarding the solution of constrained Neumann problems we refer to [4].

The coarse space $\tilde{\mathbb{V}}_{1_c} \subset \tilde{\mathbb{V}}_1$ is built as $\tilde{\mathbb{V}}_{1_c} = \text{span}\{\phi_1^1, \phi_1^2, \ldots, \phi_1^{n_1^{\text{cts}}}\}$, where $\phi_1^\alpha \in \tilde{\mathbb{V}}_{1_c}$ is the *coarse shape function* associated to the coarse DoF α, i.e., it takes value one on α and vanishes on the rest of the coarse DoFs. As a result, its computation also involves $K_{1_f}^{-1}$ (see [4]). Let us note that the support of ϕ_1^α is the set of subdomains that contain α. Thus, at every subdomain we only compute the coarse space basis functions related to owned coarse DoFs. We denote by Φ_1 the matrix with columns the coarse shape functions. We define the coarse matrix K_{1_c} as the assembly of the subdomain *local* matrices $K_{1_c}^i = {\Phi_1^i}^t K_1^i \Phi_1^i$ for $i = 1, \ldots, n_1^{\text{sbd}}$. Further, in a two-level algorithm, we have to set up $A_{1_c}^{-1}$, e.g., after assembling the subdomain contributions in one processor. Using the fact that $P_1 A_1 P_1 = P_1$, the two-level BDDC preconditioner can be stated in a compact form as

$$M_1 := P_1 + (I_1 - P_1 A_1) E_1^t (\Phi_1 A_{1_c}^{-1} \Phi_1^t + K_{1_f}^{-1}) E_1 (I_1 - P_1 A_1)^t,$$

where I_1 is the identity matrix.

2.3. Multilevel BDDC preconditioner. The two-level BDDC algorithm involves as input $(\mathcal{T}_1, \bar{\mathbb{V}}_1, K_1)$ and automatically generates the coarse DoFs and the corresponding coarse BDDC space $\tilde{\mathbb{V}}_{1_c}$ and coarse matrix K_{1_c}. We can easily observe that the BDDC method is suitable for a multilevel generalization (see [37] for three levels and [29] for the general case). We can consider a coarse partition $\mathcal{T}_2$ of $\mathcal{T}_1$, obtained by aggregation (coarsening) of subdomains in $\mathcal{T}_1$. For every subdomain $\Omega_2^j \in \mathcal{T}_2$, its portion of the coarse system matrix is at our disposal via the assembly of local sub assembled matrices $K_{1_c}^i$ for subdomains $\Omega_1^i \in \mathcal{T}_1$ such that $\Omega_1^i \subseteq \Omega_2^j$, in the same fashion as above with FE matrices. Further, the coarse space $\tilde{\mathbb{V}}_{1_c}$ is an FE-like space associated to $\mathcal{T}_1$, and we can also generate its corresponding space of discontinuous functions with respect to $\mathcal{T}_2$. Therefore, we can readily define the BDDC preconditioner associated to the coarse system matrix K_{1_c}, the coarse space $\tilde{\mathbb{V}}_{1_c}$, and the partition $\mathcal{T}_2$. In what follows, we express the multilevel method in a formal way.

We create a hierarchy of quasi-uniform partitions $(\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_{n_\ell - 1})$ by aggregation, where n_ℓ is the number of levels in the BDDC method. $\mathcal{T}_{\ell+1}$ is a partition of $\mathcal{T}_\ell$. For every subdomain $\Omega_\ell^i \in \mathcal{T}_\ell$ there is only one $\Omega_{\ell+1}^j \in \mathcal{T}_{\ell+1}$ such that $\Omega_\ell^i \subset \Omega_{\ell+1}^j$, and we will denote j as $fat(\ell, i)$. We can readily use all the notation in section 2.2 just replacing 1 by ℓ.

As *input* of the MLBDDC method, we require the subassembled subdomain matrix K_1 related to $\mathcal{T}_1$, the global FE space $\bar{\mathbb{V}}_1$, and the hierarchical partitions $(\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_{n_\ell - 1})$. The set-up of the MLBDDC preconditioner is as follows:

For $\ell = 1, \ldots, n_\ell - 1$,
- Set up $(P_\ell, K_{\ell_f}^{-1}, K_{\ell_c}, \Phi_\ell, E_\ell)$ using the procedure described in section 2.2 replacing $(\mathcal{T}_1, \bar{\mathbb{V}}_1, K_1)$ by $(\mathcal{T}_\ell, \bar{\mathbb{V}}_\ell, K_\ell)$
- If $\ell == n_\ell - 1$
  - Set up $A_{n_\ell - 1_c}^{-1}$
- Else
  - Initialize the next level with $\bar{\mathbb{V}}_{\ell+1} \equiv \tilde{\mathbb{V}}_{\ell_c}$, $K_{\ell+1} \equiv K_{\ell_c}$

After all these operators are set up, we are in position to define the MLBDDC preconditioner M in a recursive way as follows: $M \equiv M_1$, where

(2)
$$M_\ell = P_\ell + (I_\ell - P_\ell A_\ell) E_\ell^t (\Phi_\ell M_{\ell+1} \Phi_\ell^t + K_{\ell_f}^{-1}) E_\ell (I_\ell - P_\ell A_\ell)^t, \text{ for } \ell = 1, \ldots, n_\ell - 1,$$
$$\text{and } M_{n_\ell} = A_{n_\ell - 1_c}^{-1}.$$

We refer to [29] for a proof of the following theorem, about the condition number of the BDDC-preconditioned system matrix. We note that the results in [29] have been originally proved for the Laplacian problem. It can be extended to the linear elasticity case by combining the two-level bound for the condition number of BDDC and FETI-DP methods for 3D linear elasticity (see, e.g., [24]) with the multilevel analysis in [29].

THEOREM 2.1. *The condition number of the BDDC preconditioned system matrix for the Laplacian and linear elasticity problem is bounded as*

$$\kappa(MA) \leq C \prod_{\ell=1}^{n_\ell - 1} \left(1 + \log\left(\frac{h_\ell}{h_{\ell-1}}\right)\right)^2$$

for BDDC(c) or BDDC(ce) in $2D$ *and BDDC(ce) and BDDC(cef) in* $3D$*, where* $C > 0$ *is a constant that does not depend on the hierarchical partition, i.e., number of levels* n_ℓ *and characteristic sizes.*

The condition number for MLBDDC methods depends on n_ℓ, even though in a mild way. Let us note that this fact seems to be less than an issue in practice (see section 5.2.2). Due to the aggressive coarsening, MLBDDC methods require much fewer levels than AMG for the same global problem size. Adding one additional level to BDDC in our proposed overlapped implementation can dramatically improve scalability. Tests under low loaded cores for which the two-level BDDC codes in [5] did not scale beyond 1K cores can now perfectly scale up to 458,752 cores (and beyond) in section 5. Four levels will probably be sufficient in many simulations on exascale supercomputers.

On the other hand, MG schemes lead to condition numbers independent of the number of levels. But they do not allow for a mathematically supported aggressive coarsening (as BDDC does) and most of them suffer from a densification of matrices with the coarsening process. In fact, these two issues are the main bottlenecks of MG methods at large scales. We note, however, that densification can be addressed by aggregation-based AMG combined with K-cycling [30].

3. An extreme scale parallel distributed-memory implementation. In this section we cover in detail a novel approach for the parallel distributed-memory implementation of the MLBDDC-PCG algorithm. In section 3.1, we present the rationale underlying the novel approach that we pursue for the extreme scale implementation of this algorithm. In section 3.2 we cover the building blocks of the parallel

algorithm subject of study. In section 3.3 we discuss a use case that illustrates how the techniques proposed are applied to a three-level BDDC-PGC solver in order to reach maximum performance benefit. Finally, in section 3.4 we analyze how to choose values for the coarsening ratios governing subdomain aggregation in order to let the techniques proposed achieve an efficient parallel execution, including an estimation for the number of subdomains in which the global problem can be split while still reaching this goal.

3.1. Rationale underlying novel implementation approach. In this section we present the rationale behind the novel approach that we pursue for the extreme scale implementation of the PCG-MLBDDC solver. This rationale is built around Figure 1, which illustrates two possible implementation approaches for this algorithm.

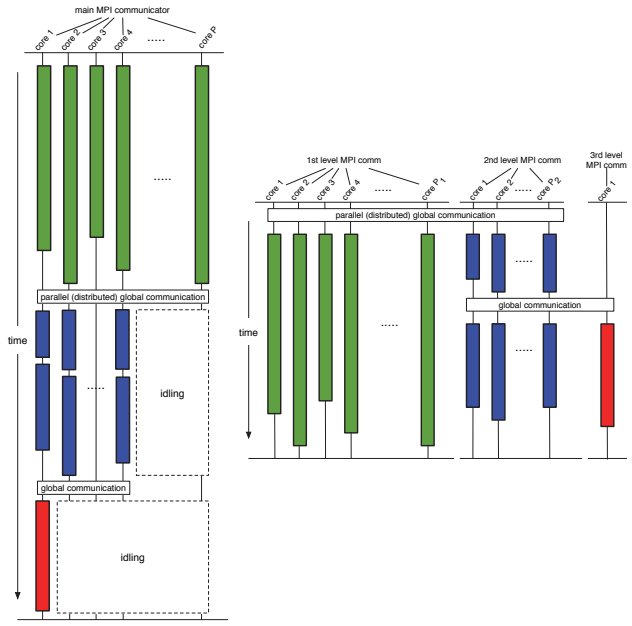


FIG. 1. *Pictorial view of two possible approaches for the distributed-memory implementation of the MLBDDC preconditioner.* (a) *Recursive, fully distributed.* (b) *Recursive, fully distributed, communicator-aware, interlevel overlapped.*

The implementation approach illustrated in Figure 1(a) is the one typically followed by most of the existing code implementations of (ML)BDDC and related DD algorithms [8, 31, 34]. It comes naturally to mind as it reflects the multilevel structure of the preconditioner. It is also relatively easy to code due to its bulk-synchronous structure. However, *it does not exploit all the parallelism which is readily available in the algorithm as it serializes the computation and application of the MLBDDC hierarchy.* For example, as discussed in section 2, the fine-grid and coarse-grid correction can be computed in parallel due to the K-orthogonality constraint underlying the BDDC space. Many other opportunities for parallelism are not exploited by this implementation approach. (See section 3.3 for full coverage of such opportunities in the case of a three-level MLBDDC-PCG solver.) As a first side effect, there is a significant loss of parallel efficiency due to idle MPI tasks. Due to the aggressive

coarsening of these methods, the number of tasks at level 1 is much larger than those at higher levels. Thus, in such implementations, there is a notable aggregated idling time roughly equal to the number of cores being used times the time spent at levels higher than 1. It also has a negative impact in the energy consumption of such implementations. As a second side effect, the memory available on the core responsible for the coarsest-grid problem has to be shared among data structures corresponding to all levels. Provided the already very limited memory per core on current (and future) multicore-based massively parallel processors (e.g., it is only 1 GB for the IBM BG/Q supercomputer), this limits even more the load per core that fits into memory (and thus that of the global problem size).[1]

Figure 1(b) illustrates the novel implementation approach that we propose in this paper. It pursues full exploitation of the parallelism available in the algorithm. In order to reach this goal, the global MPI communicator is split into as many *disjoint* subsets of MPI tasks, i.e., subcommunicators, as levels in the MLBDDC preconditioning hierarchy. All MPI tasks are only in charge of a single subdomain at a particular level. Besides, the steps of the algorithm are reorganized (i.e., rescheduled) in such a way that computation and communication in the global critical path is performed/issued as soon as possible, letting a high degree of coarse-grain overlapping be exploited among levels. Provided that the MPI tasks at each level are mapped to disjoint (specialized) compute nodes, the full memory available per core can be used to accommodate the data structures corresponding to each of the levels in the hierarchy. The target is to tune the number of levels and number of tasks per level for the problem at hand in such a way that first-level duties can completely absorb (i.e., fully overlap) coarser-grid duties by the effect of inter-level overlapping. We note that this strategy has been pursued in [5] for a two-level BDDC preconditioner with remarkable scalability for the solution of 3D Laplacian and linear elasticity problems on medium-sized clusters (up to 27K cores). In order to boost scalability up to current supercomputer core counts (in the order of 1 million), in this work we extend the techniques in [5] from the two-level BDDC method to the multilevel setting.

3.2. Building blocks. In Algorithm 1 we present the main phases of the parallel distributed-memory solution of the linear system (1) via the MLBDDC-PCG solver. After the initial setup in line 1 of Algorithm 1 the iterative PCG solver using the MLBDDC preconditioner is called in line 4. The PCG consists of a repeated sequence of the following four basic operations: application of the preconditioner, sparse matrix-vector products, inner products, and vector updates [32].

ALGORITHM 1. Solve $A_1 x_1 = f_1$.

1: Set up M (symbolic and numeric stages) and set initial solution x_1^0
2: $x_{1_I}^0 := x_{1_I}^0 + A_{1_{II}}^{-1} R_{1_I}(f_1 - A_1 x_1^0)$
3: $r_1^0 := f_1 - A_1 x_1^0$
4: $x_1 := \mathrm{PCG}(A_1, M, r_1^0, x_1^0)$ Invokes Algorithm 2

In the message-passing implementation of Algorithm 1, all matrices, vectors, and associated computations are distributed among MPI tasks conformally with the hierarchy of nonoverlapping partitions underlying the MLBDDC preconditioner. Let us

[1] We note that a distributed-memory solver for the coarsest-grid problem [31, 18, 34] can mitigate the impact of these side effects, but the parallel overhead due to idle MPI tasks still remains.

denote by $task(\ell, i)$ the one-to-one mapping that assigns an MPI task to every subdomain $\Omega_\ell^i \in \mathcal{T}_\ell$ for every level ℓ. Any vector $x_\ell \in \mathbb{V}_\ell$ is distributed conformally with $\mathcal{T}_\ell$, i.e., the portion x_ℓ^i of x_ℓ is stored in MPI task $task(\ell, i)$. Analogously, matrix K_ℓ is distributed such that every diagonal block K_ℓ^i is stored in MPI task $task(\ell, i)$. Vectors in $\bar{\mathbb{V}}_\ell \subset \mathbb{V}_\ell$ are distributed as those in $\mathbb{V}_\ell$, with the particularity that they must be continuous on the interface, i.e., values on interface DoFs must match among MPI tasks sharing them. As a result, data structures describing the distributed-memory layout of such vectors must provide additional *gluing* information. We refer the reader to [4] for comprehensive coverage of such data structures in a two-level BDDC preconditioner context. The discussion there straightforwardly applies to the MLBDDC preconditioner.

In this work we consider sparse direct methods [14] for the *exact* (up to machine precision) solution of the subproblems within the MLBDDC hierarchy. In particular, preconditioner set-up is split into a symbolic stage followed by a numerical stage. The symbolic stage essentially builds and symbolically factorizes the graph associated to local matrices $A_{\ell_{II}}^{i_\ell}$, $K_{\ell_f}^{i_\ell}$, for $\ell = 1, \ldots, n_\ell - 1$, $i_\ell = 1, \ldots, n_\ell^{\mathrm{sbd}}$, and that of the global coarsest-grid matrix $A_{n_\ell - 1_c}$. As part of the symbolic analysis, a fill-in reordering is applied to the graph. On the other hand, the numerical set-up stage computes the sparse Cholesky factor of (each of) these matrices. Finally, the application of the MLBDDC preconditioner hierarchy, shown in Algorithm 2, solves the corresponding linear systems by means of sparse forward/backward substitution. We note that the Dirichlet precorrection in Algorithm 2 (see lines 2 and 3) can be omitted at the first level, since the interior residual is already zero due to the initial interior precorrection (see line 2 of Algorithm 1).

Let us finally describe the meaning of labels at the end of the steps in Algorithm 2. On the one hand, MPI tasks $task(\ell, \cdot)$ are in charge of the steps labeled as "ℓ," while MPI tasks $\cup_{k=\ell+1}^{n_\ell} task(k, \cdot)$ perform those labeled as "$\ell + 1, \ldots, n_\ell$." Note that in Algorithm 2 we are assuming, without loss of generality, that $A_{n_\ell - 1_c}$ is centralized on a single MPI task at the last level (so that a serial sparse direct solver can be used for the solution of the coarsest-grid problem). However, this problem can also be distributed among several MPI tasks (and solved by means of a message-passing sparse direct solver as the one in MUMPS [2]). On the other hand, labels "$\ell \to \ell + 1$" refer to data transfers among MPI tasks in two consecutive levels. More precisely, among level ℓ MPI tasks, $task(\ell, i)$, and their parents at level $\ell + 1$, i.e., task $task(\ell + 1, j)$ such that $j = fat(\ell, i)$.

3.3. Use case for a three-level BDDC-PCG solver. According to section 3.1, the steps in Algorithms 1 and 2 have to be judiciously reorganized in order to fully exploit all parallelism which is readily available in these algorithms. Table 1 depicts the result of this exercise for a three-level BDDC-PCG solver. (We have considered a three-level algorithm for the sake of simplicity when discussing the overlapping potential of our approach, even though the implementation of the algorithm is recursive and can be applied to an arbitrary number of levels; see section 3.4 and 4.) In this table, the gather and scatter communication stages among MPI tasks at consecutive levels are depicted in gray. Among two consecutive levels, the areas limited by these synchronizing stages can be computed in parallel, leading to three overlapping regions for a pair of consecutive levels. In the following, we refer to these (three) overlapping regions between levels X and Y as (first, second, and third) X-Y regions.

Since a perfect balance among the time spent in the overlapping regions of Table 1 across all levels is hard to strike in practice, we propose instead an almost-optimal

ALGORITHM 2. $z := Mr$ with M defined in (2).

1:	**if** $\ell > 1$ **then**	
2:	Compute $\delta^i_{\ell_I} := (A^i_{\ell_{II}})^{-1} r^i_{\ell_I}$	ℓ
3:	Compute $r^i_{\ell_\Gamma} := r^i_{\ell_\Gamma} - A^i_{\ell_{\Gamma I}} \delta^i_{\ell_I}$	ℓ
4:	**end**	
5:	Compute $r^i_\ell := (E_\ell{}^i)^t r_\ell$	ℓ
6:	Compute $r^i_{\ell_c} := (\Phi^i_\ell)^t r^i_\ell$	ℓ
7:	Gather $r^i_{\ell_c}$	$\ell \to \ell+1$
8:	Compute $s^i_{\ell_f} := (K^i_{\ell_f})^{-1} r^i_\ell$	ℓ
9:	**if** $\ell == n_\ell - 1$ **then**	
10:	$r_{n_\ell - 1}$ = assemble$(r^i_{\ell_c})$	$\ell+1$
11:	Compute $z_{n_\ell-1} := A^{-1}_{n_\ell-1} r_{n_\ell-1}$	$\ell+1$
12:	Scatter $z_{n_\ell-1}$ into $z^i_{\ell_c}$	$\ell+1 \to \ell$
13:	**else**	
14:	$r^j_{\ell_c} :=$ assemble$(r^i_{\ell_c})$ for i such that $j = fat(\ell, i)$	$\ell+1$
15:	Define $r_{\ell+1} \leftarrow r_{\ell_c}$, $z_{\ell+1} \leftarrow z_{\ell_c}$,	
16:	and invoke Algorithm 2 with $\ell \leftarrow \ell+1$	$\ell+1, \ldots, n_\ell$
17:	Scatter $z^j_{\ell+1}$ into $z^i_{\ell_c}$, for i such that $j = fat(\ell, i)$	$\ell+1 \to \ell$
18:	**end**	
19:	Compute $s^i_{\ell_c} := \Phi^i_\ell z^i_{\ell_c}$	ℓ
20:	Compute $z^i_\ell := E^i_\ell (s^i_{\ell_f} + s^i_{\ell_c})$	ℓ
21:	Compute $z^i_{\ell_I} := -(A^i_{\ell_{II}})^{-1} A^i_{\ell_{I\Gamma}} z^i_{\ell_\Gamma}$	ℓ
22:	**if** $\ell > 1$ **then**	
23:	Compute $z^i_{\ell_I} := z^i_{\ell_I} + \delta^i_{\ell_I}$	ℓ
24:	**end**	

strategy. This strategy is based on a set of estimates presented in section 3.4 that determine how large coarsening ratios among levels can be such that *level*-1 *MPI tasks never wait at collectives*, i.e., they do not waste resources idling at these communication stages. These estimates are such that they still allow for an extremely aggressive coarsening, ending up with almost all tasks at the first level. Indeed, let us remark that in the largest scale numerical examples in section 5, *around* 99% *of the MPI tasks are at the first level*. With this strategy the bulk of the aggregated computation time is concentrated on level-1 tasks. Any idling in the second-or higher-level tasks will only represent a small fraction of the aggregated computation time.

Let us assume for a moment that level-3 MPI tasks do not delay level-2 ones. We describe how we can achieve idling-free level-1 tasks, restricting ourselves to the most computationally dominant steps:

- In the first 1-2 region, the symbolic factorization of the graph associated to Dirichlet and constrained Neumann problems, the numerical factorization of a constrained Neumann problem, and the computation of $\Phi^{i_1}_1$ at the first level, should take at least the same time as the former two symbolic factorizations at the second level. Provided that numerical factorization has a higher order of complexity than symbolic factorization (see section 3.4), this goal is easy to achieve.
- In the second 1-2 region, the numerical factorization of the Dirichlet problem matrix, and the solution of a linear system with this matrix, should take at least the same time as the numerical factorization of the constrained Neumann

TABLE 1

Mapping of Algorithms 1 and 2 *to MPI tasks that maximizes interlevel overlapping for a three-level BDDC-PCG solver.*

$\ell = 1$ MPI tasks	$\ell = 2$ MPI tasks	$\ell = 3$ MPI task
Identify local coarse DoFs		
Gather coarse-grid DoFs		
Symb fact($G_{K_{\ell_f}^{i_1}}$)	Build $G_{K_{1_c}^{i_2}}$	
Symb fact($G_{A_{\ell_{II}}^{i_1}}$)	Identify local coarse DoFs	
Num fact($K_{\ell_f}^{i_1}$)	Gather coarse-grid DoFs	
Compute $\Phi_1^{i_1}$	Symb fact($G_{K_{\ell_f}^{i_2}}$)	Build $G_{A_{2_c}}$
$K_{1_c}^{i_1} := (\Phi_1^{i_1})^t K_1^{i_1} \Phi_1^{i_1}$	Symb fact($G_{A_{\ell_{II}}^{i_2}}$)	Symb fact($G_{A_{2_c}}$)
Gather $K_{1_c}^{i_1}$		
Num fact($A_{\ell_{II}}^{i_1}$)	$K_{1_c}^{i_2} := \text{assemb}(K_{1_c}^{i_1})$	
Solve $A_{1_{II}} \delta_{1_I}^0 = R_{1_I} r_1$	Num fact($K_{\ell_f}^{i_1}$)	
$x_{1_I}^0 := x_{1_I}^0 + \delta_{1_I}^0$	Compute $\Phi_2^{i_2}$	
$r_1^0 := f_1 - A_1 x_1^0$	$K_{2_c}^{i_2} := (\Phi_2^{i_2})^t K_2^{i_2} \Phi_2^{i_2}$	
Algorithm 2 lines 1–6 ($i \equiv i_1$)	Gather $K_{2_c}^{i_2}$	
	Num fact($A_{\ell_{II}}^{i_1}$)	$A_{2_c} := \text{assemb}(K_{2_c}^{i_2})$
		Num fact(A_{2_c})
Gather $r_{1_c}^{i_1}$		
Solve $K_{\ell_f}^{i_1} s_{\ell_f}^{i_1} := r_\ell^{i_1}$	$r_{1_c}^{i_2} := \text{assemb}(r_{1_c}^{i_1})$	
	Algorithm 2 lines 1–6 ($i \equiv i_2$)	
	Gather $r_{2_c}^{i_2}$	
	Solve $K_{\ell_f}^{i_2} s_{\ell_f}^{i_2} := r_\ell^{i_2}$	$r_{2_c} := \text{assemb}(r_{2_c}^{i_2})$
		Solve $A_{2_c} z_{2_c} = r_{2_c}$
	Scatter z_{2_c} into $z_{2_c}^{i_2}$	
	Algorithm 2 lines 19–24 ($i \equiv i_2$)	
Scatter $z_2^{i_2}$ into $z_{1_c}^{i_1}$		
Algorithm 2 lines 19–24 ($i \equiv i_1$)		

and Dirichlet problems matrices and the computation of $\Phi_2^{i_2}$ at the second level. Provided that there are two numerical factorizations at level 2 per one at level 1, it becomes harder to achieve our goal in this case. This is still feasible though, e.g., by enforcing sufficiently smaller subdomain sizes at the second level compared to the sizes of the ones at the first level.

- Analogously, in the last 1-2 region, the solution of a constrained Neumann problem at the first level should at least take the same time as the solution of two Dirichlet problems and a Neumann constrained problem at the second level.

At this point, we note that level-2 regions are in turn split by communication stages among second-and third-level MPI tasks (shaded in gray in Table 1). Therefore, *if the time spent in level-*3 *regions becomes "large enough," then communication stages below these level-*3 *regions can in turn delay second-level MPI tasks with respect to first-level ones up to an extent that starts threatening the parallel efficiency achieved with this approach.* In fact, it irremediably happens at some point when the number of MPI tasks at the coarsest level is kept fixed in a weak scaling analysis. In order to analyze this effect, we split again the discussion among overlapping regions, to infer how the coarsest level duties can harm the effectiveness of the 1-2 regions. We focus on the second and third 1-2 regions since level-3 tasks do not affect the first 1-2 region:

- When the symbolic factorization of the level-3 coarse matrix in the first 2-3 region becomes dominant, i.e., it cannot be fully overlapped by the symbolic

factorization of the graph associated to Dirichlet and constrained Neumann problems and the numerical factorization of a constrained Neumann problem, level-2 MPI tasks have to wait at the corresponding gather communication. As a result, the first 2-3 region can potentially degrade the effectiveness of the second 1-2 region.

- When the numerical factorization of the coarsest matrix (in charge of level-3 MPI tasks) cannot be fully overlapped by the numerical factorization and triangular solve of the level-2 tasks, level-2 tasks have to wait in the corresponding gather communication. Idling of level-2 MPI tasks also occurs at the third 2-3 region, where one sparse backward/forward solve at the third-level cannot be fully overlapped by the three sparse backward/forward solves in the second level. As a result, level-3 tasks can potentially degrade the effectiveness of the third 1-2 region.

In a weak scaling scenario, the subdomain problem size at the first and second levels is fixed. On the other hand, for a fixed number of MPI tasks at the third level, the third-level local problem size increases with the number of subdomains at the second level. As a result, to maximize the scalability of the method, it is interesting to make subdomain coarsening from the first to the second level as aggressive as possible, but always keeping second-level regions fully overlapped by first-level ones. If this strategy does not allow US to strike such a balance within the whole range of core counts of interest, a possible solution is to introduce an additional level in the hierarchy to ameliorate the growth of the time spent at the coarsest level.

We stress that the discussion in this section applies up to minor details to MLBDDC preconditioning hierarchies with an arbitrary number of levels. At any pair of intermediate levels (1-2, 2-3, 3-4, etc.), one has to deal with pairs of regions very close to those 1-2 regions in Table 1 (up to the fact that the input data structures of first-level MPI tasks are provided by an FE integration module, instead of being gathered from the previous level). Any of the regions at the intermediate levels (included the ones at the last level) have the potential of threatening parallel efficiency if the time spent on any of them becomes "sufficiently large" in some sort of delay-ripple effect that finally causes first-level MPI tasks to waste time on the communication stages among first- and second-level MPI tasks. In any case, the effect coarsest-level duties have in higher-level overlapping region pairs is the same regardless of the number of levels.

3.4. Estimates for effective coarsening ratios. In this section we provide a comprehensive discussion about an effective choice of the coarsening ratios governing subdomain aggregation among partitions at different levels, in order to let the overlapped multilevel implementation strike the desired balance among MPI tasks across levels. It depends on the number of levels being used and the subdomain size at every level, which in turn depends on the coarsening ratios being used in the definition of the hierarchical partitions.

Before entering upon the subject let us recall the complexity of the main stages of the direct solution of sparse linear systems. In sparse direct methods, provided that a 3D problem is being solved, the symbolic factorization has a cost $c_{\mathrm{sf}} n^{\frac{4}{3}}$, the numerical factorization a cost $c_{\mathrm{nf}} n^2$, and the triangular system solve a cost $c_{\mathrm{ts}} n^{\frac{4}{3}}$, where n is the size of the coefficient matrix, and the constants c_{sf}, c_{nf}, and c_{ts} do depend on the particular algorithms, their implementation, and the underlying hardware.

Let us now enter upon the subject and consider partitions that are well-balanced at all levels, i.e., all subdomains at the same level have (roughly) the same number of elements. We denote by m_ℓ (resp., N_ℓ) the local (resp., global) number of elements at level ℓ and by p_ℓ the number of subdomains at level ℓ. We note that $N_{\ell+1} = p_\ell$ and that $m_\ell = \frac{N_\ell}{p_\ell} = \frac{p_{\ell-1}}{p_\ell} = \frac{N_\ell}{N_{\ell+1}} = (\frac{h_{\ell+1}}{h_\ell})^d$.

In a weak scaling scenario of one-level algorithms, the global problem size N_1 and the number of subdomains p_1 with level-1 duties increase in such a way that $m_1 = \frac{N_1}{p_1}$ is kept fixed. For a multilevel algorithm, we aim to keep the effectiveness of the preconditioner in terms of iteration counts, which requires US to keep the bound of the condition number given by Theorem 2.1 constant. Thus, we enforce that the local number of elements $m_\ell = \frac{p_{\ell-1}}{p_\ell}$ is fixed at levels $\ell = 1, \ldots, n_\ell - 2$, which involves increasing p_ℓ accordingly. Level n_ℓ is assumed to be mapped to one or at least to a fixed number of MPI tasks.

Given m_ℓ (number of local mesh *elements*, considering the subdomains of the previous level), the actual size of the local system is denoted by $\beta_\ell m_\ell$ (number of local DoFs), where β_ℓ is affected by the type of DoFs in the BDDC method for $\ell > 1$ or by the order of the FE space for $\ell = 1$.

Let us assume that we keep fixed the number of levels n_ℓ in a weak scalability analysis. The questions that arise are, How do we choose the coarsening ratios between levels to attain optimal scalability? Fixing a hierarchical partition, at which number of MPI tasks will the overlapping strategy stop being effective?

The definition of the hierarchical partition can be parameterized by the desired number of elements per level, namely, $m_1, \ldots, m_{n_\ell - 1}$. In order to answer (qualitatively) the former questions, let us make two assumptions:

1. The bulk of the CPU time in the overlapped regions is spent at the different phases of the sparse direct solvers (symbolic factorization, numerical factorization, triangular system solves). Nearest-neighbor communications at level 1, all communications among levels $\ell = 2, \ldots, n_\ell$, and other steps are assumed to be negligible.
2. The cost of the phases of a sparse direct method when applied to the constrained Neumann problem is assumed to be identical to that of the corresponding ones involved in the solution of the Dirichlet problem.

The validity of the first assumption increases with the local problem size, but it is mild for reasonably loaded cores (about 256 KB of memory per core), due to the low CPU cost (and excellent scalability) of nearest-neighbor communications and the reduced number of cores at levels greater than one. The second assumption is also reasonable, since both problems have almost the same size.

Taking into account the steps in each of the three 1-2 region pairs in Table 1, their complexities (see above), and the previous assumptions, we easily infer that the steps at level $\ell+1$ are fully overlapped by level-ℓ steps if the following over-pessimistic condition holds:

$$c_{\mathrm{nf}}(\beta_\ell m_\ell)^2 + 2c_{\mathrm{ts}}(\beta_\ell m_\ell)^{\frac{4}{3}} \geq 2c_{\mathrm{nf}}(\beta_{\ell+1} m_{\ell+1})^2 + n_\Phi c_{\mathrm{ts}}(\beta_{\ell+1} m_{\ell+1})^{\frac{4}{3}}, \tag{3}$$

which corresponds to the second and third 1-2 regions in Table 1, where n_Φ denotes the maximum number of coarse DoFs per subdomain. This is a sufficient condition to fully overlap coarse duties by level-1 (fine) duties, but by any means necessary.

As noted above, in a weak scaling scenario and a fixed number of levels, the coarsest-grid related steps CPU time will increase. Scalability loss will affect the

overall execution time *only* when level-1 regions cannot fully overlap the ones at level 2. Considering the effect that the coarsest level has in level-1 regions (see above), we get the following additional sufficient (over-pessimistic) conditions to fully overlap coarse duties by level-1 (fine) duties:

(4)
$$\beta_1 m_1 \geq \left(\frac{c_{\mathrm{sf}}}{c_{\mathrm{nf}}}\right)^{\frac{1}{2}} (\beta_{n_\ell} m_{n_\ell})^{\frac{2}{3}}, \quad (\beta_1 m_1)^{\frac{4}{3}} \geq (\beta_2 m_2)^{\frac{4}{3}} + (\beta_{n_\ell} m_{n_\ell})^{\frac{4}{3}} + \left(\frac{c_{\mathrm{nf}}}{c_{\mathrm{ts}}}\right) (\beta_{n_\ell} m_{n_\ell})^2,$$

which correspond to the second and third regions, respectively. We can easily observe out of these constraints that the most challenging one is the latter, because we have to fully overlap coarse steps with fine tasks of lower complexity. Let us consider the worst-case scenario in which the last level is centralized on a single MPI task. The number of elements at the last level will increase in a weak scaling scenario as follows:

(5)
$$m_{n_\ell} = p_{n_\ell - 1} = \frac{p_{n_\ell - 2}}{m_{n_\ell - 1}} = \cdots = \frac{p_1}{m_{n_\ell - 1} \cdots m_2}.$$

Replacing (5) in (3) and (4), we get an indication about the maximum number of cores that can be used before losing parallel efficiency. However, in order to use this formula, we need to provide a value for the ratios $\frac{c_{\mathrm{ts}}}{c_{\mathrm{nf}}}$ and $\frac{c_{\mathrm{sf}}}{c_{\mathrm{nf}}}$. We estimated this constant by regression for the sparse direct solver codes and computer architecture used in section 5 for the 3D Laplacian problem for five structured cubic meshes (from 4,096 to 512,000 FEs), getting $\frac{c_{\mathrm{ts}}}{c_{\mathrm{nf}}} \approx 400$ and $\frac{c_{\mathrm{sf}}}{c_{\mathrm{nf}}} \approx 500$. These large values are expected, since the cost per flop for numerical factorization is low, due to the intensive use of the cache hierarchy by means of the level 3 BLAS, whereas the triangular solution and symbolic factorization are memory-bounded.

Let us consider two practical examples from section 5. In three dimensions, and linear FEs, a local problem size per subdomain of $m_1 = 20^3$ fits into the 1 GB of memory available per core. $\beta \approx 1$ for linear FEs, and $\beta \approx 4$ for BDDC(ce) when solving the Laplacian problem on a structured hexahedral mesh. In this case, taking as an example a three-level BDDC(ce) method with $m_2 = 7^3$, condition (3) easily holds, whereas condition (4) combined with (5) leads to $p_1 \lesssim 422,919$ subdomains. A four-level algorithm with $m_1 = 20^3$, $m^2 = 4^3$, and $m^3 = 3^3$ leads to $p_1 \lesssim 2,252,500$. For linear elasticity, where the values of β have to be multiplied by a factor of three, the last case leads to $p_1 \lesssim 1,714,300$. These bounds are in agreement with the numerical results obtained in section 5 for these combinations and are an indication of the potential of the strategy presented so far.

4. Code implementation details. In this section, we sketch, through (simplified) Fortran 95 code snippets, some key hints for the MPI-parallel implementation of the approach presented in section 3. In particular, Listing 1 sketches the initialization of the MLBDDC preconditioner data structure (i.e., `type(mlbddc)` derived data type), while Listing 2 sketches its numerical set-up stage. We stress that for simplicity and brevity, many details related to software engineering practices in our software package (see section 5) have been omitted, as, e.g., those related with algorithm and code parameters, error and memory handling, or data encapsulation. Still, we expect these hints to be useful for practitioners willing to implement in their software the novel techniques proposed herein.

LISTING 1
Simplified Fortran 95 MPI recursive subroutine in charge of preconditioner initialization.

```
recursive subroutine mlbddc_init(comm_world,comm_l1,comm_lgt1,intcomm_l1_lgt1,&
                                 nlev,tsks_x_lev,fat_map,M)
  integer     , intent(in)  :: comm_world           ! World MPI communicator
  integer     , intent(in)  :: comm_l1              ! 1st lev tasks MPI comm
  integer     , intent(in)  :: comm_lgt1            ! > 1st lev tasks MPI comm
  integer     , intent(in)  :: intcomm_l1_lgt1      ! Intercomm l1 <=> lgt1
  integer     , intent(in)  :: nlev                 ! # of levels in MLBDDC
  integer     , intent(in)  :: tsks_x_lev(nlev-1)   ! # of MPI tasks per level
  integer     , intent(in)  :: fat_map(nlev-1)      ! 1st->2nd map of coarse FEs
  type(mlbddc), intent(out) :: M                    ! MLBDDC preconditioner

  ! Local variable declarations (ierr, lgt1_rank, etc.) go here
  M%comm_world=comm_world
  M%comm_l1=comm_l1
  M%comm_lgt1=comm_lgt1
  M%intcomm_l1_lgt1=intcomm_l1_lgt1
  M%nlev=nlev



  call mpi_comm_get_rank(M%comm_world,wrank,ierr)

  ! Create comm_world's subcommunicators for level1 to/from level2 data transfers
  !  * A subcommunicator is created for each task in comm_l2 (l2_rank=0,1,...)
  !  * It includes l2_rank and all tasks in comm_l1 that transfer data to l2_rank
  if (belongs_to(M%comm_l1)) then
    call mpi_comm_split(comm_world, color=fat_map(1), key=wrank, &
                        M%comm_l1_to_l2, ierr)
  else if (belongs_to(M%comm_l2)) then
    call mpi_comm_get_rank(M%comm_l2, l2_rank)
    call mpi_comm_split(comm_world, color=l2_rank, key=wrank, &
                        M%comm_l1_to_l2, ierr)
  else
    call mpi_comm_split(comm_world, color=mpi_undefined, key=wrank, &
                        M%comm_l1_to_l2, ierr)
  end if

  ! Recursively init M%p_M_c (i.e., mlbddc preconditioner for coarse-grid problem)
  if (nlev>2 .and. belongs_to(comm_lgt1)) then
    ! Split comm_lgt1 into comm_l2 and comm_lgt2
    call mpi_comm_get_rank(M%comm_lgt1, lgt1_rank, ierr)
    if (lgt1_rank < tsks_x_lev(2)) then
      call mpi_comm_split(M%comm_lgt1, color=1, key=lgt1_rank,   M%comm_l2, ierr)
      M%comm_lgt2 = mpi_comm_null
    else
      call mpi_comm_split(M%comm_lgt1, color=2, key=lgt1_rank, M%comm_lgt2, ierr)
      M%comm_l2   = mpi_comm_null
    end if

    ! Create intercomm comm_l2 <=> comm_lgt2
    if (belongs_to(M%comm_l2)) then
      call mpi_intercomm_create( M%comm_l2, loc_lead=0, comm_world, &
                                 & rem_lead=tsks_x_lev(2),intcomm_l2_lgt2,ierr)
    else
      call mpi_intercomm_create( M%comm_lgt2, loc_lead=0, comm_world, &
                                 & rem_lead=0,intcomm_l2_lgt2,ierr)
    end if

    ! comm_lgt1 => world_comm, comm_l2 => comm_l1, comm_lgt2 => comm_lgt1
    ! intcomm_l2_lgt2 => intcomm_l1_lgt1
    call mlbddc_init(M%comm_lgt1, M%comm_l2, M%comm_lgt2, intcomm_l2_lgt2,  &
                     nlev-1, tsks_x_lev(2:), fat_map(2:), M%p_M_c)
  end if
end subroutine mlbddc_init
```

LISTING 1. (*cont.*).

```
1 function belongs_to(mpi_comm)
2   integer, intent(in)  :: mpi_comm
3   logical              :: belongs_to
4
5   belongs_to = (mpi_comm /= mpi_comm_null)
6 end function belongs_to
```

Let us start with the subroutine in Listing 1. This subroutine takes as input arguments a set of four MPI communicator handlers. `comm_world` is a communicator handler that includes all MPI tasks that contribute to the computation. (In most cases it will be `mpi_comm_world` or, more conveniently, a duplicate of it.) `comm_l1` and `comm_lgt1` are two communicator handlers for (disjoint) subcommunicators of `comm_world` which include the MPI tasks at the first level in the hierarchy and those at higher levels, respectively. Finally, `intcomm_l1_lgt1` is a handler for an MPI intercommunicator among the former and latter subcommunicators. This intercommunicator is not used for preconditioner set-up but actually is used during the iterative solution phase. It allows (one of the) first-level MPI tasks to signal (broadcast) higher-levels tasks whether the iterative process converged or not. We note that these four communicators are created outside the subroutine, in an initialization stage of our simulation software, and encapsulated in a derived data type which controls the MPI parallel environment. Any subroutine in the code needs access to this object to properly dispatch the path to be followed by each of the MPI tasks in `comm_world`. Finally, `nlev` and `tsks_per_lev` dummy arguments refer to n_ℓ and p_ℓ, for $\ell = 1, \ldots, n_\ell - 1$, respectively, while `fat_map` is an array that drives subdomain aggregation among levels. In particular, on input to the root call of this subroutine, `fat_map`(ℓ)=$task(\ell{+}1, fat(\ell, i))$ on MPI task $task(\ell, i)$. `fat_map` is generated in the preprocessing phase, when the partition of the FE mesh is performed, by (recursively) partitioning the graph of subdomains.

In lines 26–36 of Listing 1, `comm_world` is split into subcommunicators such that first-level MPI tasks provide `fat_map(1)` as the `color`, while second-level ones provide their rank identifier in `comm_l2`. The rest of the tasks do not contribute to data transfers among the first and second level and therefore provide `color=mpi_undefined`. Therefore, a subcommunicator is created for each subset of first- and second-level MPI tasks that have to exchange data. (Later on this section we will cover how to efficiently implement these data transfers.) In preparation to the recursive call, second-level and higher-level MPI tasks enter lines 39–63, while those at the first level just exit the subroutine in the search of additional first-level duties. The former set of tasks split `comm_lgt1` into `comm_l2` and `comm_lgt2` in lines 41–48, while an intercommunicator among `comm_l2` and `comm_lgt2` is created in lines 51–57. This intercommunicator is required if the coarse-grid problem is to be solved, e.g., by an $(n_\ell - 1)$-level BDDC-PCG or Richardson iterative solver (instead of by a single application of this preconditioner). Finally, the data structure corresponding to the coarse-grid problem $n_\ell - 1$-level BDDC preconditioner (i.e., `M%p_M_c`) is initialized by a recursive call to `mlbddc_init` in line 62. Note that in the recursive call the set of four communicators (`comm_lgt1`, `comm_l2`, `comm_lgt2`, `intcomm_l2_lgt2`) plays the role of (`comm_world`, `comm_l1`, `comm_lgt1`, `intcomm_l1_lgt1`), respectively, on input to the recursive call.

The subroutine in charge of the numerical set-up stage of the MLBDDC preconditioner is shown in Listing 2. It takes as input an instance `A` of the `type(par_matrix)` derived data type, and it sets up an instance `M` of the preconditioner data structure. The former derived data type internally accommodates a distributed sparse matrix, including the data describing its memory layout conformally with a nonoverlapping partition into subdomains. A first stage of preconditioner set-up encompasses lines 6–

LISTING 2

Simplified Fortran 95 MPI recursive subroutine in charge of preconditioner set-up (numeric).

```
recursive subroutine mlbddc_setup_num(A,M)
  type(par_matrix), intent(in)    :: A ! (Distributed-memory) matrix
  type(mlbddc)    , intent(inout) :: M ! MLBDDC preconditioner data structure
  real(8), allocatable :: subd_elmat(:,:) ! Subdomain contrib to coarse matrix

  if (belongs_to(M%comm_l1)) then
    call constrained_neumann_problem_setup(A,M)
    call compute_coarse_grid_basis_vectors(A,M)
    allocate(subd_elmat(M%nl_coarse,M%nl_coarse))
    call compute_subd_elmat(A,M,subd_elmat)
    call transfer_assemble_snd(M,subd_elmat)
    deallocate(subd_elmat)
    call setup_dirichlet_problem(A,M)
  else if (belongs_to(M%comm_l2)) then
    call transfer_assemble_rcv(M)
  end if

  if (M%nlev>2 .and. belongs_to(comm_lgt1)) then
    ! Recursively set-up M%p_M_c (i.e., mlbddc preconditioner for coarse
        problem)
    call mlbddc_setup_num(M%p_A_c,M%p_M_c)
  else if (M%nlev == 2 .and. belongs_to(comm_lgt1)) then
    ! Set-up M%M_c (serial solver) for M%A_c (serial coarse-grid matrix)
    call serial_solver_setup_num(M%A_c,M%M_c)
  end if
end subroutine mlbddc_setup_num

subroutine transfer_assemble_snd(M,subd_elmat)
  type(mlbddc)    , intent(inout) :: M ! MLBDDC preconditioner data structure
  real(8)         , intent(in)    :: subd_elmat(M%nl_coarse, M%nl_coarse)
  real(8), allocatable :: buf_snd(:) ! Message buffer
  ! Rest of local variable declarations go here (np, ierr, etc.)

  call mpi_comm_get_size(M%comm_l1_to_l2,np,ierr)

  allocate(buf_snd(M%max_nl_coarse**2))
  call pack(M, subd_elmat, buf_snd) ! Pack subd_elmat into buf_snd

  ! Issue (parallel) global collective
  call mpi_gather (   buf_snd, M%max_nl_coarse**2, mpi_double_precision, &
                      rcv_dum,        int_rcv_dum, mpi_double_precision, &
                    root=np-1,    M%comm_l1_to_l2, ierr)

  deallocate(buf_snd)
end subroutine transfer_assemble_snd

subroutine transfer_assemble_rcv(M)
  type(mlbddc)    , intent(inout) :: M ! MLBDDC preconditioner data structure
  real(8), allocatable :: buf_snd(:), buf_rcv(:), all_subd_elmat(:)
  ! Rest of local variable declarations go here (np, ierr, etc.)

  call mpi_comm_get_size(M%comm_l1_to_l2,np,ierr)

  allocate(buf_snd(M%nl_coarse**2),buf_rcv(np*M%max_nl_coarse**2), &
           all_subd_elmat(M%sz_all_subd_elmat))

  ! Issue (parallel) global collective
  call mpi_gather (   buf_snd, M%max_nl_coarse**2, mpi_double_precision, &
                      buf_rcv, M%max_nl_coarse**2, mpi_double_precision, &
                    root=np-1,    M%comm_l1_to_l2, ierr)

  call unpack(M, buf_rcv, all_subd_elmat) ! Unpack buf_rcv in all_subd_elmat

  ! Assemble contribs to serial (M%A_c) or distributed (M%p_A_c) coarse matrix
  if (M%nlev > 2) then
    call par_mat_ass(np, M%p_coarse, M%l_coarse, all_subd_elmat, M%p_A_c)
  else ! M%nlev == 2
    call mat_ass(np, M%p_coarse, M%l_coarse, all_subd_elmat, M%A_c)
  end if

  deallocate(buf_snd,buf_rcv,all_subd_elmat)
end subroutine transfer_assemble_rcv
```

16, and only involves first- and second-level MPI tasks. In particular, in lines 7 and 13 first-level MPI tasks set up the local solver for the constrained Neumann and Dirichlet problems, respectively, and compute the coarse-grid basis vectors in line 8. In preparation for the coarse-grid problem matrix assembly, first-level MPI tasks first compute subdomain contributions to the coarse-grid problem and store them in `subd_elmat`; see line 10. Then, first- and second-level MPI tasks enter `transfer_assemble_snd` and `transfer_assemble_snd` in lines 11 and 15, respectively. By means of these two subroutines, second-level MPI tasks gather subdomain contributions from (their corresponding) first-level MPI tasks. These contributions are then assembled in lines 64–68 of Listing 2. We note that these contributions are assembled in a `type(par_matrix)` instance `M%p_A_c`, in case of any intermediate level of the hierarchy, or in a serial (centralized) sparse matrix instance `M%A_c`, at the end of the hierarchy (see lines 65 and 67, respectively). A final stage of Listing 2 encompassing lines 18–24, recursively sets up, at any intermediate level, the $(n_\ell - 1)$-BDDC preconditioner on second- and higher-level MPI tasks (see line 20), or sets up, at the end of the hierarchy, the serial solver instance `M%M_c` for the coarse-grid problem matrix instance `M%A_c` on the last-level MPI task (see line 23).

The implementation of interlevel data transfers in Listing 2 deserves further attention. A subcommunicator for each subset of ℓ and $\ell+1$ level MPI tasks that have to exchange data was created during preconditioner initialization. Each set is defined as those i having the same $fat(\ell, i))$ together with $fat(\ell, i))$ (their master) which permits the reuse of the code implementing a two-level BDDC in which the coarse solver is solved serially in a single separated MPI task [5]. These data transfers are implemented in lines 39 and 57 in such a way that multiple independent `mpi_gather` operations are issued simultaneously on each of these subcommunicators. By means of performance analysis tools, we could confirm that this implementation approach is able to efficiently exploit the underlying network hardware parallelism of the IBM BG/Q supercomputer. In particular, these communication stages take asymptotically constant time provided ℓ and $\ell+1$ level MPI tasks are scaled proportionally (i.e., weak scaling scenario). Another technical detail is that the code in Listing 2 exploits fixed message size collectives, instead of variable message sizes ones (`mpi_gatherv`), and therefore the actual (variable size) data to be sent has to be packed to/unpacked from the (padded) message buffer on entry/exit to `mpi_gather`. Although our actual code provides both solutions, we observed in practice that fixed message size collectives lead to much better performance/scalability (despite the overhead associated to padding).

5. Numerical experiments. In this section, we study the weak scalability of the MLBDDC-PCG solver codes based on the implementation techniques presented in sections 3 and 4. As model problems, we consider the 3D Laplacian PDE on regular domains, discretized with structured (cartesian) and unstructured FE meshes (section 5.2 and 5.3, respectively), and the 3D linear elasticity problem on such domains, discretized with structured FE meshes (section 5.4). As performance metrics, we will focus in the number of PCG iterations required to converge and the total computation time. In all the experiments reported in this section, this time will include *both* preconditioner set-up and the preconditioned iterative solution of the linear system (1).

Given that this work is focused on novel implementation approaches for MLBDDC algorithms, and their evaluation on extreme scales, we just consider model problems with constant coefficients. In any case, BDDC and related DD algorithms are also robust for problems with strong coefficient jumps, as far as they are aligned with the interface among subdomains or restricted to their interior [23]. (These cases pose

no additional complication for the implementation presented herein.) In multiscale problems with highly varying coefficients that are not resolved by the subdomain partition, typical in reservoir modeling, to achieve physical-jumps-independent convergence rates is much more involved. Robustness can also be attained in this case by using MLBDDC methods with adaptive coarse spaces [34]. The adaptive definition of the coarse spaces in MLBDDC can be integrated in the overlapping framework proposed herein.

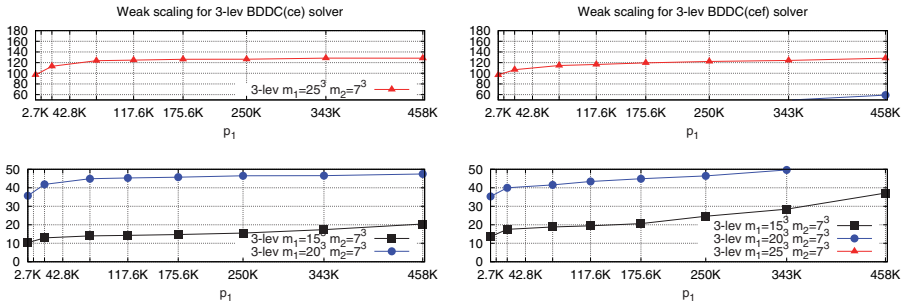
5.1. Experimental framework. The novel techniques proposed in this paper for the MLBDDC-PCG solver were implemented in FEMPAR. FEMPAR, developed by the members of the LSSC team at CIMNE, is a parallel hybrid OpenMP/MPI, object-oriented software package for the massively parallel FE simulation of multiphysics problems governed by PDEs. Among other features, it provides the basic tools for the efficient parallel distributed-memory implementation of substructuring DD solvers [4]. The parallel codes in FEMPAR heavily use standard computational kernels provided by (highly efficient vendor implementations of) BLAS and LAPACK. Besides, through proper interfaces to several third-party libraries, the local Dirichlet and constrained Neumann problems at each intermediate level of the hierarchy, and the global coarsest-grid problem at the last level, can be *exactly* solved via sparse direct solvers. In this work, we in particular explore HSL_MA87 [20], which provides a highly efficient *parallel multithreaded* DAG-based code implementation of the supernodal sparse direct Cholesky solver. The nested dissection algorithm available in METIS (v5.1.0) [21] was used as a fill-in reducing ordering for HSL_MA87. FEMPAR is released under the GNU GPL v3 license and is more than 200K lines of Fortran 95/2003/2008 code long.

All experiments reported in this section were obtained on either FERMI, located in Bologna, Italy, at CINECA, or JUQUEEN, located in Jülich, Germany, at the Jülich Supercomputing Center. They belong to the next generation of the IBM BG family of supercomputers, the so-called BG/Q supercomputers. In particular, FERMI is configured as a 10-rack system, featuring a total of 10,240 compute nodes, and JUQUEEN as a 28-rack one with a total of 28,672 compute nodes. These nodes are interconnected by an extremely low-latency five-dimensional torus interconnection network. Each compute node is equipped with a 16-core, 4-way hardware threaded core, IBM Power PC A2 processor, and 16 GB of SDRAM-DDR3 memory (i.e., 1 GB /core) and runs a lightweight proprietary CNK Linux kernel. The codes were compiled using IBM XLF Fortran compilers for BG/Q (v14.1) with recommended optimization flags. The customized MPICH2 library available on these systems was used for message-passing. The codes were linked against BLAS/LAPACK available on the single-threaded IBM ESSL library for BG/Q (v5.1) and HSL_MA87 (v2.1.1).

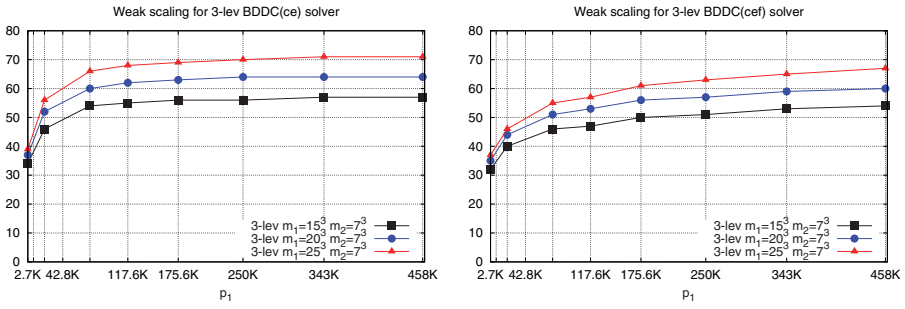
5.2. 3D Laplacian results with structured mesh discretizations. In this section we study the weak scalability of the MLBDDC-PCG solver codes in FEMPAR for the solution of the 3D Laplacian problem on the unit cube $\Omega = [0,1] \times [0,1] \times [0,1]$, with a constant forcing term $f = 1$, and homogeneous Dirichlet boundary conditions on the whole boundary $\partial\Omega$. We consider a global conforming uniform mesh (partition) of Ω into hexahedra and a trilinear FE discretization (i.e., Q1 FEs). This 3D mesh is partitioned into a cubic grid of ${p_1}^{\frac{1}{3}} \times {p_1}^{\frac{1}{3}} \times {p_1}^{\frac{1}{3}}$ cubic subdomains, with p_1 being the total number of first-level subdomains. The first-level subdomain size is ${m_1}^{\frac{1}{3}} \times {m_1}^{\frac{1}{3}} \times {m_1}^{\frac{1}{3}}$ FEs. The size of the global FE mesh is therefore equal to $(m_1 p_1)^{\frac{1}{3}} \times (m_1 p_1)^{\frac{1}{3}} \times (m_1 p_1)^{\frac{1}{3}}$.

5.2.1. Scalability under low loads on FERMI. In this section we study the weak scalability of the MLBDDC-PCG solver codes on FERMI with "small" first-level subdomain sizes, in particular, of $m_1 = 10^3$ (1K) and 15^3 (3.4K) FEs. For these values of m_1, memory consumption per core for MPI tasks at the first level is very moderate compared to the 1 GB available per core on FERMI, in particular of 19.7 and 29.9 MB, respectively. On the other hand, the time spent in the level-2 areas of Table 1 is also very moderate, posing a challenge to the effectiveness of the approach proposed in this paper (see section 3.3).

Figures 2(a) and 2(b) show, up to 64K FERMI cores, the weak scalability for the total computation time and number of PCG iterations, respectively, for the three-level (solid lines) and four-level (dashed lines) BDDC(ce) (left-hand side) and BDDC(cef) (right-hand side) solvers, with $m_1 = 10^3$ (black) and $m_1 = 15^3$ (blue). We consider two different set-ups for the three-level BDDC preconditioner, corresponding to $m_2 = 4^3$, and $m_2 = 6^3$, respectively, while a single one for the four-level BDDC, in particular, $m_2 = 3^3$, and $m_3 = 3^3$. For the three-level BDDC, we kept fixed $p_3 = 1$, and the number of subdomains in the first and second levels was scaled *proportionally* as $p_1 = m_2 p_2$ and $p_2 = k^3$, respectively, with $k = 2, 3, 4, \ldots$. For example, the five points of the plots corresponding to $m_2 = 6^3$ (see, e.g., curve with unfilled squares in the right-hand side of Figure 2(b)) are obtained with $k = 2, 3, 4, 5$, and 6, respectively. For the four-level BDDC, the number of first-, second-, and third-level subdomains were scaled proportionally as $p_1 = m_2 p_2$, $p_2 = m_3 p_3$, $p_3 = k^3$, with $k = 2, 3, 4$, so that we only have three points for the four-level BDDC plots.



(a) Total computation time (secs.).



(b) Number of PCG iterations.

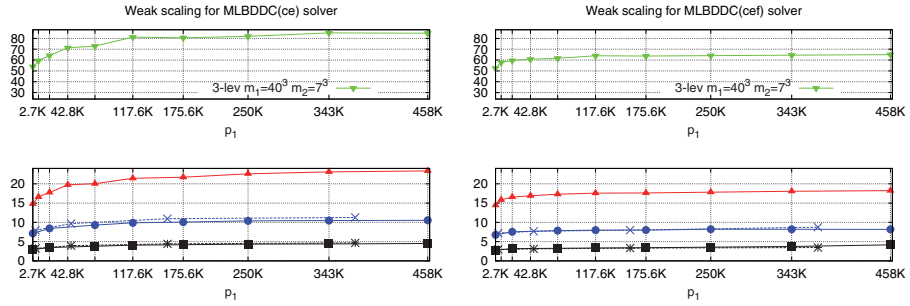
FIG. 2. *Weak scalability for the total computation time* (a) *and number of PCG iterations* (b) *of the MLBDDC(ce) (left) and MLBDDC(cef) (right) solvers in the solution of the 3D Laplacian PDE with* $m_1 = 10^3$ *and* 15^3 *FEs on FERMI.*

The results of Figure 2 clearly reveal three different scenarios for the balance achieved among the time spent in the overlapping regions in Table 1. First, for the three-level BDDC with $m_2 = 4^3$, a heavy third level is revealed (i.e., too much time spent in the coarsest-level areas of the table). For example, if we focus on the scalability of the three-level BDDC(ce) solver with $m_1 = 10^3$ on the left-hand side of Figure 2(a), we can see that scalability starts (significantly) degrading beyond $p_1 = 4$K cores; this degradation is even more severe for the three-level BDDC(cef) solver (see the right-hand side of the figure), due to a heavier coarsest-grid problem. In order to (try to) get rid of this, one can be more aggressive when aggregating first-level subdomains into second-level ones, that is, to consider a larger value of $m_2 = 6^3$, leading to a second interesting scenario in Figure 2. With such choice of m_2, we reduce the size of the coarsest-grid problem (and therefore the time spent in the coarsest-level areas of Table 1), but we increase the size of second-level subdomains. In particular, as can be observed from Figure 2(a) with $m_1 = 10^3$, up to an extent that now a heavy second level is revealed. This is confirmed by the (initially) higher computation times of the three-level BDDC solver with $m_2 = 6^3$ compared to those of $m_2 = 4^3$. We stress that although in this scenario an asymptotically constant computation time is reached (up to 46.6K cores), a lot of computational resources are wasted (and therefore parallel and energy efficiency), as first-level cores have to wait (for second-level ones) on communication stages among these two levels. The final (desirable) scenario is the one corresponding to the four-level BDDC method, which reveals a balance such that level-1 MPI tasks do not idle at communication stages among level-1 and level-2 MPI tasks. For small p_1, computational times of the four-level BDDC method are comparable to those of the three-level BDDC method with $m_2 = 4^3$ (actually smaller for the former due to a smaller $m_2 = 3^3$), as in both cases first-level duties can fully overlap coarser-grid duties (i.e., the level-2 areas in Table 1 dominate). As we scale p_1, the four-level BDDC can still maintain the desired balance (within the range of core counts studied), as it cuts down the time spent in the coarsest-level areas by the introduction of an additional level in the hierarchy.

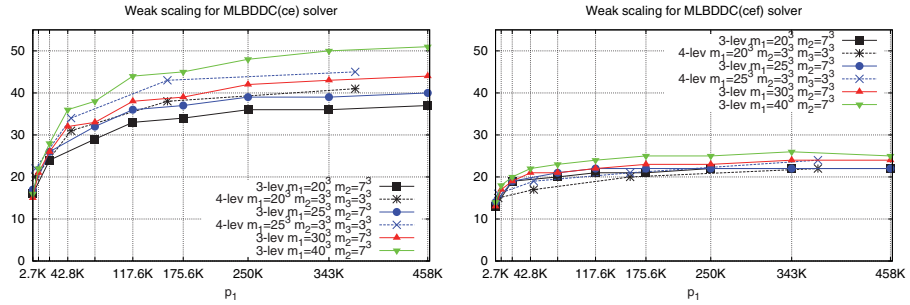
5.2.2. Scalability under medium and high loads on JUQUEEN. In this section we study the weak scalability of the MLBDDC-PCG solver codes *up to the full JUQUEEN* 28*-rack IBM BG/Q system*, with larger first-level subdomain sizes than those considered in section 5.2.1, in particular, of $m_1 = 20^3$ (8K), 25^3 (15.6K), 30^3 (27K), and 40^3 (64K) FEs. Memory consumption per first-level MPI task was in this case of 80, 146, 233, and 651 MB, respectively.

Figures 3(a) and 3(b) show, up to the full JUQUEEN system (i.e., 458,762 cores), the weak scalability for the total computation time and number of PCG iterations, respectively, for the three-level (solid lines) and four-level (dashed lines) BDDC(ce) (left-hand side) and BDDC(cef) (right-hand side) solvers, with $m_1 = 20^3$ (black), 25^3 (blue), 30^3 (red), and 40^3 (green) FEs. The results for the four-level BDDC are reported only as a reference for the first two values of m_1. We consider $m_2 = 7^3$ for the three-level BDDC preconditioner and $m_2 = 4^3$ and $m_3 = 3^3$ for the four-level BDDC one. The number of subdomains in each level were proportionally scaled as in section 5.2.1, with k being at most 6 and 11, respectively. With these choices of the coarsening ratios, we have 99.7% of the MPI tasks at the first level for the three-level case and 96.2% for four levels.

Remarkable scalability can be observed in Figure 3(a) for the three-level BDDC-PCG solver up to a full 28-rack IBM BG/Q system, in a practical demonstration of the tremendous potential of the algorithms and codes subject of study. For all values



(a) Total computation time (secs.).



(b) Number of PCG iterations.

FIG. 3. *Weak scalability for the total computation time* (a) *and number of PCG iterations* (b) *of the MLBDDC(ce) (left) and MLBDDC(cef) (right) solvers in the solution of the* 3*D Laplacian PDE with* $m_1 = 20^3, 25^3, 30^3$*, and* 40^3 *FEs on JUQUEEN.*

of m_1 considered, the time spent in level-2 regions of Table 1 is "sufficiently large" up to an extent that no idling of level-1 MPI tasks can already be obtained with a three-level method in the whole range of core counts of interest. In other words, a balance can be struck such that first-level duties fully overlap duties at coarser-grid levels. This is fulfilled for both the BDDC(ce) and the BDDC(cef) preconditioner, despite the additional work incurred in coarser-grid levels by the introduction of face constraints in the latter BDDC space. This can be readily observed in Figure 3 by the fact that the increased robustness of the BDDC(cef) (i.e., fewer PCG iterations in Figure 3(b)) immediately translates to lower computation times (compare the plots in the left and right sides of Figure 3(a)). The fact that level-1 MPI tasks do not idle combined with the fact that most of the MPI tasks are level-1 tasks shows the excellent parallel efficiency of these computations.

Another observation that confirms all this evidence is that the computational times of the four-level BDDC preconditioner are very close to those of the three-level BDDC preconditioner (and therefore the three-level BDDC method suffices to keep under control the growth of the time spent in the coarsest-grid level). We note that the degradation in the number of iterations due to the addition of a fourth level is very mild (around 10%).

5.2.3. Raising the challenge: Extra concurrency via overdecomposition. In this section we study the weak scalability of the MLBDDC-PCG solver codes under a partition of the problem into more subdomains than physical cores in-

volved in the parallel computation (i.e., under an *overdecomposition* of the problem at hand). The cores of the IBM BG/Q supercomputer require that at least two instructions are issued per cycle from different hardware threads in order to fully fill their instruction pipeline [19] (and therefore achieve peak flop performance). Therefore, using more MPI tasks per physical core it might be possible to improve the aggregated efficiency of the parallel computation.[2] We in particular explore in this section 4 MPI tasks/core. (We also performed experiments with 2 MPI tasks/core, although the results with 4 MPI tasks/core confirm a higher profit from the hardware threads in terms of aggregated efficiency.) However, the usage of this technique significantly challenges scalability of the algorithm/code/hardware combination. In particular, 4 MPI tasks/core implies a very moderate amount of memory of 256 MB/MPI task and a four-fold increase in the coarse-grid problem size to be solved at each level of the hierarchy. This challenge is, however, aligned to current and future HPC trends of having much more concurrency and less memory per core. Besides, each physical core becomes responsible for the computation and communication of four different subdomains. In order to cope with a smaller load per core, and larger coarse-grid problems, we had to consider a four-level BDDC preconditioner to account for the growth of time spent in the coarsest-grid level.

Figures 4(a) and 4(b) report the weak scalability for the total computation time and number of PCG iterations, respectively, for the four-level BDDC(ce) and BDDC(cef) solvers, with $m_1 = 10^3$ (black), 20^3 (blue), and 25^3 (red) FEs. We considered $m_2 = 4^3$ and $m_3 = 3^3$, scaling the number of first-, second-, and third-level MPI tasks proportionally as $p_1 = m_2 p_2$, $p_2 = m_3 p_3$, $p_3 = k^3$, with $k = 2, 3, 4, \ldots, 10$. The resulting number of tasks were mapped to four time fewer cores (i.e., four MPI tasks/-core). Therefore, for the largest value of k, we mapped 1.73M first-level subdomains on 448.3K cores. In this case, 98.5% of the MPI tasks are level-1 tasks.

As can be observed in Figure 4, remarkable scalability is still achieved with our approach despite the fourfold increase in the number of subdomains. In particular, full overlap of coarse-grid duties is achieved by means of interlevel-overlapping for all combinations of preconditioner and m_1, except for the four-level BDDC(cef) and $m_1 = 10^3$, where a mild degradation of scalability is achieved beyond $p_1 = 592.7$K subdomains. This observation, combined with the fact that the bulk of MPI tasks are at the first level, leads to excellent parallel efficiency. We stress that we did not actually fine-tune the values of m_2 and m_3, nor the number of levels for this experiment. A larger value of m_3 (e.g., 4^3), or a further level, could improve the situation for this particular case.

Apart from confirming remarkable scalability, we compared, on smaller test cases, the computation times of the codes using 16 and 64 MPI tasks/node (with the same number of MPI tasks/level in both cases), confirming an approximately 50% savings in aggregated efficiency by the exploitation of hardware multithreading (i.e., the computation time with 64 MPI tasks/node was approximately twice as much the one with 16 MPI tasks/node).

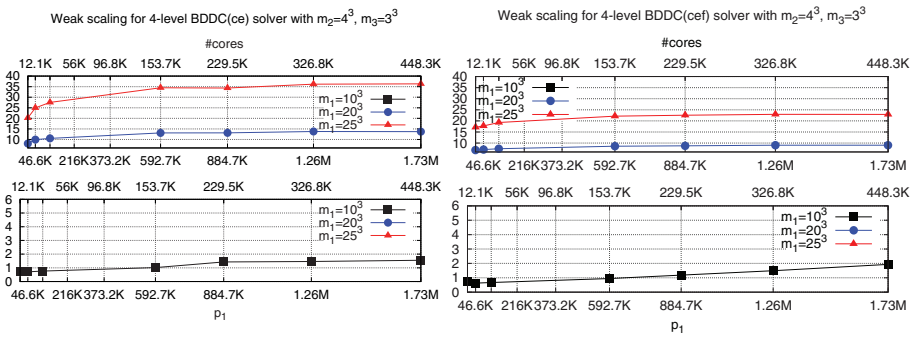
5.3. 3D Laplacian results with unstructured mesh discretizations. Unlike other algorithms like geometric MG, MLBDDC is not restricted to structured meshes and can straightforwardly be applied to general unstructured meshes. In fact,

[2]Due to restrictions inherent to the IBM BG/Q supercomputer software/hardware stack, this can be done at most up to four MPI tasks/physical core (i.e., four subdomains handled by each physical core).

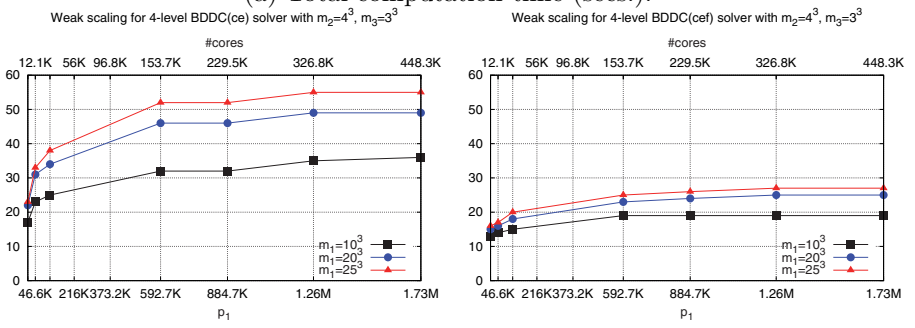
the overlapped implementation proposed in this article has been coded in the software platform FEMPAR [4] for application on unstructured meshes.

In this section we study the weak scalability of the MLBDDC-PCG solver codes in FEMPAR when applied to the 3D Laplacian problem described in section 5.2 discretized by means of unstructured meshes of tetrahedra and linear FEs (i.e., P1 FEs). A set of (increasingly refined) unstructured meshes were generated such that, when partitioned (using automatic mesh partitioning) into $p_1 = 3207$, 7440, 19,996, 46,374, 80,032, and 109,550 subdomains, a fixed load per core of approximately 262.1K tetrahedra and 43.7K vertices was obtained. Subdomain aggregation among first- and second-level subdomains was set up such that each second-level subdomain aggregates *approximately* 384 first-level subdomains. This was achieved by scaling the number of second-level subdomains as $p_2 = 8, 19, 52, 121, 208$, and 285 subdomains. With this choice of the coarsening ratio, condition (3) (that indicates that level-2 MPI tasks can be fully overlapped with level-1 MPI tasks) easily holds. For BDDC(ce), if we take $\beta_1 \approx 1/6$ and $\beta_2 \approx 4/3$ (typical in 3D tetrahedral meshes for P1 and P2 elements, respectively) the condition holds for $n_\Phi \leq 1193$, whereas n_Φ (coarse DoFs per subdomain) is around 10 in practice. Further, 99.74% of the MPI tasks are at the first level.

Figures 5(a) and 5(b) report the weak scalability for the total computation time and number of PCG iterations, respectively, for the three-level BDDC(ce) and BDDC(cef) solvers. We can observe a remarkable weak scalability in terms of both iterations and



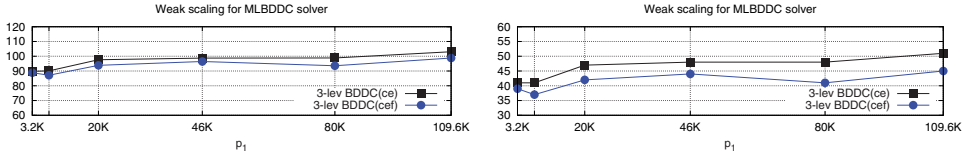
(a) Total computation time (secs.).



(b) Number of PCG iterations.

FIG. 4. *Weak scalability for the total computation time* (a) *and number of PCG iterations* (b) *of the four-level BDDC(ce) (left) and BDDC(cef) (right) solvers in the solution of the 3D Laplacian PDE with* $m_1 = 10^3, 20^3$, *and* 25^3 *FEs on JUQUEEN with four MPI tasks/core.*

total computation time, despite the sources of irregularity related to unstructured meshes. Besides, the overlapping strategy is effective on the whole span of cores studied, with slight variations in time related to slight changes in the number of iterations. As a result, these parallel executions are almost idling-free, since level-1 MPI tasks do not idle and almost all tasks are of this type. We also note that the last point in these figures involves an unstructured mesh with about 29 billion FEs and 5 billion DoFs/nodes, being one of the largest problems on unstructured meshes solved so far.



(a) Total computation time (secs.). (b) Number of PCG iterations.

FIG. 5. *Weak scalability for the total computation time* (a) *and number of PCG iterations* (b) *of the three-level BDDC(ce) and BDDC(cef) solvers in the numerical solution of the 3D Poisson PDE with unstructured mesh discretizations on JUQUEEN. A set of (increasingly refined) unstructured meshes of tetrahedra corresponding to the values of p_1 displayed in the figure was generated such that when partitioned (using automatic mesh partitioning) into p_1 subdomains, a fixed load per core of approximately $262.1K$ tetrahedra and $43.7K$ vertices was obtained.*

5.4. 3D linear elasticity results with structured mesh discretizations. In this section we evaluate the weak scalability of the MLBDDC-PCG codes when applied to a vector-valued problem, namely, the Q1 FE approximation of the (compressible) 3D linear elasticity PDE (with first and second Lamé parameters equal to 1 and $\frac{1}{10}$, respectively) on a unit cube $\Omega = [0,1] \times [0,1] \times [0,1]$, a constant forcing term $f = 1$, and homogeneous Dirichlet boundary conditions on the whole boundary. The same experiment set-up to that selected for the three-level BDDC preconditioner in section 5.2.2 is considered here, except for the size of first-level subdomains, which we now set up as $m_1 = 15^3$ (3.4K), 20^3 (8K), and 25^3 (15.6K) FEs. The largest subdomain size in this set is much smaller than the largest one considered for the 3D Laplacian problem in section 5.2.2 (i.e., $m_1 = 40^3$). The linear elasticity problem is a vector-valued problem with three unknowns per FE mesh node. This implies that for a given FE mesh, the size of the discrete operator is three times larger than that of the Laplacian problem and has nine times more nonzero entries. Indeed, with $m_1 = 25^3$ the memory consumption per first-level MPI task is 713MB, compared to 146MB for the same value of m_1 in the case of the 3D Laplacian problem.

Figures 6(a) and 6(b) report, up to the full 28-rack IBM BG/Q system, the weak scalability for the total computation time and number of PCG iterations, respectively, for the three-level BDDC(ce) and BDDC(cef) solvers, with $m_1 = 10^3$ (black), 20^3 (blue), and 25^3 (red) FEs. In these experiments, 99.7% of the MPI tasks are at the first level.

As shown in Figure 6, the approach pursued in this paper is also able to obtain remarkable scalability and parallel efficiency even for a much more computationally intensive problem such as the linear elasticity problem. Provided a "sufficiently large" m_1, i.e., 20^3 and 25^3 FEs for BDDC(ce) and BDDC(cef), respectively, an MLBDDC hierarchy equipped with three levels is already sufficient to fully overlap coarser-grid duties in the full range of cores available on the JUQUEEN supercomputer. For smaller values of m_1, some (mild) degradation of scalability is observed beyond some

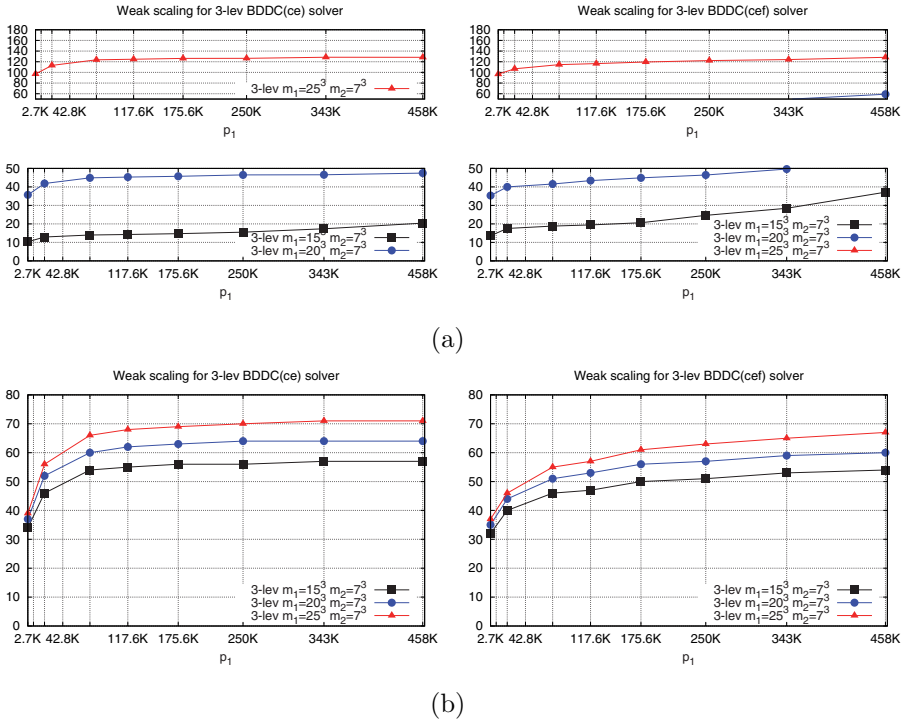


FIG. 6. *Weak scalability for the total computation time* (a) *and number of PCG iterations* (b) *of the three-level BDDC(ce) (left) and BDDC(cef) (right) solvers in the solution of the* 3*D linear elasticity PDE with* $m_1 = 10^3, 20^3$*, and* 25^3 *FEs on JUQUEEN.*

point, e.g., for 15^3, beyond $p_1 = 175.6$K first-level subdomains, which would require an additional level to keep perfect scalability.

6. Conclusions and future work. In this article, we have presented a highly scalable parallel implementation of exact MLBDDC methods, i.e., based on sparse direct solvers for the local (and coarsest) problems. The proposed implementation is based on recursion and communicator-awareness, in order to strategically deploy MPI tasks with duties at only one level and overlap interlevel tasks, based on the fact that fine and coarse corrections at every level of the MLBBDC method can be computed in parallel. The result is a bulk-asynchronous implementation with reduced synchronization among cores, which overlaps communications/computations of the different levels. Due to the recursive implementation, it can be used for an arbitrary number of levels.

This implementation leads to close to perfect weak scalability results as far as (embarrassingly parallel) level-1 duties can fully overlap tasks at higher levels. We have provided a model that helps us to choose effective coarsening ratios among cores and indicates when the overlapped strategy will start losing effectiveness. In any case, the main motivation of this work is not only to attain excellent weak scalability but also to reduce drastically the aggregated idling, via the interlevel-overlapped implementation. It results in improved parallel efficiency, energy awareness, and reduced time to solution.

A detailed scalability analysis has been carried out for the proposed implementation of the algorithms, reaching the whole JUQUEEN BG/Q, up to 458,752 cores and 1.8 million MPI tasks (subdomains), for both Laplacian and linear elasticity problems on structured meshes. We have also analyzed the scalability on unstructured meshes, with the largest problem reaching 29 billion FEs and 5 billion DoFs. These are the largest-scale problems reported so far for exact DD preconditioners. Both three-level and four-level algorithms have been explored. In order to stress the proposed implementation, (1) the coarsest-level problem has been solved in *only one processor*, i.e., no parallelization has been exploited at the last level, and (2) we have considered overdecomposition, i.e., to use a partition of the problem into more subdomains than physical cores involved in the parallel computation, reaching up to 1.8M subdomains. The results show a close to perfect weak scalability for low to moderate local problem sizes, i.e., infra-utilizing the memory resources per core. The use of more than four levels or a distributed computation of the coarsest problem can be of interest with the thousand-fold increase in the number of cores expected in the near future. Further, in order to stress the proposed setting, we have dedicated only one processor to the coarsest-level problem, but scalability can be easily improved by using a multithreaded [33] or distributed-memory [2] sparse direct solver.

Future work will include the extension of the framework to Krylov-cycling MLBDDC methods, in order to increase preconditioner robustness and keep constant the number of iterations as we go to more levels, and the development of inexact MLBDDC (hybrid AMG-BDDC) algorithms and implementations. We would also like to explore the application of the ideas in this work to additive MG preconditioners, like the BPX algorithm [10] or the new additive variants of AMG in [38]. Some additive AMG methods have recently been proved to be significantly faster than multiplicative AMG [38] by only exploiting fine-grain communication-computation overlap. Inter-level computation-computation overlap in additive AMG can lead to even further gains for such algorithms, in order to improve parallel efficiency, time-to-solution, and weak scalability.

Acknowledgement. We acknowledge PRACE for awarding us access to FERMI based in Bologna (Italy) at CINECA, and the Gauss Centre for Supercomputing (GCS) for providing computing time through the John von Neumann Institute for Computing on the GCS share of the supercomputer JUQUEEN at Jülich Supercomputing Centre (JSC). GCS is the alliance of the three national supercomputing centres HLRS (Universität Stuttgart), JSC, and LRZ (Bayerische Akademie der Wissenschaften), funded by the German Federal Ministry of Education and Research (BMBF) and the German State Ministries for Research of Baden-Württemberg (MWK), Bayern (StMWFK) and Nordrhein-Westfalen (MIWF). We gratefully acknowledge JSC staff in general, and Dirk Broemmel in particular, for their support in porting/debugging FEMPAR and its dependencies to/on JUQUEEN.

REFERENCES

[1] *Report on the Workshop on Extreme-Scale Solvers: Transition to Future Architectures*, Tech. report, U.S. Department of Energy, 2012.

[2] P. R. Amestoy, I. S. Duff, and J.-Y. L'Excellent, *Multifrontal parallel distributed symmetric and unsymmetric solvers*, Comput. Methods Appl. Mech. Engrg., 184 (2000), pp. 501–520.

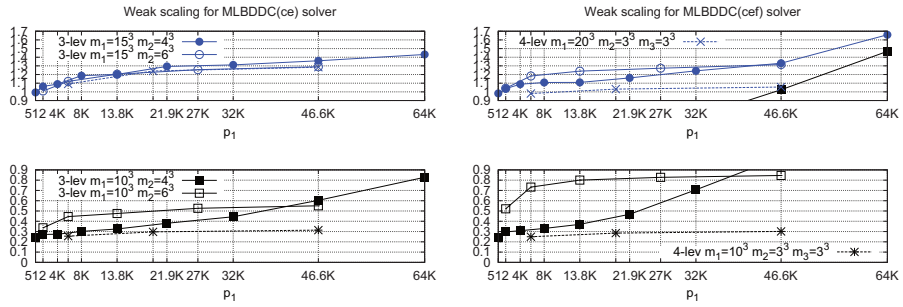
[3] S. Badia, A. F. Martín, and J. Principe, *Enhanced balancing Neumann-Neumann preconditioning in computational fluid and solid mechanics*, Internat. J. Numer. Methods Engrg., 96 (2013), pp. 203–230.

[4] S. Badia, A. F. Martín, and J. Principe, *Implementation and scalability analysis of balancing domain decomposition methods*, Arch. Comput. Methods Engrg., 20 (2013), pp. 239–262.
[5] S. Badia, A. F. Martín, and J. Principe, *A highly scalable parallel implementation of balancing domain decomposition by constraints*, SIAM J. Sci. Comput., (2014), pp. C190–C218.
[6] S. Badia, A. F. Martín, and J. Principe, *On the scalability of inexact balancing domain decomposition by constraints with overlapped coarse/fine corrections*, Parallel Comput., 50 (2015), pp. 1–24.
[7] A. H. Baker, T. Gamblin, M. Schulz, and U. M. Yang, *Challenges of scaling algebraic multigrid across modern multicore architectures*, in Proceedings of the International Parallel Distributed Processing Symposium, IEEE, 2011, pp. 275–286.
[8] S. Balay, J. Brown, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. Curfman McInnes, B. F. Smith, and H. Zhang, *PETSc*, `http://www.mcs.anl.gov/petsc` (2012).
[9] P. Bastian, G. Wittum, and W. Hackbusch, *Additive and multiplicative multi-grid — A comparison*, Computing, 60 (1998), pp. 345–364.
[10] J. H. Bramble, J. E. Pasciak, and J. Xu, *Parallel multilevel preconditioners*, Math. Comp., 55 (1990), pp. 1–22.
[11] S. C. Brenner and R. Scott, *The Mathematical Theory of Finite Element Methods*, 3rd ed., Springer, New York, 2010.
[12] D. Brömmel and P. Gibbon, *High-Q Club The highest scaling Codes on JUQUEEN*, Innov. Supercomput. Deutschland, 11 (2013), pp. 106–107.
[13] E. Chow, R. D. Falgout, J. J. Hu, R. S. Tuminaro, and U. M. Yang, *A survey of parallelization techniques for multigrid solvers*, Parallel Process. Sci. Comput., 20 (2006), pp. 179–201.
[14] T. A. Davis, *Direct Methods for Sparse Linear Systems*, Vol. 2, SIAM, Philadelphia, 2006.
[15] C. R. Dohrmann, *A preconditioner for substructuring based on constrained energy minimization*, SIAM J. Sci. Comput., 25 (2003), pp. 246–258.
[16] C. R. Dohrmann, *An approximate BDDC preconditioner*, Numer. Linear Algebra Appl., 14 (2007), pp. 149–168.
[17] C. Farhat, K. Pierson, and M. Lesoinne, *The second generation FETI methods and their application to the parallel solution of large-scale linear and geometrically non-linear structural analysis problems*, Comput. Methods Appl. Mech. Engrg., 184 (2000), pp. 333–374.
[18] V. Hapla, D. Horak, and M. Merta, *Use of direct solvers in TFETI massively parallel implementation*, in Applied Parallel and Scientific Computing, Lecture Notes in Comput. Sci. 7782, Springer, Berlin, 2013, pp. 192–205.
[19] R. A. Haring, M. Ohmacht, T. W. Fox, M. K. Gschwind, D. L. Satterfield, K. Sugavanam, P. W. Coteus, P. Heidelberger, M. A. Blumrich, R. W. Wisniewski, A. Gara, G. L.-T. Chiu, P. A. Boyle, N. H. Chist, and Changhoan Kim, *The IBM blue Gene/Q compute chip*, IEEE Micro., 32 (2012), pp. 48–60.
[20] J. Hogg, J. Reid, and J. Scott, *Design of a multicore sparse Cholesky factorization using DAGs*, SIAM J. Sci. Comput., 32 (2010), pp. 3627–3649.
[21] G. Karypis, *A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices. Version* 5.1.0, Tech. report, Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN, 2013; `http://glaros.dtc.umn.edu/gkhome/fetch/sw/metis/manual.pdf`.
[22] A. Klawonn and O. Rheinbach, *Highly scalable parallel domain decomposition methods with an application to biomechanics*, ZAMM Z. Angew. Math. Mech., 90 (2010), pp. 5–32.
[23] A. Klawonn, O. Widlund, and M. Dryja, *Dual-primal FETI methods for three-dimensional elliptic problems with heterogeneous coefficients*, SIAM J. Numer. Anal., 40 (2002), pp. 159–179.
[24] A. Klawonn and O. B. Widlund, *A domain decomposition method with lagrange multipliers and inexact solvers for linear elasticity*, SIAM J. Sci. Comput., 22 (2000), pp. 1199–1219.
[25] P. T. Lin, J. N. Shadid, M. Sala, R. S. Tuminaro, G. L. Hennigan, and R. J. Hoekstra, *Performance of a parallel algebraic multilevel preconditioner for stabilized finite element semiconductor device modeling*, J. Comput. Phys., 228 (2009), pp. 6250–6267.
[26] J. Mandel, *Balancing domain decomposition*, Commun. Numer. Methods Engrg., 9 (1993), pp. 233–241.
[27] J. Mandel and C. R. Dohrmann, *Convergence of a balancing domain decomposition by constraints and energy minimization*, Numer. Linear Algebra Appl., 10 (2003), pp. 639–659.
[28] J. Mandel, C. R. Dohrmann, and R. Tezaur, *An algebraic theory for primal and dual substructuring methods by constraints*, Appl. Numer. Math., 54 (2005), pp. 167–193.
[29] J. Mandel, B. Sousedík, and C. Dohrmann, *Multispace and multilevel BDDC*, Computing, 83 (2008), pp. 55–85.

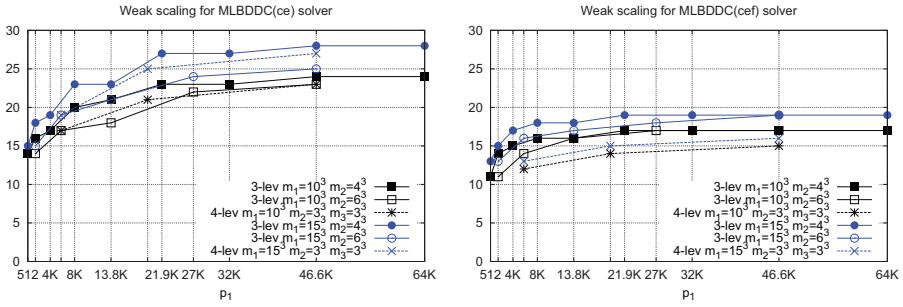
[30] Y. Notay and A. Napov, *A massively parallel solver for discrete Poisson-like problems*, J. Comput. Phys., 281 (2015), pp. 237–250.
[31] O. Rheinbach, *Parallel iterative substructuring in structural mechanics*, Arch. Comput. Methods Eng., 16 (2009), pp. 425–463.
[32] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed., SIAM, Philadelphia, 2003.
[33] O. Schenk and K. Gärtner, *On fast factorization pivoting methods for sparse symmetric indefinite systems*, Electron. Trans. Numer. Anal., 23 (2006), pp. 158–179.
[34] B. Sousedík, J. Šístek, and J. Mandel, *Adaptive-multilevel BDDC and its parallel implementation*, Computing, 95 (2013), pp. 1087–1119.
[35] K. Stüben, *A review of algebraic multigrid*, J. Comput. Appl. Math., 128 (2001), pp. 281–309.
[36] A. Toselli and O. Widlund, *Domain Decomposition Methods—Algorithms and Theory*, Springer-Verlag, Berlin, 2005.
[37] X. Tu, *Three-level BDDC in three dimensions*, SIAM J. Sci. Comput., 29 (2007), pp. 1759–1780.
[38] P. S. Vassilevski and U. M. Yang, *Reducing communication in algebraic multigrid using additive variants*, Numer. Linear Algebra Appl., 21 (2014), pp. 275–296.
[39] J. Šístek, M. Čertíková, P. Burda, and J. Novotný, *Face-based selection of corners in 3D substructuring*, Math. Comput. Simul., 82 (2012), pp. 1799–1811.

ERRATUM

Figure 2 is incorrect. The correct figure is as follows.



(a) Total computation time (secs.).



(b) Number of PCG iterations.

FIG. 2. *Weak scalability for the total computation time* (a) *and number of PCG iterations* (b) *of the MLBDDC(ce) (left) and MLBDDC(cef) (right) solvers in the solution of the 3D Laplacian PDE with* $m_1 = 10^3$ *and* 15^3 *FEs on FERMI.*