SIAM J. SCI. COMPUT.
Vol. 00, No. 0, pp. 0000–0000

A HIGHLY SCALABLE PARALLEL IMPLEMENTATION OF BALANCING DOMAIN DECOMPOSITION BY CONSTRAINTS*

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

Abstract. In this work we propose a novel parallelization approach of two-level balancing domain decomposition by constraints preconditioning based on overlapping of fine-grid and coarse-grid duties in time. The global set of MPI tasks is split into those that have fine-grid duties and those that have coarse-grid duties, and the different computations and communications in the algorithm are then re-scheduled and mapped in such a way that the maximum degree of overlapping is achieved while preserving data dependencies among them. In many ranges of interest, the extra cost associated to the coarse-grid problem can be fully masked by fine-grid related computations (which are embarrassingly parallel). Apart from discussing code implementation details, the paper also presents a comprehensive set of numerical experiments, that includes weak scalability analyses with structured and unstructured meshes for the 3D Poisson and linear elasticity problems on a pair of state-of-the-art multicore-based distributed-memory machines. This experimental study reveals remarkable weak scalability in the solution of problems with thousands of millions of unknowns on several tens of thousands of computational cores.

Key words. Domain decomposition, coarse-grid correction, BDDC, parallelization, overlapping, MPMD, scalability

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

1. Introduction.

Scientific phenomena governed by partial differential equations (PDEs) can be approximated by finite element (FE) methods, which results in a sparse linear system of equations to be solved via numerical linear algebra. The ever increasing demand of reality in the simulation of the complex scientific and engineering three-dimensional (3D) problems faced nowadays involves the solution of very large sparse linear systems with several hundreds and even thousands of millions of equations/unknowns. The solution of these systems in a moderate time requires the vast amount of computational resources provided by current multicore-based distributed-memory machines. It is therefore essential to design parallel algorithms able to take profit of their underlying architecture.

Domain decomposition (DD) preconditioning of Krylov iterative solvers provides a natural framework for the development of fast parallel solvers tailored for distributed-memory machines, as it has by construction the desirable design principle of maximizing local computations while minimizing interprocessor communication. One-level DD preconditioners, such as the Neumann-Neumann preconditioner [29], are highly parallel as they only require the solution of local problems and communication among neighboring subdomains, but the condition number of the preconditioned system matrix grows with the number of subdomains (processors) and/or the global size of the

†Centre Internacional de Mètodes Numèrics en Enginyeria (CIMNE), Parc Mediterrani de la Tecnologia, UPC, Esteve Terradas 5, 08860 Castelldefels, Spain ({sbadia,amartin,principe}@cimne.upc.edu).

‡Universitat Politècnica de Catalunya, Jordi Girona 1-3, Edifici C1, 08034 Barcelona, Spain.

§A. F. Martín was also partially funded by the UPC postdoctoral grants under the programme "BKC5-Atracció i Fidelització de talent al BKC"

*This work has been funded by the European Research Council under the FP7 Programme Ideas through the Starting Grant No. 258443 - COMFUS: Computational Methods for Fusion Technology. We acknowledge PRACE for awarding us access to resource CURIE based in France at TGCC. The support of Yohan Lee-tin-yien and Marie Cadennes from TGCC, France to the technical work is gratefully acknowledged. We also thank IFERC-CSC, Japan, for granting access to HELIOS and providing technical support.

linear system. As a result, these algorithms are not scalable since the number of preconditioned Krylov iterations increases with the number of cores.

Two-level DD preconditioners combine local and global corrections (in an additive or in a multiplicative fashion) in order to achieve quasi-optimal condition number bounds, i.e., independent of the number of subdomains and global problem size, for second-order coercive problems. The global correction involves the solution of a "small" (relative to the original linear system) coarse-grid problem that couples all the subdomains. In the frame of non-overlapping DD methods (also referred to as iterative sub-structuring or Schur complement methods), we find the Balancing Neumann-Neumann preconditioner [21] (BNN), the Balancing DD by Constraints preconditioner [9] (BDDC) and the family of FETI preconditioners [10, 11]. These methods are quasi-optimal (algorithmically scalable) with a poly-logarithmic expression of the condition number of the preconditioned system $\kappa = 1 + \log^2(\frac{H}{h})$, where h and H are, respectively, the mesh and subdomain characteristic sizes, $(\frac{H}{h})^d$ is the size of the local problems and d is the space dimension. Consequently, in weak scaling [13] scenarios (i.e., $\frac{H}{h}$ fixed), the number of iterations of the preconditioned conjugate gradient (PCG) solver is (asymptotically) independent of the number of processors.

However, weak scalability is endangered in practice, since the coarse solver size increases (at best) linearly with the number of subdomains, which results in increasing parallel overheads (i.e., loss of efficiency) with the number of processors. For large-scale problems, the coarse problem rapidly becomes the bottleneck of the algorithm. For the BDDC preconditioner, a strategy to alleviate this situation is the use of a multilevel algorithm [23] where the coarse problem is only approximated. This way, the CPU cost of the coarse problem is reduced but the condition number bound increases exponentially with the number of levels [23].[1] Alternatively, one cycle of the AMG solver in BoomerAMG [18] has been used as inexact coarse solver in [20, 24] for FETI-DP methods. Another approach, which does not spoil the preconditioner robustness, is to use a distributed-memory solver for the coarse problem, e.g., the use of the MPI-distributed sparse direct solver MUMPS [1] for BDDC [28] and FETI-DP methods [16].

In any case, as far as we know, the coarse component is serialized in all the efficient implementations of two-level DD preconditioners so far (see, e.g., [5, 6, 16, 17, 20, 24, 28]), i.e., no fine duties are performed in the meantime, since the subset of processors with coarse duties have also fine duties. As a result, the computational time associated to the application of the preconditioner and the memory usage of cores with coarse duties irremediably increases with the number of processors, specially when sparse direct solvers are used. Current trends in distributed-memory platforms will make this situation even more dramatic.

However, the BDDC preconditioner is defined in such a way that the coarse and fine correction spaces are orthogonal with respect to the energy norm and there is no algorithmic limitation for the serialized computation of the coarse solver.[2] We consider that this key algorithmic property has not been properly exploited in current implementations of BDDC and other additive Schwarz DD preconditioners. *This is the objective of this work.* The contributions are the following:

- Novel parallelization approach of BDDC based on overlapped fine-grid and

[1] It is unclear from the detailed numerical experiments in [28] the benefit of the multilevel extension (see, e.g., Tables 4, 7, and 8 in [28]).

[2] This property is not true for multiplicative Schwarz preconditioners, e.g., the classical BNN method. (See [3] for an additive version of the BNN method.)

coarse-grid duties in time. These novel techniques are designed in order to tackle the bottleneck associated with the solution of the coarse-grid problem.
- A comprehensive discussion of how these novel techniques are exploited in order to reach maximum performance benefit, including code implementation details for a MPMD (Multiple Program Multiple Data) parallel execution mode.
- Weak scalability study of the new parallelization approach for the 3D Poisson and linear elasticity problems up to 27K cores on structured meshes and up to 8K cores on unstructured meshes on a pair of state-of-the-art multicore-based distributed-memory machines (HELIOS and CURIE).

This work is structured as follows. Section 2 is devoted to non-overlapping DD and the BDDC preconditioner. In Section 3, we design a highly scalable parallel distributed-memory implementation of the BDDC algorithm, which overlaps fine and coarse computations. In Section 4, we report a comprehensive set of numerical experiments that includes weak scalability analyses with structured and unstructured meshes in 3D for Poisson and linear elasticity problems. Finally, in Section 5, we draw some conclusions and define future lines of work.

2. Balancing Domain Decomposition.

2.1. General framework. Let us consider a bounded polyhedral domain $\Omega \subset \mathbb{R}^d$ with $d = 2, 3$ and a uniform FE partition (mesh) $\mathcal{T}$ of $\overline{\Omega}$ with characteristic size h. Further, we consider a partition of the global mesh $\mathcal{T}$ into local meshes $\{\mathcal{T}_i : i = 1, \ldots, n_{\text{sbd}}\}$, which induces a domain decomposition of Ω into subdomains $\{\Omega_i : i = 1, \ldots, n_{\text{sbd}}\}$ (of characteristic size H) such that $\mathcal{T}_i$ is a conforming mesh of Ω_i. We denote the number of nodes in $\mathcal{T}$ and $\mathcal{T}_i$ as n and n_i, respectively. The interface of Ω_i is defined as $\Gamma_i = \partial\Omega_i \setminus \partial\Omega$ and the whole interface (skeleton) of the domain decomposition is $\Gamma = \bigcup_{i=1}^{n_{\text{sbd}}} \Gamma_i$.

As model problem, let us consider the Poisson problem on a domain $\Omega \subset \mathbb{R}^d$, with homogeneous Dirichlet boundary conditions on $\partial\Omega$, where $d = 2, 3$ is the number of space dimensions. We also consider a uniform FE partition (mesh) $\mathcal{T}$ of Ω with characteristic size h. We are interested in solving the set of linear equations

$$Ax = f, \tag{2.1}$$

which arises from the Galerkin FE discretization of the continuous problem corresponding to $\mathcal{T}$.

The set of nodes of $\mathcal{T}$ that belong to Γ (resp. Γ_i) is denoted by Γ_h (resp. Γ_h^i), and we denote its cardinality by n_Γ (resp. n_Γ^i). This partition of the domain into non-overlapping subdomains induces the following block reordered structure of (2.1):

$$\begin{bmatrix} A_{II} & A_{I\Gamma} \\ A_{\Gamma I} & A_{\Gamma\Gamma} \end{bmatrix} \begin{bmatrix} x_I \\ x_\Gamma \end{bmatrix} = \begin{bmatrix} f_I \\ f_\Gamma \end{bmatrix}, \tag{2.2}$$

where x_Γ contains the unknowns corresponding to the nodes in Γ_h and x_I the remaining ones, associated with subdomain interiors. Besides, A_{II} presents a block diagonal structure (and therefore very amenable to parallelization), i.e.,

$$A_{II} = \text{diag}\left(A_{II}^{(1)}, A_{II}^{(2)}, \ldots, A_{II}^{(n_{\text{sbd}})}\right),$$

where $A_{II}^{(i)}$ is the local matrix which represents the coupling of internal unknowns at subdomain i. After the static condensation of x_I from (2.2), this linear system is

reduced to the Schur complement problem

$$Sx_\Gamma = g, \qquad \text{where} \quad S = A_{\Gamma\Gamma} - A_{\Gamma I}A_{II}^{-1}A_{I\Gamma}, \quad \text{and} \quad g = f_\Gamma - A_{\Gamma I}A_{II}^{-1}f_I. \tag{2.3}$$

The vector space of interface nodal values in Γ_h is denoted by $\widehat{\mathbb{V}}$; clearly, $\widehat{\mathbb{V}}$ is equivalent to $\mathbb{R}^{n_\Gamma}$. We also define the local space $\mathbb{V}_i$ of interface nodal values on Γ_h^i, which is equivalent to $\mathbb{R}^{n_\Gamma^i}$.[3] Clearly, the Schur complement matrix $S : \widehat{\mathbb{V}} \times \widehat{\mathbb{V}} \to \mathbb{R}$.

System (2.3) can be written as the assembly (sum) of local Schur complement matrices and right hand side vectors as

$$S = \sum_{i=1}^{n_{\text{sbd}}} R_i^t S^{(i)} R_i, \quad g = \sum_{i=1}^{n_{\text{sbd}}} R_i^t g^{(i)}, \tag{2.4}$$

where $R_i : \widehat{\mathbb{V}} \to \mathbb{V}_i$ is the restriction operator and R_i^t its transpose. The former applied to a vector $y \in \widehat{\mathbb{V}}$ gives the vector of local values $y^{(i)} = R_i y \in \mathbb{V}_i$, while the latter applied to a local vector gives a global vector (filled with zeros for nodes not belonging to subdomain i). The local Schur complement $S^{(i)}$ and local right hand side vector $g^{(i)}$ are defined as:

$$S^{(i)} = A_{\Gamma\Gamma}^{(i)} - A_{\Gamma I}^{(i)}(A_{II}^{(i)})^{-1}A_{I\Gamma}^{(i)}, \quad g^{(i)} = f_\Gamma^{(i)} - A_{\Gamma I}^{(i)}(A_{II}^{(i)})^{-1}f_I^{(i)}. \tag{2.5}$$

Let $\mathbb{V} = \mathbb{V}_1 \times \ldots \times \mathbb{V}_{n_{\text{sbd}}}$. We denote by $s^{(i)}$ the i-th component of $s \in \mathbb{V}$, i.e., its restriction to Ω_i. By definition, the cardinality of this space is $n_{\text{r}} = \sum_{i=1}^{n_{\text{sbd}}} n_\Gamma^i$, which is equivalent to $\mathbb{R}^{n_{\text{r}}}$.[4] We denote by $S_{\text{sub}} : \mathbb{V} \times \mathbb{V} \to \mathbb{R}$ the sub-assembled cartesian product Schur complement matrix composed by $S^{(i)}$, i.e.,

$$S_{\text{sub}} = \text{diag}(S^{(1)}, S^{(2)}, \ldots, S^{(n_{\text{sbd}})}).$$

It is possible to obtain an *averaged* global vector $z \in \widehat{\mathbb{V}}$ from $s \in \mathbb{V}$ as

$$z = \sum_{i=1}^{n_{\text{sbd}}} I_i s^{(i)} = Is, \tag{2.6}$$

where $I_i = R_i^t W_i : \mathbb{V}_i \to \widehat{\mathbb{V}}$ is the local injection operator, and W_i is a diagonal weighting matrix such that

$$z = \sum_{i=1}^{n_{\text{sbd}}} I_i R_i z, \quad \text{for any } z \in \widehat{\mathbb{V}}. \tag{2.7}$$

If we denote by $n(p)$ the number of subdomains sharing node p, W_i can be defined as the diagonal matrix $(W_i)_{pp} = 1/n(p)$, $p = 1, \ldots, n_\Gamma^i$, although more elaborated expressions must be considered for discontinuous physical properties [29]. I is the cartesian product matrix composed by I_i.

At this point, we have observed that the original (global) problem can be recasted as an interface problem (2.3). However, the Schur complement enforces continuity of

[3] The spaces $\widehat{\mathbb{V}}$ and $\mathbb{V}_i$ can also be understood in a functional setting as the global and local spaces of discrete harmonic functions (see [7]).

[4] In a functional setting, functions in $\widehat{\mathbb{V}}$ are uni-valued on Γ. On the contrary, since nodes in Γ_h are replicated, functions in $\mathbb{V}$ can take different values at different subdomains. As in [29], $\widehat{\cdot}$ is used to denote uni-valued functions on Γ.

all interface nodes Γ_h. The assembly of its corresponding matrix S (of size n_Γ) in a set of processors and its subsequent factorization via a (distributed-memory) direct solver is unacceptable for large core counts. On the other hand, the Schur complement matrix is poorly conditioned; its condition number increases with the size of the global problem and number of subdomains (see Section 2.3). Thus, the definition of a preconditioner for the Schur complement matrix is required to obtain a scalable linear solver.

2.2. Balancing DD by constraints preconditioner. The BDDC preconditioner is a two-level DD preconditioner where a local fine-grid and a global coarse-grid correction (that couples all subdomains and makes the preconditioner both scalable and optimal) are combined. The idea of the BDDC preconditioner is to approximate the original FE problem by another one in which we relax the continuity conditions, drastically reducing the size of the modified Schur complement. Thus, the BDDC Schur complement can now be assembled and solved, e.g., by a sparse direct solver.

The construction of the BDDC preconditioner is based on a topological classification of the nodes on the interface into objects, which can be corners, or members of edges or faces. This classification can be properly stated in the general case of unstructured meshes and automatic mesh partitioners; see, e.g., the definition in [4]. Next, we associate to some (or all) of these objects a *coarse DoF*. The three most 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.

For a given $x \in \mathbb{V}$, the coarse DoF is the value of the function on the corner or the mean value of the function on the edge/face (see [4]). We define the BDDC space $\widetilde{\mathbb{V}}$ as the subspace of vectors in $\mathbb{V}$ that are continuous on coarse DoFs; we note that $\widehat{\mathbb{V}} \subset \widetilde{\mathbb{V}} \subset \mathbb{V}$. Let us denote by $S_{\text{BDDC}} : \widetilde{\mathbb{V}} \times \widetilde{\mathbb{V}} \to \mathbb{R}$ the Schur complement related to $\widetilde{\mathbb{V}}$, i.e., the restriction of S_{sub} to $\widetilde{\mathbb{V}}$. The BDDC preconditioner is $M^{-1}_{\text{BDDC}} = I S^{-1}_{\text{BDDC}} I^t$. *We assume that this preconditioner is nonsingular.* The well-posedness of this preconditioner depends on the definition of the coarse DoFs; there are existing mechanisms that modify the definition of objects in order to fulfill this assumption (see [9, 31]).

Some additional work has to be performed in order to end up with an implementation of this problem that is well-suited for distributed-memory machines. Let us define n_{cts} as the number of coarse DoFs of the BDDC space and n^i_{cts} as its local counterpart. We define the local matrix of constraints $C_i \in \mathbb{R}^{n^i_{\text{cts}} \times n^i_\Gamma}$ such that given a local vector of unknowns provides its local coarse DoFs values; we refer to [4] for a detailed implementation of C_i. Next, we consider the following decomposition of the BDDC space $\widetilde{\mathbb{V}}$ into a *fine space* $\widetilde{\mathbb{V}}_{\text{F}}$ of vectors that vanish on coarse DoFs and the $S_{\text{sub}}-$orthogonal complement $\widetilde{\mathbb{V}}_{\text{C}}$, denoted as the *coarse space*. Since fine and coarse spaces are $S_{\text{sub}}-$orthogonal by definition, fine and coarse correction can be computed in parallel.

Since the fine space $\widetilde{\mathbb{V}}_{\text{F}}$ vanishes on coarse DoFs (which are the only DoFs that involve continuity among subdomains), the fine correction involves local problems. The local, fine-grid preconditioner in the BDDC method is defined as

$$M^{-1}_{\text{F}} = \sum_{i=1}^{n_{\text{sbd}}} I_i (S^{(i)}_{\text{F}})^{-1} I_i^t,$$

where $(S^{(i)}_{\text{F}})^{-1}$ is a "constrained" inverse of the local Schur complement $S^{(i)}$. The

application of $(S_{\mathrm{F}}^{(i)})^{-1}$ to a vector, denoted as $s_{\mathrm{F}}^{(i)} = (S_{\mathrm{F}}^{(i)})^{-1} r^{(i)}$, involves the solution of the following (constrained) linear system

$$\begin{bmatrix} A_{II}^{(i)} & A_{I\Gamma}^{(i)} & 0 \\ A_{\Gamma I}^{(i)} & A_{\Gamma\Gamma}^{(i)} & C_i^t \\ 0 & C_i & 0 \end{bmatrix} \begin{bmatrix} t \\ s_{\mathrm{F}}^{(i)} \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ r^{(i)} \\ 0 \end{bmatrix}.$$

We can easily check that M_{F}^{-1} is the inverse of the Schur complement related to $\widetilde{\mathbb{V}}_{\mathrm{F}}$. The coarse space $\widetilde{\mathbb{V}}_{\mathrm{C}} \subset \mathbb{V}$ is built as

$$\widetilde{\mathbb{V}}_{\mathrm{C}} = \mathrm{span}\{\Phi_1, \Phi_2, \ldots, \Phi_{n_{\mathrm{cts}}}\},$$

where every coarse function is associated to a coarse DoF. We denote by Φ the matrix with columns Φ_i. The coarse basis Φ (the matrix with columns Φ_i) is the solution of a multiple right-hand side global system. Fortunately, since the values on the coarse DoFs are prescribed and the rest of DoFs are local, the coarse space can also be computed via (parallel) local constrained Neumann problems, i.e.,

$$\begin{bmatrix} A_{II}^{(i)} & A_{I\Gamma}^{(i)} & 0 \\ A_{\Gamma I}^{(i)} & A_{\Gamma\Gamma}^{(i)} & C_i^t \\ 0 & C_i & 0 \end{bmatrix} \begin{bmatrix} \Phi_I^{(i)} \\ \Phi_\Gamma^{(i)} \\ \Lambda^{(i)} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \mathrm{Id}^{(i)} \end{bmatrix},$$

where $\mathrm{Id}^{(i)}$ is the identity matrix of size n_{cts}^i. Let us note that any function Φ_i is associated to an object and its support is the set of subdomains that share this object. Thus, at every subdomain we only compute the non-zero restrictions, i.e., the coarse space basis functions related to local coarse DoFs. We compute the coarse matrix S_0 as

$$S_0 = \sum_{i=1}^{n_{\mathrm{sbd}}} R_{0i}^t {\Phi^{(i)}}^t A^{(i)} \Phi^{(i)} R_{0i}.$$

where R_{0i} is the coarse matrix assembly operator, i.e. the local-to-global correspondence for coarse DoFs. The subdomain contributions ${\Phi^{(i)}}^t A^{(i)} \Phi^{(i)}$ can readily be computed (in parallel) and assembled, e.g., in one processor. Once S_0 is assembled, the coarse correction can readily be computed, e.g., by a sparse direct or iterative solver. The coarse residual $\Phi^t I^t r$ is computed analogously (see [4]).

The final preconditioner can be written as the combination of coarse and fine-grid contributions as

$$M_{\mathrm{BDDC}}^{-1} = M_{\mathrm{C}}^{-1} + M_{\mathrm{F}}^{-1},$$

where $M_{\mathrm{C}}^{-1} = \Phi S_0^{-1} \Phi^t$. We observe that *the S_{sub}-orthogonality between coarse and fine BDDC spaces allows one to overlap coarse and fine tasks in the application of the BDDC preconditioner*, which corresponds to the overlapping area #3 in the highly scalable implementation of Section 3.3.

2.3. Condition numbers. Assuming a one-to-one mapping between subdomains and processors, we denote the number of processors $P = n_{\mathrm{sbd}}$. The condition number of the global matrix A is $\mathcal{O}(n^{\frac{2}{d}})$ whereas that of the Schur complement S is $\mathcal{O}(n^{\frac{1}{d}} P^{\frac{1}{d}})$ [29]. It is well known that the number of iterations required by the PCG

Krylov solver is $\mathcal{O}\left(\sqrt{\kappa}\right)$, where κ is the condition number of the preconditioned operator [25]. Therefore, the estimated number of PCG iterations is $\mathcal{O}(n^{\frac{1}{d}})$ and $\mathcal{O}(n^{\frac{1}{2d}}P^{\frac{1}{2d}})$ when it is applied to (2.1) and (2.3), respectively. Although the number of PCG iterations is certainly cut down by the re-statement of the problem on the interface by the DD approach (since $n \gg P$ for practical ranges of application), there is a lot of margin for improvement via preconditioning.

On the other hand, when introducing the BDDC preconditioner, the condition number can be bounded by (cf. [22])

$$\kappa\left(M_{\mathrm{BDDC}}^{-1}S\right) \leq C\left[1+\frac{1}{d^2}\log^2\left(\frac{n}{P}\right)\right],$$

which results in a constant number of PCG iterations for weak scaling analysis, i.e., increasing P and n but keeping constant its ratio (load per processor). This bound does not apply in 3D for BDDC(c). Instead,

$$\kappa\left(M_{\mathrm{BDDC}}^{-1}S\right) \leq C\frac{n}{P}\left[1+\frac{1}{d^2}\log^2\left(\frac{n}{P}\right)\right]. \tag{2.8}$$

This bound is certainly worse than the previous one, since n/P (local problem size) can easily be of the order of 10^4. The BDDC(c) method in 3D requires a larger number of iterations to get convergence but the method is still weakly scalable.

3. A highly scalable distributed-memory implementation. In this section we cover in detail a highly scalable distributed-memory implementation of the BDDC-PCG parallel solver. The section is structured as follows. In Section 3.1 we cover the basic building blocks that make up a parallel distributed-memory implementation of the algorithm subject of study. In Section 3.2, we discuss several bottlenecks of the typical implementation approach of two-level DD methods and the rationale behind the techniques we are proposing to tackle them. In Section 3.3, we discuss in detail how these novel techniques are exploited in order to reach maximum performance benefit, and finally, Section 3.4 comprises a number of implementation details that can be very useful for code developers.

3.1. Basic building blocks. In this section we briefly cover the basic building blocks that make up the parallel distributed-memory solution of large and sparse linear systems by means of BDDC preconditioning. The solution of the global linear system (2.1) is reduced to the preconditioned (iterative) solution of the interface problem (2.3), where M_{BDDC} is used as a preconditioner for the latter problem. Krylov subspace methods applied to the interface problem require the Schur complement-vector multiplication to be computed with a "sufficiently high" level of accuracy. In order to multiply S by a vector, several (i.e., as many as subdomains) local linear systems of size proportional to subdomain size have to be solved. The memory available per core in current multicore-based machines limits the size of these local linear systems within a range where sparse direct methods are significantly faster than preconditioned Krylov subspace (local) solvers to reach the aforementioned "sufficiently high" level of accuracy. Therefore, for efficiency purposes, it is highly recommended to use sparse direct methods [8] in this setting.

Table 3.1 depicts the main phases involved in the solution of the global linear system (2.1), when its solution is reduced to the preconditioned (iterative) solution of the interface problem (2.3). We can distinguish an initial phase encompassing lines 1 and 2 of Algorithm 1, where the Schur complement and BDDC preconditioner are set

up, respectively, and an iterative phase in line 5, where the conjugate gradient (CG) method is accelerated using the BDDC preconditioner for the solution of the interface problem (2.3). The former phase is illustrated with more detail in Algorithm 2. It consists of a repeated sequence of the following four basic operations: application of the preconditioner (lines 2 and 8), Schur complement-vector products (lines 1, 5, and 7), inner products (lines 5, and 9), and vector updates (lines 1, 6, 7, and 10).[5] Inbetween the PCG stage, a pair of Dirichlet problems for the interior nodes are solved in lines 3 and 6 of Algorithm 1. The first one is required to set up the right-hand side g of the interface linear system right before its iterative solution, while the second one extends the solution to the interior nodes (x_I) from the solution on the interface (x_Γ) once it has been computed iteratively.

Algorithm 1: Solve $Ax = f$ via BDDC-PCG	
1: Set-up Schur complement S	See Algorithms 3 and 4
2: Set-up preconditioner M_{BDDC}	See Algorithms 6 and 7
3: $g := f_\Gamma - A_{\Gamma I} A_{II}^{-1} f_I$	
4: Set initial solution x_0	
5: $x_\Gamma := \text{PCG}(S,\, M_{\text{BDDC}},\, g,\, x_0)$	See Algorithm 2
6: $x_I := A_{II}^{-1}(f_I - A_{I\Gamma} x_\Gamma)$	

Algorithm 2: $x := \text{PCG}(S,\, M_{\text{BDDC}},\, g,\, x_0)$	
1: $r_0 := g - S x_0$	See Algorithm 5
2: $z_0 := M_{\text{BDDC}}^{-1} r_0$	See Algorithm 8
3: $p_0 := z_0$	
4: **for** $j = 0, 1, \ldots,$ *till convergence* **do**	
5: $\quad \alpha_j := (r_j, z_j)/(S p_j, p_j)$	See Algorithm 5
6: $\quad x_{j+1} := x_j + \alpha_j p_j$	
7: $\quad r_{j+1} := r_j - \alpha_j S p_j$	See Algorithm 5
8: $\quad z_{j+1} := M_{\text{BDDC}}^{-1} r_{j+1}$	See Algorithm 8
9: $\quad \beta_j := (r_{j+1}, z_{j+1})/(r_j, z_j)$	
10: $\quad p_{j+1} := z_{j+1} + \beta_j p_j$	
11: **end**	

TABLE 3.1
General roadmap for the solution of the global linear system $Ax = f$ via the BDDC-PCG solver.

In a distributed-memory implementation of Algorithms 1 and 2, all data structures (i.e., matrices and vectors) and computations are split and distributed among MPI tasks conformally with the underlying non-overlapping partition of the domain. We refer the reader to [4] for a comprehensive coverage of the efficient implementation of Algorithms 1 and 2 in a distributed-memory framework. In the rest of the section, we only identify and briefly describe those computations and communications (and their corresponding complexities) that are required for Schur complement and preconditioner set-up, as well as for their application at each PCG iteration. These basic building blocks are those strictly necessary to present in Section 3.3 the rationale be-

[5] We stress that in the actual efficient implementation of Algorithm 2 only one preconditioner application and Schur complement-vector product, three inner products (including that required to compute $||r_j||_2$ for the evaluation of the convergence criterion), and three vector updates are required per PCG iteration.

hind the ideas that lead to a highly scalable parallel implementation of Algorithms 1 and 2.

In the rest of the section we consider that sparse direct methods are used for the implementation of the basic building blocks in Algorithms 1 and 2. Table 3.2 summarizes the well-known [12] order of complexity of the different stages [8] in the serial direct solution of sparse linear systems arising from the FE discretization of a square or cube with a structured mesh with n DoFs, with $d = 2, 3$ the dimension of the space.

Phase	2D complexity ($d = 2$)	3D complexity ($d = 3$)
Reordering	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Symbolic Factorization	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^{\frac{4}{3}})$
Numerical Factorization	$\mathcal{O}(n^{\frac{3}{2}})$	$\mathcal{O}(n^2)$
Triangular Solution	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^{\frac{4}{3}})$

TABLE 3.2
Complexities of the different stages in the serial direct solution of sparse linear systems.

3.1.1. Schur complement basic building blocks. Let us first start the discussion with the Schur complement set-up, and its application to a vector at each PCG iteration. The former is split into a symbolic and a numerical phase in Algorithms 3 and 4, respectively, while the latter is illustrated in Algorithm 5.

Algorithm 3: S set-up (symbolic)

1: $G_{A^{(i)}} \rightarrow \begin{bmatrix} G_{A_{II}^{(i)}} & G_{A_{I\Gamma}^{(i)}} \\ G_{A_{\Gamma I}^{(i)}} & G_{A_{\Gamma\Gamma}^{(i)}} \end{bmatrix}$
2: Reord+Symb fact($G_{A_{II}^{(i)}}$)

Algorithm 4: S set-up (numerical)

1: $A^{(i)} \rightarrow \begin{bmatrix} A_{II}^{(i)} & A_{I\Gamma}^{(i)} \\ A_{\Gamma I}^{(i)} & A_{\Gamma\Gamma}^{(i)} \end{bmatrix}$
2: Num fact($A_{II}^{(i)}$)

Algorithm 5: $y_\Gamma := Sx_\Gamma$

1: $t := -A_{I\Gamma}^{(i)} x_\Gamma^{(i)}$
2: Solve $A_{II}^{(i)} u = t$
3: $y_\Gamma^{(i)} := A_{\Gamma\Gamma}^{(i)} x_\Gamma^{(i)} + A_{\Gamma I}^{(i)} u$

In Algorithm 3, line 1, the graph that captures the coupling among local interior DoFs is first extracted from that which captures the coupling among all local DoFs, which is then reordered and symbolically factorized in line 2. Similarly, in Algorithm 4, $A_{II}^{(i)}$ is extracted from $A^{(i)}$ and the sparse Cholesky factorization of $A_{II}^{(i)}$ is computed in line 2. Finally, Algorithm 5 follows a three-step process, where a local Dirichlet problem is solved for the local interior nodes in line 2 by means of sparse backward/forward substitution. The bulk of the computation is concentrated in line 2 of Algorithms 3, 4, and 5. Complexities for these computations are given in Table 3.2, with $n = n_I^i$ the number of local interior DoFs in a subdomain. No communications are required in any of these three algorithms.

3.1.2. Preconditioner basic building blocks. The BDDC preconditioner set-up is split into a symbolic and a numerical phase in Algorithms 6 and 7, respectively, while its application to a residual is depicted in Algorithm 8. Communication stages are labeled as "GC" or "LC" depending on whether they are of global (i.e., all MPI tasks involved) or local (i.e., MPI tasks communicate with each other within subsets of tasks) nature, respectively. Algorithms 6, 7, and 8 require global gather/scatter communication, and local exchanges among nearest neighbours.

For Algorithm 6, it is assumed that a topological classification of the nodes in each subdomain interface into geometrical entities (i.e., corners, edges, and faces) is already available, as well as a global numbering of these entities [4]. This greatly simplifies the (local) identification of coarse-grid DoFs in line 1 of Algorithm 6. Lines 2-3 are related to the fine-grid preconditioning level, while lines 4-11 to the coarse-grid one. In the former lines, the adjacency graph of the sparse coefficient matrix corresponding to the Neumann problem is first locally assembled, which is then reordered and symbolically factorized in line 3. In the latter lines, the adjacency graph which describes the coupling among coarse DoFs is first assembled by means of a distributed-memory algorithm in lines 4-11, and is then reordered and symbolically factorized in line 12. It is important to stress that Algorithm 6 is more scalable than the one presented in [4], as the latter does not parallelize the computation of the row counts and adjacency lists of G_{S_0}.

Algorithm 6: M_{BDDC} set-up (symbolic)

1:	Identify and count (n^i_{cts}) local coarse DoFs	
2:	Construct $G_{A^{(i)}_{\mathrm{F}}}$	
3:	Reord+Symb fact($G_{A^{(i)}_{\mathrm{F}}}$)	
4:	Gather n^i_{cts}	GC
5:	Gather global identifiers of geometric entities of each coarse DoF	GC
6:	Compute a global ordering of coarse DoFs (define R_{0i} and its transpose)	
7:	Scatter global ordering of coarse DoFs	GC
8:	Fetch n^i_{cts} of/from my neighbours	LC
9:	Fetch global identifiers of the coarse DoFs of my neighbours	LC
10:	Compute row counts/adjacency lists of G_{S_0} for local coarse DoFs	
11:	Gather row counts/adjacency lists of G_{S_0}	GC
12:	Reord+Symb fact(G_{S_0})	

The computations in Algorithm 7 can be subdivided into those related to the fine-grid preconditioning level (lines 1-2) and those related to the coarse-grid one (lines 3-7). Fine-grid duties include the construction of the sparse coefficient matrix corresponding to the Neumann problem and the computation of its sparse Cholesky factorization in lines 1 and 2, respectively. At the coarse-grid preconditioning level, a basis of the coarse-grid correction space is first computed in line 3. Once the contributions from each subdomain to the coarse-grid coefficient matrix are computed in line 4, the MPI task in charge of it then gathers these contributions and performs the matrix assembly corresponding to R_{0i} in order to build S_0 in lines 5 and 6, respectively. Finally, the MPI task in charge of the coarse-grid problem performs a sparse Cholesky factorization of S_0 (see line 7).

Algorithm 8 first injects the residual into the BDDC correction space via I^t (see line 1). On the one hand, fine-grid preconditioning level duties include the extraction of a correction from the fine-grid space by means of the solution of local Neumann problems (see line 2). On the other hand, coarse-grid correction duties encompass lines 3-8. Once the contributions from each subdomain to the coarse-grid residual are computed in line 3, the MPI task in charge of it then gathers these contributions and performs the vector assembly associated to R_{0i} in order to build r_0 in lines 4 and 5, respectively. The MPI task in charge of the coarse-grid linear system solves it by means of sparse forward/backward substitution, and then this solution is scattered

Algorithm 7: M_{BDDC} set-up (numerical)

1: Construct $A_{\mathrm{F}}^{(i)} = \begin{bmatrix} A_{II}^{(i)} & A_{I\Gamma}^{(i)} & 0 \\ A_{\Gamma I}^{(i)} & A_{\Gamma\Gamma}^{(i)} & C_i^t \\ 0 & C_i & 0 \end{bmatrix}$

2: Num fact($A_{\mathrm{F}}^{(i)}$)

3: Solve $\begin{bmatrix} A_{II}^{(i)} & A_{I\Gamma}^{(i)} & 0 \\ A_{\Gamma I}^{(i)} & A_{\Gamma\Gamma}^{(i)} & C_i^t \\ 0 & C_i & 0 \end{bmatrix} \begin{bmatrix} \Phi_I^{(i)} \\ \Phi_\Gamma^{(i)} \\ \Lambda^{(i)} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \mathrm{Id}^{(i)} \end{bmatrix}$

4: Compute $S_0^{(i)} \leftarrow {\Phi^{(i)}}^t A^{(i)} \Phi^{(i)} = {\Phi^{(i)}}^t (-C_i^T \Lambda^{(i)})$

5: Gather $S_0^{(i)}$ GC

6: Compute $S_0 \leftarrow \sum_{i=1}^{n_{\mathrm{sbd}}} R_{0i}^t S_0^{(i)} R_{0i}$

7: Num fact(S_0)

from this task to all subdomains, so that all subdomains get the coarse-grid correction on its local coarse DoFs. The solution on the local coarse DoFs is then extended to subdomain interior nodes in line 8. Finally, both corrections are injected into the original space of vectors via the weighting operator I (see line 9).

Algorithm 8: $z := M_{\mathrm{BDDC}}^{-1} r$

1: Compute $r^{(i)} \leftarrow I_i^t r$ LC

2: Solve $\begin{bmatrix} A_{II}^{(i)} & A_{I\Gamma}^{(i)} & 0 \\ A_{\Gamma I}^{(i)} & A_{\Gamma\Gamma}^{(i)} & C_i^t \\ 0 & C_i & 0 \end{bmatrix} \begin{bmatrix} t \\ s_{\mathrm{F}}^{(i)} \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ r^{(i)} \\ 0 \end{bmatrix}$

3: Compute $r_0^{(i)} \leftarrow {\Phi^{(i)}}^t r^{(i)}$

4: Gather $r_0^{(i)}$ GC

5: Compute $r_0 \leftarrow \sum_{i=1}^{n_{\mathrm{sbd}}} R_{0i}^t r_0^{(i)}$

6: Solve $S_0 z_0 = r_0$

7: Scatter z_0 into $z_0^{(i)}$, $i = 1, 2, \ldots, n_{\mathrm{sbd}}$

8: Compute $s_{\mathrm{C}}^{(i)} \leftarrow \Phi^{(i)} z_0^{(i)}$

9: Compute $z^{(i)} \leftarrow I_i(s_{\mathrm{F}}^{(i)} + s_{\mathrm{C}}^{(i)})$ LC

At the fine-grid preconditioning level, the bulk of the computation is concentrated in lines 3, 2, and 2 of Algorithms 6, 7, and 8, respectively. Complexities for these computations are given in Table 3.2, with n the number of local DoFs in a subdomain (see Remark 3.1). At the coarse-grid preconditioning level, it is concentrated in line 12 of Algorithm 6, in lines 3, and 7 of Algorithm 7, and in line 6 of Algorithm 8, with n in Table 3.2 the global number of coarse DoFs (except for line 3 of Algorithm 7 where n is equal to the number of local DoFs in a subdomain).[6]

REMARK 3.1. *The implementation of BDDC within our codes follows the strategy*

[6] The global number of coarse DoFs is in turn proportional to the number of subdomains in the partition for structured meshes [4]. However, we have experimentally observed a super-linear growth of coarse DoFs with the number of subdomains when using automatic partitioning (see [2, 3]).

comprehensively covered in [4] *(and originally presented in* [9]*) in order to transform lines 2 and 3 of Algorithm 7, and line 2 of Algorithm 8 in a four-step process, where the computational bulk is concentrated in the factorization and solution, respectively, of a local symmetric positive definite linear system. The size of this latter system is equivalent for BDDC(c), BDDC(ce), and BDDC(cef).*

3.2. Tackling bottlenecks associated to the solution of the coarse-grid problem. In this section we comparatively overview two parallel distributed-memory implementation approaches for BDDC. First, a typical implementation approach of DD methods with two-level structure is covered.[7] Then, our novel parallelization approach is presented.

Figure 3.1 illustrates a typical implementation approach of two-level DD methods. On a first fine-grid preconditioning level, the subdomains resulting from the non-overlapping partition of the global domain are mapped to the MPI tasks, with a one-to-one mapping among subdomains, MPI tasks and computational cores of the underlying distributed-memory computer. On this level, both computation and message-passing among MPI tasks are inherently of local nature, therefore, highly parallel. On the second level, the one corresponding to the global coupling among subdomains, the coarse-grid problem is assembled and solved serially on one of the MPI tasks (or redundantly solved serially in all MPI tasks of the main MPI communicator [4]) and therefore no parallelism is exploited at all.

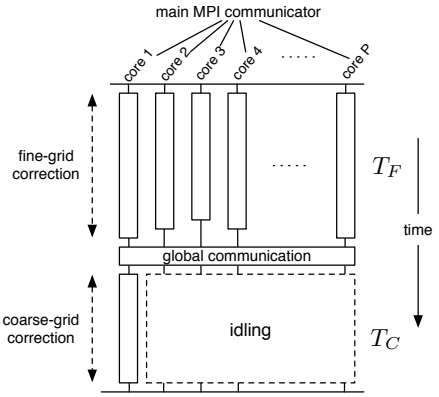


FIG. 3.1. *Typical parallel distributed-memory implementation of Algorithms 6, 7 and 8.*

In Figure 3.1, the fine-grid and coarse-grid computations are serialized in time, when there are actually many chances to perform some of these computations in parallel. For example, in Algorithm 8, there is no data dependency among lines 2 and 6. We refer the reader to Section 3.3 for a comprehensive treatment of parallelism opportunities within BDDC preconditioning. Besides, as the fine and coarse-grid corrections are solved serially, the amount of parallel overhead caused by idle MPI tasks grows with the same order of complexity as that of the complexity of the solver applied to the coarse-grid problem. What is more problematic, given the memory available per core in current multicore-based distributed-memory architectures (in the range 1-4 GBytes per core) and the order of memory complexity of sparse direct methods

[7] This implementation approach is followed by state-of-the art numerical libraries such as PETSc [5] or Freefem++ [17].

when applied to the coarse-problem, it is evident that for large-scale simulations the Cholesky factor of the coarse-grid matrix will no longer fit into the memory available per core. This does not only imply an inefficient usage of the underlying computational resources, but that the problem at hand cannot be solved. This bottleneck will become even more severe in the near future, given the current design trend of adding more and more cores at the node level. *A multilevel extension* [23, 28, 30] *of the two-level BDDC preconditioner and/or a distributed-memory solver for the coarse problem* [16, 24, 28] *can reduce the effect of these bottlenecks, but still is not possible to get rid of the parallel overhead associated to idle MPI tasks.*

In order to tackle the aforementioned bottlenecks, we propose the novel implementation approach illustrated in Figure 3.2. The global set of MPI tasks (i.e., the global MPI communicator) is split into those that have duties on the fine-grid preconditioning level (fine-grid MPI communicator), and those that have duties on the coarse-grid one (coarse-grid MPI communicator), *so that the computation of fine-grid and coarse-grid corrections can be overlapped in time.* Besides, as there are separate tasks devoted to the solution of the coarse-grid problem, additional node(s) resources (memory and cores) can be allocated for this computation. The hope is that now the number of tasks devoted to fine-grid duties (P_{F}) and coarse-grid duties (P_{C}) can always be tuned for the problem at hand so that the coarse-grid computations are (completely) masked due to the effect of overlapping. The degree of success of this approach depends though on a number of factors such as, e.g., subdomain size/load per core or the parallel scalability of the coarse-grid solver. Finally, we would like to stress that the approach can also be applied in a multilevel setting [23, 28, 30] (by splitting recursively the coarse-grid communicator into two additional communicators), although only two-level preconditioning is explored in this work.

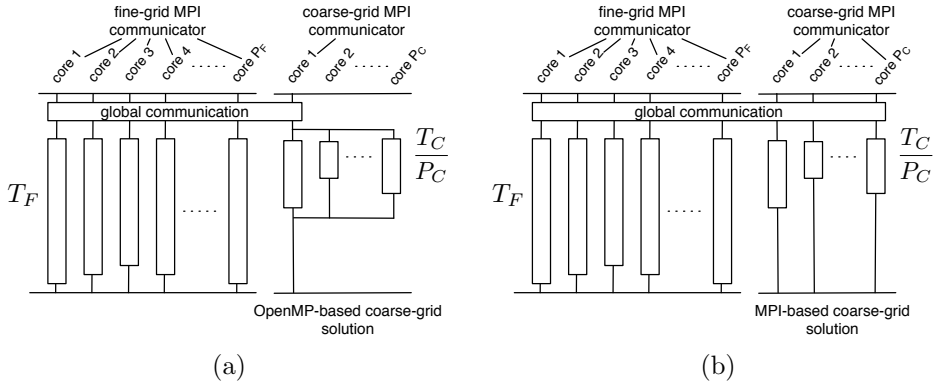


FIG. 3.2. *Highly scalable parallel distributed-memory implementation of Algorithms 6, 7 and 8. It combines two novel implementation techniques: overlapping in time of fine-grid/coarse-grid duties, and a parallel (a) multi-threaded or (b) distributed-memory solver for the coarse-grid problem.*

Figure 3.2 illustrates two possible approaches for the parallel solution of the coarse-grid problem. In Figure 3.2 (a), the task devoted to coarse-grid duties exploits multithreading within a dedicated node using an OpenMP coarse-grid solver. In Figure 3.2 (b), this approach is generalized into a MPI-based solution that distributes the coarse-grid problem (possibly) spanning multiple nodes. The former approach is (significantly) easier to code than the latter, and can already efficiently solve very

large-scale problems (as shown in Section 4). The latter approach is not explored here, and is left as future work for the solution of extremely large-scale problems.

3.3. Computation/communication re-scheduling and mapping. In order to extract the maximum performance benefit from the techniques proposed in Section 3.2, the different computations and communications arising in Algorithms 1 and 2 (and their basic building blocks in Section 3.1.1 and 3.1.2) have to be re-scheduled and mapped to the MPI tasks in such a way that the maximum degree of overlapping among fine-grid and coarse-grid duties is achieved. This non-trivial exercise considers these computations/communications as a whole (instead of being part of separated algorithms) and modifies the order in which they are scheduled while taking care of data dependencies among them. The solution that we have designed is depicted in Table 3.3.

Table 3.3 clearly evidences three areas or regions, separated by global communication stages, where overlapping among fine-grid and coarse-grid duties is possible. These three areas encompass the most three computationally-dominant operations of the coarse-grid preconditioning level:

- Overlapping area #1: the reordering and symbolic factorization of G_{S_0} is overlapped with lines 2-3 of Algorithm 6, Algorithm 3, and the start of Algorithm 7 (lines 1-4). In 3D, the complexity of the former computation is $\mathcal{O}(n^{\frac{4}{3}})$, with n being n_{cts}, while that of the most computationally dominants from the latter set, $\mathcal{O}(n^{\frac{4}{3}})$, $\mathcal{O}(n^{\frac{4}{3}})$, $\mathcal{O}(n^2)$, and $\mathcal{O}(n^{\frac{4}{3}})$, with n being n_i.
- Overlapping area #2: the assembly and numerical factorization of S_0 is overlapped with Algorithm 4, and those communications/computations preceding the gathering of coarse-grid residual contributions within the first application of the preconditioner in line 2 of Algorithm 2. These encompass line 3 of Algorithm 1, line 1 of Algorithm 2 and lines 1 and 3 of Algorithm 8. The most computationally dominant computation of the former set grows as $\mathcal{O}(n^2)$, with n being n_{cts}, while the latter also grows as $\mathcal{O}(n^2)$, but with n being n_i.
- Overlapping area #3: the assembly of r_0 and the solution of the coarse-grid linear system is overlapped with the computation of the fine-grid correction. The most computationally dominant operation in both sets grows as $\mathcal{O}(n^{\frac{4}{3}})$, with n being n_{cts} and n_i, respectively.

Summarizing, the three overlapping regions have the same order of complexity for both the coarse and fine component, the difference being that the fine component depends on the local problem size (load per processor) whereas the coarse one grows with the number of coarse DoFs (which increase with the number of processors).

Table 3.3 as a whole only considers the S and M_{BDDC} set-up stages and the header of the PCG phase. During the PCG loop, overlapping among fine-grid/coarse-grid duties is present within each application of the preconditioner (line 8 of Algorithm 2), as depicted on the region of Table 3.3 below the dashed horizontal line. Overall, the main conclusion that can be extracted from Table 3.3 is that there are great chances for overlapping in the solution of linear systems via BDDC preconditioning.

3.4. Code implementation details. The techniques proposed in Section 3.2 and 3.3 have been implemented in the FEMPAR (FE Multiphysics and massively PARallel) numerical software. FEMPAR is an in-house developed, parallel hybrid OpenMP/MPI, object-oriented (OO) framework which, among other features, provides the basic tools for the efficient parallel distributed-memory implementation of sub-structuring DD solvers [4]. Before this work, FEMPAR provided a typical parallel

Fine-grid MPI tasks	Coarse-grid MPI task
Identify and count (n^i_{cts}) local coarse DoFs	
Gather n^i_{cts}	
Gather global identifiers of geometric entities of each coarse DoF	
	Compute a global ordering of coarse DoFs (define R_{0i} and its transpose)
Scatter global ordering of coarse DoFs	
Fetch n^i_{cts} of/from my neighbours	
Fetch global identifiers of the coarse DoFs of my neighbours	
Compute row counts/adjacency lists of G_{S_0} for local coarse DoFs	
Gather row counts/adjacency lists of G_{S_0}	
Construct $G_{A^{(i)}_{\text{F}}}$ Reord+Symb fact($G_{A^{(i)}_{\text{F}}}$) $G_{A^{(i)}} \rightarrow \begin{bmatrix} G_{A^{(i)}_{II}} & G_{A^{(i)}_{I\Gamma}} \\ G_{A^{(i)}_{\Gamma I}} & G_{A^{(i)}_{\Gamma\Gamma}} \end{bmatrix}$ Reord+Symb fact($G_{A^{(i)}_{II}}$) Construct $A^{(i)}_{\text{F}}$ Num fact($A^{(i)}_{\text{F}}$) Solve $\begin{bmatrix} A^{(i)}_{II} & A^{(i)}_{I\Gamma} & 0 \\ A^{(i)}_{\Gamma I} & A^{(i)}_{\Gamma\Gamma} & C^t_i \\ 0 & C_i & 0 \end{bmatrix} \begin{bmatrix} \Phi^{(i)}_I \\ \Phi^{(i)}_\Gamma \\ \Lambda^{(i)} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \text{Id}^{(i)} \end{bmatrix}$ Compute $S^{(i)}_0 \leftarrow {\Phi^{(i)}}^t A^{(i)} \Phi^{(i)} = {\Phi^{(i)}}^t(-C_i^T \Lambda^{(i)})$	Reord+Symb fact(G_{S_0})
Gather $S^{(i)}_0$	
$A^{(i)} \rightarrow \begin{bmatrix} A^{(i)}_{II} & A^{(i)}_{I\Gamma} \\ A^{(i)}_{\Gamma I} & A^{(i)}_{\Gamma\Gamma} \end{bmatrix}$ Num fact($A^{(i)}_{II}$) $g := f_\Gamma - A_{\Gamma I} A^{-1}_{II} f_I$ $r_0 := g - S x_0$ Compute $r^{(i)} \leftarrow I^t_i r$ Compute $r^{(i)}_0 \leftarrow {\Phi^{(i)}}^t r^{(i)}$	Compute $S_0 \leftarrow \sum_{i=1}^{n_{\text{sbd}}} R^t_{0i} S^{(i)}_0 R_{0i}$ Num fact(S_0)
Gather $r^{(i)}_0$	
Solve $\begin{bmatrix} A^{(i)}_{II} & A^{(i)}_{I\Gamma} & 0 \\ A^{(i)}_{\Gamma I} & A^{(i)}_{\Gamma\Gamma} & C^t_i \\ 0 & C_i & 0 \end{bmatrix} \begin{bmatrix} t \\ s^{(i)}_{\text{F}} \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ r^{(i)} \\ 0 \end{bmatrix}$	Compute $r_0 \leftarrow \sum_{i=1}^{n_{\text{sbd}}} R^t_{0i} r^{(i)}_0$ Solve $S_0 z_0 = r_0$
Scatter z_0 into $z^{(i)}_0$, $i = 1, 2, \ldots, n_{\text{sbd}}$	
Compute $s^{(i)}_{\text{C}} \leftarrow \Phi^{(i)} z^{(i)}_0$	
Compute $z^{(i)} \leftarrow I_i(s^{(i)}_{\text{F}} + s^{(i)}_{\text{C}})$	

TABLE 3.3

Mapping of Algorithms 1 and 2 and its basic building blocks in Section 3.1.1 and 3.1.2 to fine-grid and coarse-grid MPI tasks to achieve the maximum degree of overlapping in time.

distributed-memory implementation of two-level DD preconditioning. The implementation of the techniques proposed in this work required significant code re-factoring within FEMPAR, as these techniques involve a switch from a classical SPMD (Single Program Multiple Data) model of parallel execution to a less standard MPMD model of execution. The following development tasks were (essentially) performed within

FEMPAR.

Task 1. Implementation of two different paths/fluxes for the MPI tasks, the one corresponding to the fine-grid tasks and that followed by the MPI task devoted to coarse-grid duties. The global communicator is split into a sub-communicator which includes the former tasks and another sub-communicator which includes the latter task. All communication/computation subroutines in the FEMPAR stack were classified into those which have only to be executed by fine-grid tasks (e.g., a global reduction for the computation of a scalar product, sparse matrix-vector multiplication), those which have only to be performed by the dedicated coarse-grid task (e.g., solution of the coarse-grid problem) and those which have to be executed by both fine-grid and coarse-grid tasks (e.g, the assembly of the coarse-grid problem on the dedicated MPI task from data distributed over the fine-grid tasks). The data flow in such a way that all subroutines have access to the global communicator and both sub-communicators. The rank of the MPI tasks on each communicator and the kind of subroutine determines the action to be performed by each task. For example, when the coarse-grid task enters a subroutine that has to be executed only by fine-grid tasks, the subroutine immediately returns the control to the calling subroutine on the coarse-grid task. As a result of this approach, the fine-grid tasks and the coarse-grid task follow different paths in the code, so that the desired overlapping effect can be achieved.

Task 2. Explicit change of the order in which some subroutines are called within the DD codes. For example, the classical order in Algorithm 8: (1) computation of the fine-grid correction; (2) gather/assembly of the coarse-grid residual; (3) solution of the coarse-grid problem; and (4) scatter of the solution, is rescheduled in such a way that (2) is performed before (1) so that (1) and (3) can be performed in parallel. A similar strategy was followed for the subroutines implementing the rest of algorithms.

Task 3. Replacement of MPI-1 intracommunicators by MPI-2 intercommunicators (see [14, 15]) in order to accurately capture the communication pattern among the fine-grid and coarse-grid tasks where appropriate (e.g., for transferring coarse-grid residual contributions from the fine-grid tasks to the coarse-grid task).

Task 4. Design of solutions in order to let the coarse-grid task have access to data available only on the fine-grid tasks. The data distribution is such that the coarse-grid task does not have access to data related to the fine-grid preconditioning level. For example, the coarse-grid task does not hold on its memory space a piece of the fine-grid residual, so that it does not participate in the evaluation of its 2-norm, and thus cannot decide whether the PCG iteration converged or not. We had to extend our templated implementation of Krylov subspace solvers within FEMPAR so that a new templated subroutine associated to the preconditioner allows to transfer data from the fine-grid tasks to the coarse-grid task. Any preconditioner object (e.g., BDDC) that specializes FEMPAR's templated Krylov subspace solvers has to provide a new method which is in charge of this data transfer. This is easy to implement as the preconditioner object has access to the MPI sub-communicators that split the global communicator into fine-grid/coarse-grid tasks.

4. Numerical experiments.

In this section we evaluate the *weak scalability* of the proposed highly scalable parallel distributed-memory implementation of the BDDC method, and compare it to that of the typical parallel implementation, for the 3D Poisson and linear elasticity problems on structured and unstructured meshes.

4.1. Experimental framework.

The parallel codes in FEMPAR heavily use standard computational kernels provided by highly-efficient vendor implementations

of the BLAS. Besides, through proper interfaces to several third party libraries, the local fine-grid and the global coarse-grid problems in two-level DD methods can be solved by either sparse direct or approximate solvers. In this work, we explore PARDISO [26, 27] software package for the direct solution of these problems. PARDISO provides highly-efficient *parallel multi-threaded* code implementations of sparse direct solvers.

All experiments reported in this section were obtained on a pair of similar large-scale multicore-based distributed-memory machines:

- HELIOS, located in Rokkasho (Japan) at the Computer Simulation Centre (CSC) of the International Fusion Energy Research Centre (IFERC). HELIOS features 4,410 bullx B510 compute blades, arranged in a QDR Infiniband interconnected cluster architecture. Each blade is equipped with two Intel Xeon E5-2680 eight-core processors running at 2.7 GHz (16 computational cores in total) and 64 GBytes of DDR3 memory (4GBytes per core), and runs a bullx SUperCOmputer Suite A.E.2.1 operating system. The codes were compiled using Intel Fortran compiler (12.1.6) with recommended optimization flags and we used bullxmpi (1.1.16.5) tools and libraries for native message-passing. The codes were linked against the BLAS/LAPACK, PARDISO available on the Intel MKL library (version 10.3, build 12),
- CURIE, located in Bruyères-le-Châtel (France) at the Très Grand Centre de Calcul (TGCC) of the French Alternative Energies and Atomic Energy Commission (CEA). CURIE features 5,040 B510 bullx nodes, with a very close hardware architecture and software stack to that of HELIOS. However, slightly older versions of the Intel Fortran compiler (12.1.0) and the Intel MKL library (version 10.3, build 7) are installed by default on CURIE.

4.2. BDDC for 3D Poisson with structured meshes on HELIOS. In this section we evaluate the *weak scalability* of the codes subject of study when applied to the solution of the Poisson problem on a rectangular prism $\overline{\Omega} = [0, 2] \times [0, 1] \times [0, 1]$. We consider a global conforming uniform mesh (partition) of $\overline{\Omega}$ into hexahedra and a trilinear FE discretization (i.e., Q1 FEs). The 3D mesh is partitioned into cubic grids of $P = 4m \times 2m \times 2m$ cubic subdomains. These subdomains are handled by as many MPI tasks as subdomains, which are distributed over $m^3 = 2^3, 3^3, \ldots, 12^3$ compute blades ($128, 432, \ldots, 27648$ cores), with $4 \times 2 \times 2$ subdomains/MPI Ranks per blade and one MPI Rank per core. In the case of the typical parallel implementation, the coarse-grid problem is handled by one of the tasks that also handles a subdomain. However, for our novel parallel implementation, an additional MPI task is spawn in order to perform coarse-grid related computations. This coarse-grid MPI task is mapped to an additional blade and has full access to its memory (64 GBytes) and computational resources (16 cores). The multi-threaded sparse direct solvers in PARDISO are employed for the solution of the coarse-grid problem. PARDISO is mapped within this node such that one thread is executed per core. The number of threads/cores considered in the study will be 1, 2, 4, 8, and 16.

The quotient among subdomain and mesh characteristic sizes, i.e., $\frac{H}{h}$, provides a measure of the local problem size.[8] The number of FEs (i.e., hexaedra) on each local cubic subdomain is indeed $\frac{H}{h} \times \frac{H}{h} \times \frac{H}{h}$, and that of the global mesh is given by $4m\frac{H}{h} \times 2m\frac{H}{h} \times 2m\frac{H}{h}$. The experiments performed in this section are selected in

[8] The number of subdomains is $n_{\text{sbd}} = \mathcal{O}(H^{-d})$ and the size of the global problem (2.1) is $n = \mathcal{O}(h^{-d})$. Thus, the local problem size is $\mathcal{O}((\frac{H}{h})^d)$.

order to evaluate at which rate the total computation time (i.e., that spent in all phases of Algorithm 1) evolves with fixed $\frac{H}{h}$ and increasing number of cores (within the aforementioned range).[9] As the trade-off among the factors determining the scalability of the codes depends on $\frac{H}{h}$, we perform the study with several values of fixed problem size $\frac{H}{h} = 10, 20, 30$, and 40.

4.2.1. Preliminary experimental study of overlapping areas. In order to preliminary assess the potential behind the techniques proposed in Section 3 to tackle the bottleneck associated to the computations in the coarse-grid preconditioning level, we consider the three overlapping areas in isolation, and measure the computation time of those operations that are related to the fine-grid preconditioning level (left side of the overlapping areas in Table 3.3), and those related to the coarse-grid problem (right side of the overlapping areas in Table 3.3).

Figure 4.1 reports the results of this preliminary experimental study for the three overlapping areas (from left to right: area #1, #2 and #3) and the BDDC(c), BDDC(ce), and BDDC(cef) solvers (from top to bottom) applied to the 3D Poisson problem on HELIOS. In each plot of Figure 4.1, the lines parallel to the x-axis report the time spent in fine-grid preconditioning level computations with increasing loads per core (from 10^3=1K to 40^3=64K FEs per core). This magnitude is constant (i.e., does not depend on the number of subdomains) because computation and communication on this level is local and therefore highly parallel/scalable. On the other hand, computation times for the coarse-grid preconditioning level are reported for 1, 2, 4, 8 and 16 cores exploited within the blade devoted for these computations, except for overlapping area #1 (first column of Figure 4.1), as the reordering and symbolic factorization phase is not parallelized within PARDISO.

Let us now focus the discussion on the results achieved with the BDDC(c) solver (i.e., first row of Figure 4.1). The main conclusion that can be extracted from overlapping area #1 (i.e., first column) is that with a relatively small load of 8K FEs per core, the reordering and symbolic factorization of G_{S_0} can be completely masked up to 27K cores. This is because the computational time of the latter coarse-grid computation grows at a very moderate pace with the number of subdomains, but also because it can be overlapped with a large bunch of fine-grid computations (in particular, the reordering and symbolic factorization of the graphs associated to the Dirichlet/Neumann problems, the numerical factorization of the coefficient matrix associated to the Dirichlet problem, and the local computation of the coarse-grid correction space basis vectors). For overlapping area #2, the numerical factorization of S_0 grows more rapidly with the number of subdomains (in particular as the square of the number of subdomains). If only one core is used for the numerical factorization of S_0, a larger load of 27K FEs per core is required to completely mask this computation. However, using additional cores for the coarse-grid problem, we can progressively get rid of this increase until finally 4 cores are already sufficient to completely mask this computation with a load of 8K FEs per core. A similar observation can be made for overlapping area #3.

The results achieved with the BDDC(ce) and BDDC(cef) solvers are reported on the second and third row of Figure 4.1, respectively. Additional continuity constraints for the BDDC correction space (i.e., continuity of mean values on edges, and continuity of mean values on edges and faces, respectively) result in increased preconditioning

[9] Ideally, the total computation time should remain constant under this scenario, meaning that the algorithm-code-architecture combination has the ability to still be 100% efficient in the exploitation of additional computational resources for the solution of proportionally larger problems.

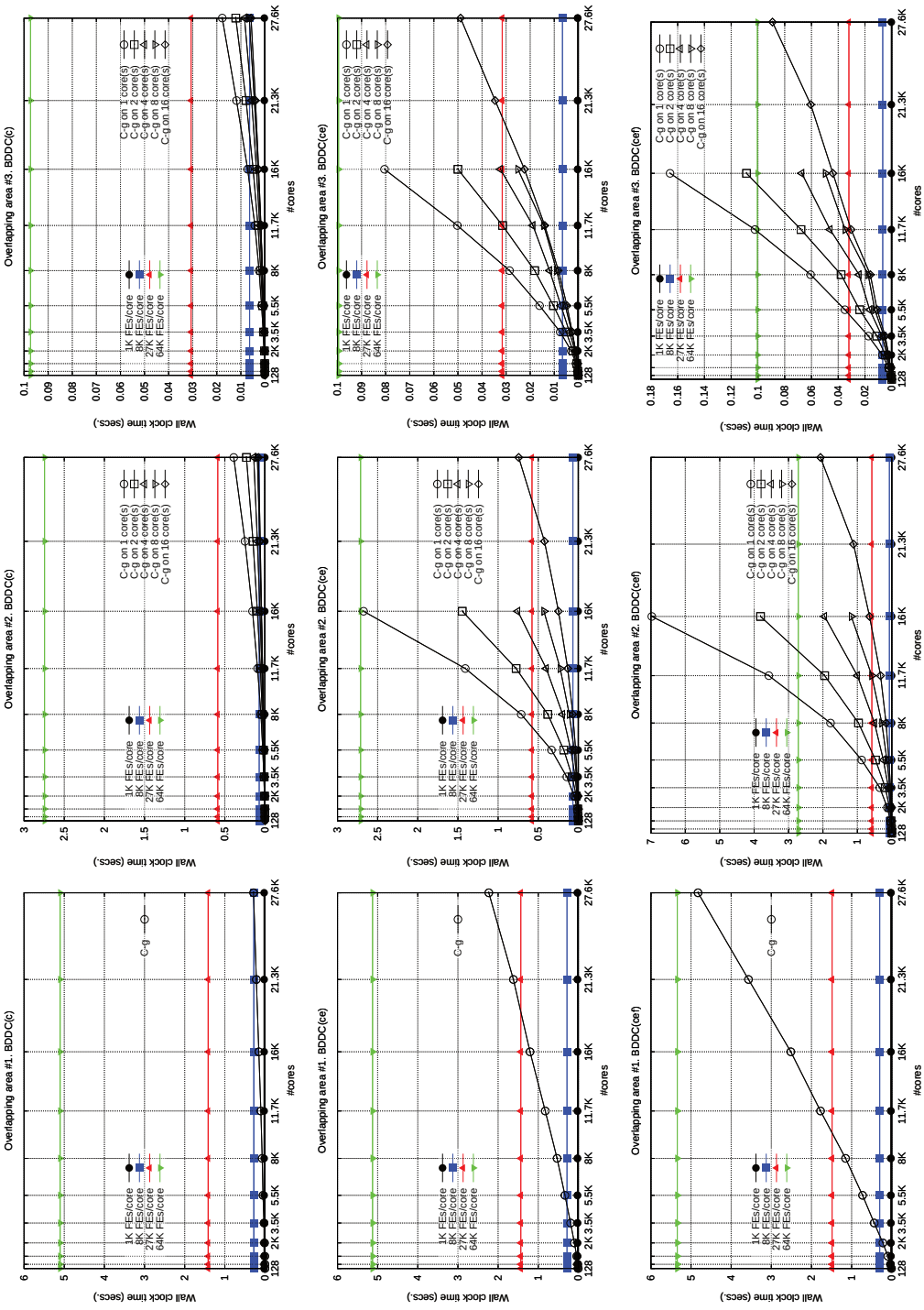


FIG. 4.1. *Computation times of fine-grid and coarse-grid (C-g) computations for the three overlapping areas (from left to right: area #1, #2 and #3) and the BDDC(c), BDDC(ce), and BDDC(cef) solvers (from top to bottom) applied to the 3D Poisson problem on HELIOS.*

robustness at the price of a larger, with a denser stencil, coarse-grid problem.[10] This does not actually affects the complexity of the phases in a sparse direct solver when applied to the coarse-grid problem, but the constant that determines the computation times observed in practice. This is immediately observed in Figure 4.1, e.g., in overlapping area #1, where the computation time spent in the symbolic factorization of G_{S_0} grows more rapidly for BDDC(ce) versus BDDC(c), and for BDDC(cef) versus BDDC(ce) . Note that additional continuity constraints do not actually increase the computation time spent in fine-grid calculations within overlapping areas #1 and #3. As pointed out in Remark 3.1, the size of the (symmetric positive definite) matrix that its symbolically and numerically factorized in overlapping area #1, and extensively that of the local linear system which is solved during overlapping area #3, is equivalent for the BDDC(c), BDDC(ce) and BDDC(cef) solvers, justifying this observation.

As a consequence of these factors, the balance achieved among the computation time of fine-grid/coarse-grid computations for BDDC(ce) and BDDC(cef) solvers is altered in such a way that, e.g., in overlapping area #1, a load of 27K FEs per core is not sufficient to completely mask the symbolic factorization of G_{S_0} beyond 16K and 8K cores for the BDDC(ce) and BDDC(cef), respectively. If we turn our attention to overlapping areas #2 and #3, we can conclude that it is highly advisable to exploit 16 cores for the numerical factorization and solution of the coarse-grid problem. With a load of 27K FEs per core, this is not even sufficient to completely mask coarse-grid computations beyond 21.3K and 16K cores for BDDC(ce) and overlapping areas #2 and #3, respectively, and beyond 16K and 11.7K cores for BDDC(cef). However, if the load is scaled up to 64K FEs per core, the coarse-grid computations in all three overlapping areas are completely masked for both BDDC(ce) and BDDC(cef) solvers. We note that this is not such a large load per core, as 550 MBytes are approximately consumed per core on the fine-grid tasks, compared to the 1-4 Gbytes of memory available per core in current multicore-based distributed-memory machines. In other words, by means of a further increase in the load per core, there is still chance for the proposed techniques to be effective in the solution of larger-scale 3D Poisson problems on larger number of cores.

4.2.2. Raw weak scalability. Figures 4.2 (a), (b), and (c) compare the weak scalability for the total computation time (in seconds) of the typical parallel implementation (curves labeled as "no overlapping") to that of our novel parallel implementation with additional number of cores devoted to the multi-threaded solution of the coarse-grid problem (curves labeled as "Coarse-grid on c core(s)", with $c = 1, 2, \ldots, 16$), for the BDDC(c), BDDC(ce) and BDDC(cef) solvers, respectively. We did not run the typical parallel implementation beyond the 5.5K core limit because we were limited in the consumption of the underlying parallel resources. In line 4 of Algorithm 1, we set the initial solution vector guess $x_0 = 0$, and the PCG iteration (see Algorithm 2) is stopped whenever the residual r_k at a given iteration k satisfies $\|r_k\|_2 \leq 10^{-6}\|r_0\|_2$. This set-up also applies to the rest of experiments in this paper.

Figure 4.2 (a) reveals a significant improvement of the scalability of our novel implementation of the BDDC(c) solver compared to that of the typical parallel implementation with the smallest load of 1K FEs per core. For such a small load per

[10] For example, adding edge constraints to a correction space which only enforces continuity of values across corners, is similar to the effect of having tricuadratic elements on the coarse-grid mesh instead of trilinear ones [4].

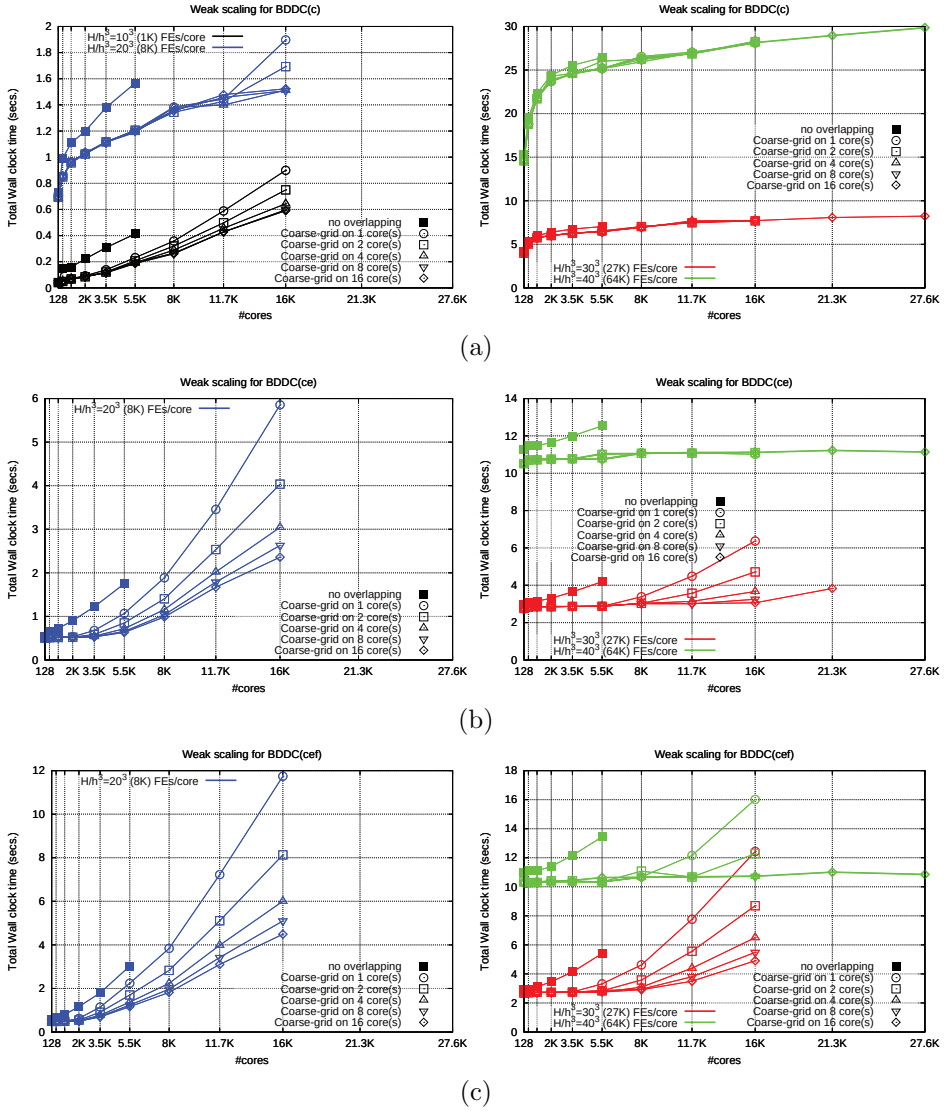


FIG. 4.2. *Weak scalability for the total computational time of the (a) BDDC(c), (b) BDDC(ce), and (c) BDDC(cef) solvers for the 3D Poisson problem on HELIOS. Left:* $\frac{H}{h} = 10, 20$. *Right:* $\frac{H}{h} = 30, 40$. *The parallel multi-threaded sparse direct solution of the coarse-grid linear system was mapped to an additional blade. Weak scaling curves are reported for 1, 2, 4, 8 and 16 cores exploited within this blade.*

core, the balance among the computation times of fine-grid and coarse-grid computations is such that the latter dominates the total computation time, justifying (part of) the loss of parallel efficiency that we observe with increasing number of cores. However, with the help of the novel techniques proposed in this paper, we can observe a first scalability boost due to the effect of overlapping, and a further improvement using additional cores for the coarse-grid problem. With a larger load of 8K FEs per core, overlapping just suffices to mask coarse-grid computations up to

11.7K cores.[11] Beyond this limit, additional cores have to be employed in the solution of the coarse-grid problem, as expected by our experimental study with overlapping areas in Section 4.2.1. With larger loads per core (i.e., 27K/64 FEs per core), it can be observed that the proposed techniques are highly successful in tackling the coarse-grid problem bottlenecks up to 27K cores, as any loss of parallel efficiency only depends on how fast the BDDC(c)-PCG achieves an asymptotically constant converge rate with the number of cores.

The reader may have realized that we did not report the results with the smallest load of 1K FEs per core for the BDDC(ce) and BDDC(cef) solvers in Figures 4.2 (b) and (c), respectively. This is justified by the fact that the curves corresponding to this load were almost coincident to those of the 8K FEs per core (specially for the BDDC(cef) solver). This observation reveals that using such small loads per core results in the coarse-grid computations largely dominating the total computation time. On these scenarios, overlapping has little impact on improving the scalability of the whole solution process, and any scalability boost mainly comes from the exploitation of additional cores for the coarse-grid problem, as observed in the left-hand side of Figures 4.2 (b) and (c). However, as the load per core is increased in the right side of Figures 4.2 (b) and (c), the combination of both overlapping and a parallel multi-threaded sparse direct solver becomes increasingly more effective to mask the extra costs associated to the the coarse-grid problem, until finally these extra costs are completely masked up to 27K cores using the largest load of 64K FEs/core. For example, to have an idea of the significance of this result, in Figure 4.2 (b), a problem with 8.2 MDoFs was solved in the same time (approximately 10 seconds) on 128 cores than a problem with 1769 MDoFs (i.e., 216 larger) on 27K cores.

Although actually not observed in Figure 4.2 (as we did not run the typical parallel implementation beyond 5.5K cores), we stress that the proposed techniques also allow to scale much further in the number of cores and size of the global problem, as the typical implementation exhausts the memory of one core much more rapidly than the highly-scalable implementation does with the memory of one node, specially with "large" (i.e., relatively to the memory capacity per core) loads per core.

4.3. BDDC for 3D linear elasticity with structured meshes on HELIOS. In this section we evaluate the *weak scalability* of the codes subject of study when applied to a vector-valued problem, the numerical approximation of the (compressible) linear elasticity PDE (with Lamé parameters equal to one) using Q1 FEs on a rectangular prism $\overline{\Omega} = [0,2] \times [0,1] \times [0,1]$. The same experiment set-up to that selected for the Poisson problem (see Section 4.2) is considered here, but we now set $\frac{H}{h} = 35$ (i.e., 43K FEs/core) as the largest local problem size. This load per core is smaller than the largest one considered for the Poisson problem (i.e., $\frac{H}{h} = 40$). The linear elasticity problem is a vector-valued problem with 3 unknowns per FE mesh node. This implies that, for a given FE mesh (i.e., that corresponding to the local Dirichlet/Neumann problems, or the global coarse-grid problem), the size of the discrete operator is 3 times larger to that of the Poisson problem, and has 9 times more nonzero entries. Roughly speaking, these factors also apply to the sparse Cholesky factor of the discrete operators. Indeed, with $\frac{H}{h} = 35$ the memory consumption per fine-grid task is approximately 2.7 GBytes, compared to the 550 MBytes for $\frac{H}{h} = 40$

[11] We stress that the loss of parallel efficiency observed in the figure up to this core limit is completely due to the fact that the BDDC(c)-PCG solver did not yet reach an asymptotically constant convergence rate.

in the case of the Poisson problem.

Figure 4.3 compares the weak scalability of both parallel implementations when applied to the linear elasticity problem. Although not reported here for brevity, we also studied the overlapping areas in isolation in the case of the linear elasticity problem. Overall, very similar observations from those made in Figure 4.2 can be made, with subtle differences, as the balance among the constants that determine the actual communication/computation times of the basic building blocks, and thus the scalability of both solutions, is affected by the switch from 1 to 3 unknowns per FE mesh node. For scenarios (largely) dominated by coarse-grid computations, as e.g., the left-hand sides of Figure 4.3 (b) and (c), (almost) any boost in the parallel scalability comes from the exploitation of multi-threading parallelism for the solution of the coarse-grid problem. However, for a "sufficiently large load per core" (e.g., 8K FEs per core in the left side of Figure 4.3 (a), or 30K FEs per core in the right side of Figures 4.3 (b) and (c)) just overlapping is already highly successful in tackling the extra costs associated to the solution of the coarse-grid problem within a first range of cores. Beyond this range, the combination of both techniques is required to (efficiently) scale the codes up to 27K cores.

It is important to stress that the study of the overlapping areas in the case of the linear elasticity problem revealed that the proposed techniques are close to reach its limits beyond 27K cores, in the sense that beyond this limit the coarse-grid computations start dominating the total computation time, even when 16 cores are used for their multi-threaded computation. Besides, by means of a further increase in the load per core, there is little chance for the proposed techniques to still be (highly) weakly scalable in the solution of larger-scale 3D linear elasticity problems on larger number of cores, as the largest load per core considered in the study already consumes an amount of memory which is close to (if not higher than) the memory available per core in current distributed-memory computers. Beyond 27K cores, a multilevel extension of the BDDC method (so that the computational/memory demands in the solution of the coarse-grid problem are reduced) and/or a distributed-memory coarse-grid problem (so that additional nodes/cores can be devoted for this computation) becomes a must to still be highly weakly scalable for the 3D linear elasticity problem on structured meshes.

4.4. BDDC for 3D Poisson with a complex domain and unstructured meshes on CURIE. The purpose of this section is to experimentally show that the proposed novel implementation can also significantly improve the scalability of the typical parallel implementation when applied to a more realistic case. More realistic cases include a domain with a more complex geometry than the one considered so far (i.e., rectangular prism) and are discretized by means of unstructured FE meshes which are split using automatic partitioners. We stress, however, that the purpose of the section is not to comprehensively assess the weak scalability of the highly scalable parallel implementation as we did with the structured test case in Sections 4.2, and 4.3. This would require, on the one hand, to consider a wide range of test cases instead of a single example (as the rate at which the size of the coarse-grid increases, and therefore the scalability of the BDDC preconditioning approach, is dependent on the underlying geometry of the domain). On the other hand, this would also require to explore wider ranges for the number of subdomains and scale of the problem to the ones considered here.

The codes subject of study were applied to the discrete operator resulting from the linear FE discretization (i.e., $P1$-elements) of the 3D Poisson problem on the domain

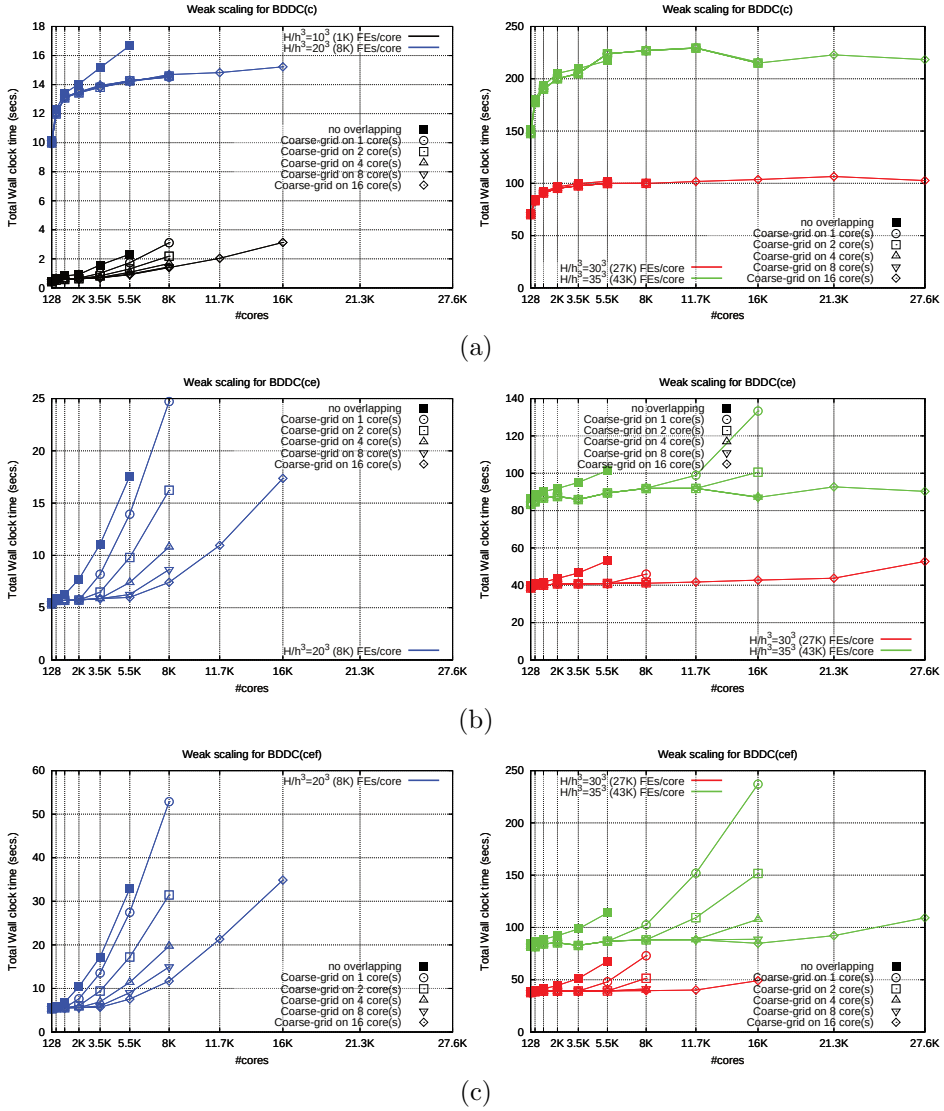
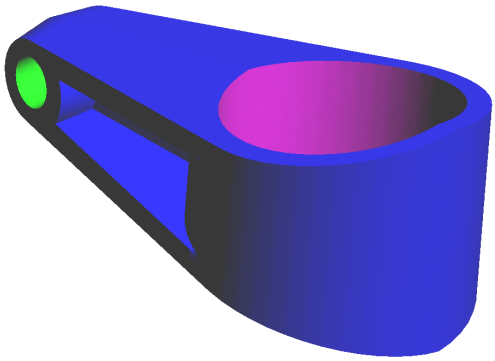


FIG. 4.3. *Weak scalability for the total computational time of the (a) BDDC(c), (b) BDDC(ce), and (c) BDDC(cef) solvers for the 3D linear elasticity problem on HELIOS. Left:* $\frac{H}{h} = 10, 20$. *Right:* $\frac{H}{h} = 30, 35$. *The parallel multi-threaded sparse direct solution of the coarse-grid (C-g) linear system was mapped to an additional blade. Weak scaling curves are reported for 1, 2, 4, 8 and 16 cores exploited within this blade.*

depicted in the top part of Table 4.1. We started from three conforming unstructured tetrahedral meshes. These were generated using three different (increasingly smaller) mesh diameters h, such that when partitioned into 16 subdomains, a local problem size of L1 $\approx$ 13.6K nodes/68.4K tetrahedra, L2 $\approx$ 45.1K nodes/241K tetrahedra, and L3 $\approx$ 99.6K nodes/547K tetrahedra are obtained, respectively. The global number of nodes and tetrahedra in these three meshes are given in the bottom part of Table 4.1, in the row corresponding to 16 subdomains. Each of these three meshes was uniformly refined up to three times. Given that each level of uniform refinement multiplies the

number of tetrahedra by a factor of 8, by increasing in the same proportion the number of subdomains (cores) at each level of mesh refinement (from 16 to 8192), we (approximately) have the desired scenario of fixed load per subdomain/core, and increasing number of cores. The global number of nodes and tetrahedra in the meshes that resulted from each level of uniform mesh refinement are given in the bottom of Table 4.1, in the rows corresponding to 128, 1024, and 8192 subdomains. All meshes were partitioned on a shared-memory multiprocessor with 2TBytes of memory using the multilevel graph bisection algorithms available in METIS 5.1.0 [19]. The largest mesh could not be partitioned because shared-memory capacity was exceeded, as indicated by a † in the bottom part of Table 4.1.



	L1		L2		L3	
#subdomains	#nodes	#FEs	#nodes	#FEs	#nodes	#FEs
16	217K	1.09M	722K	3.86M	1.59M	8.76M
128	1.59M	8.76M	5.45M	30.6M	12.2M	70.1M
1024	12.2M	70.1M	42.4M	247M	95.5M	561M
8192	95.5M	561M	334M	1.97G	†	†

TABLE 4.1

Complex domain with two cylindrical holes (top) and number of nodes and tetrahedra (#FEs) in the unstructured computational meshes used for the weak scalability study in Table 4.2 (bottom). These meshes were partitioned into 16, 128, 1024, and 8192 parts using METIS 5.1.0 [19] *(64-bit integer version), so that three different computational loads per subdomain of L1 ≈ 13.6K nodes/68.4K tetrahedra, L2 ≈ 45.1K nodes/241K tetrahedra, and L3 ≈ 99.6K nodes/547K tetrahedra are considered for the study. A † indicates that METIS could not partition the corresponding (huge) mesh because main shared-memory capacity (2TBytes) was exceeded.*

Table 4.2 reports the total computational time (row labeled as "T_{ovlap}") for the highly scalable parallel implementation of the BDDC(c), BDDC(ce), and BDDC(cef) solvers when applied to the problem described in Table 4.1 using 16 cores for the multi-threaded sparse direct solution of the coarse-grid problem, and the number of PCG iterations (row labeled as "#iter") required to reduce six orders of magnitude the 2-norm of the original residual. The ratio $\frac{T_{\mathsf{no\ ovlap}}}{T_{\mathsf{ovlap}}}$ measures the improvement of the implementation proposed with respect to the typical pure MPI implementation of the solvers. The size and non-zeros in the coarse-grid sparse coefficient matrix are also provided in the rows labeled as "n_c" and "n_z", respectively. At this point it is important to mention that the BDDC method was supplied with a corner detection mechanism that ensures the well-posedness (i.e., invertibility) of the local Neumann problems and that of the global coarse-grid problem. In particular, in line of the strategy proposed in [31] for structural problems, we are using a naive strategy for

the Poisson problem that meets this requirement by ensuring that a corner exists per each pair of subdomains that share a face. In general, this naive strategy adds more corners than those that are actually strictly required, so that the experiment can be somehow seen as a worst-case scenario for the rate at which the size of the coarse-grid problem grows with the number of subdomains (and therefore for the scalability of the codes subject of study). Indeed, while for the structured case the size of the coarse-grid problem increases linearly with the number of subdomains, it is *superlinear* for the unstructured case reported in Table 4.2. These (significantly) faster growth is justified by the combined effect of an irregular/non-uniform geometry and underlying partition, and the extra corners added by the aforementioned corner detection mechanism.[12]

A larger growth factor for the size of the coarse-grid problem immediately implies that coarse-grid related computations become earlier dominant for fixed load per core and increasing number of subdomains. This is confirmed in Table 4.2, that reveals two different scenarios for the weak scalability of T_{ovlap}. Within the range [16-1024] cores, there is only a mild increase for T_{ovlap} with fixed load per core and increasing number of subdomains for the BDDC(ce) and BDDC(cef) solvers. This mild increase is due to the fact that either T_{ovlap} was dominated by fine-grid computations (as e.g., with 16 and 128 subdomains with all loads per core, and 1024 subdomains with L3), or because the combination of overlapping of fine-grid/coarse-grid computations and a multi-threaded coarse-grid solver was highly successful to mask the extra cost associated to the solution of the coarse-grid problem (as, e.g, with 1024 subdomains and L1, L2). This is confirmed by small ratios $\frac{T_{\mathsf{no\_ovlap}}}{T_{\mathsf{ovlap}}}$ for the former case (e.g., 1.05 for BDDC(ce), 128 subdomains, and L1), and larger ones for the latter (e.g., 1.88 for BDDC(cef), 1024 subdomains, and L1). For the BDDC(c) solver there is a higher increase of T_{ovlap}, and the higher the load per core, the higher the increase (see row labeled as "#iter" in Table 4.2); this behaviour, that was also present in the structured case, can be justified by the higher increase of PCG iterations with the number of subdomains and is a direct consequence of the factor n/P in the condition number bound (2.8). However, for 8192 subdomains, a larger increase for T_{ovlap} is observed, specially for BDDC(ce) and BDDC(cef), as the coarse-grid problem becomes dominant for L1 and L2. For these cases, the largest ratios $\frac{T_{\mathsf{no\_ovlap}}}{T_{\mathsf{ovlap}}}$ are observed (e.g., 3.02 for BDDC(cef), 8192 subdomains and L1) with the more dominant the coarse-grid problem, the larger the ratio (as most of the improvement comes from the use of multi-threaded parallelism in the solution of the coarse-grid problem). Although by means of a further increase in the load per core (i.e., a further level of mesh refinement), there is still chance for the proposed techniques to still scale efficiently on larger number of cores (as the memory consumed per core was approximately 350, 560, and 890 MBytes for L1, L2 and L3, respectively), it is clear from this realistic test case that a distributed-memory and/or a multilevel extension of the two-level BDDC solver becomes earlier necessary with the number of subdomains given the larger growth factor for the size of the coarse-grid problem.

5. Conclusions and future work. The scalability of a pure MPI, SPMD implementation of the BDDC-PCG algorithm is degraded, since the coarse-grid problem size grows with the number of processors. This growth is linear for structured

[12]This behavior has already been discussed in [3]. In fact, this is the reason why a pure MPI implementation of BNN is generally faster than BDDC for unstructured meshes [3]; the number of coarse DoFs in the BNN method is always proportional to the number of subdomains.

Solver		#subdomains										
		16			128			1024			8192	
		L1	L2	L3	L1	L2	L3	L1	L2	L3	L1	L2
BDDC(c)	T_{ovlap}	0.80	4.55	16.0	1.46	8.93	31.3	1.87	12.7	45.4	5.58	16.4
	#iter	23	26	27	66	77	86	87	120	138	88	137
	$\frac{T_{\mathsf{no\ ovlap}}}{T_{\mathsf{ovlap}}}$	1.02	1.02	1.02	1.03	1.03	1.01	1.16	1.02	1.01	1.85	1.58
	n_c	26	27	32	267	271	283	2,878	2,802	2,709	40,304	37,855
	nnz_c	186	239	304	4,045	4,123	4,455	105,440	100,656	92,967	2,591,266	2,329,883
BDDC(ce)	T_{ovlap}	0.74	4.18	14.5	0.86	5.07	17.7	1.13	6.06	21.6	14.8	17.5
	#iter	17	19	19	21	24	24	26	29	30	31	36
	$\frac{T_{\mathsf{no\ ovlap}}}{T_{\mathsf{ovlap}}}$	1.03	1.02	1.02	1.05	1.04	1.02	1.58	1.12	1.01	2.69	2.86
	n_c	42	48	51	641	644	671	9,245	9,525	9,387	119,958	122,149
	nnz_c	612	874	945	28,477	27,612	30,305	861,837	916,617	911,277	17,836,942	18,617,387
BDDC(cef)	T_{ovlap}	0.76	4.16	14.6	0.90	5.17	17.7	1.22	6.24	22.1	24.1	27.8
	#iter	16	18	18	19	23	22	24	27	29	30	34
	$\frac{T_{\mathsf{no\ ovlap}}}{T_{\mathsf{ovlap}}}$	1.01	1.02	1.03	1.03	1.05	1.01	1.88	1.19	1.03	3.02	3.05
	n_c	73	83	84	1,040	1,046	1,080	13,954	14,346	14,200	169,796	173,361
	nnz_c	1,625	2,215	2,228	58,702	57,584	62,196	1,517,050	1,610,806	1,604,932	28,707,174	30,082,263

TABLE 4.2

Weak scalability on CURIE for the total computation time (T_{ovlap}) in seconds and number of PCG iterations (#iter) for the highly scalable implementation of the BDDC(c), BDDC(ce) and BDDC(cef) solvers for the 3D Poisson problem on a complex domain (see Table 4.1). A sparse direct solver was used for the solution of local Dirichlet/Neumann problems, as well as for the solution of the global coarse-grid problem. For T_{ovlap}, 16 cores were used in the multi-threaded sparse direct solution of the coarse-grid problem. The ratio $\frac{T_{\mathsf{no\ ovlap}}}{T_{\mathsf{ovlap}}}$ measures the improvement of the implementation proposed with respect to the typical pure MPI implementation of the solvers. The size (n_c) and non-zeros (n_z) in the coarse-grid sparse coefficient matrix are also provided.

meshes/partitions but it has been observed to be super-linear when automatic mesh partitioners are used. The cost of the coarse problem can be small compared to the fine (fully parallel) one for large loads per processor and modest numbers of processors. However, using sparse direct solvers for the local (Dirichlet and Neumann) problems and the coarse problem, the coarse problem becomes dominant in the order of some thousands of cores (considering the 1-4 Gbytes of memory available per core in current multicore-based distributed-memory machines). Further, as we increase the number of processors, processors with coarse and fine duties can exceed memory limits. The use of distributed-memory coarse solvers [16, 24, 28] reduces this loss of scalability (by reducing the computational time of the coarse solver) and the growth of memory consumption but this step is still *serialized*.

In this work, we have proposed a novel implementation of the BDDC-PCG parallel linear solver based on overlapping fine-grid/coarse-grid duties in time. The global set of cores is split into those that have fine-grid duties and those that have coarse-grid duties. Next, the different computations and communications arising in the BDDC-PCG algorithm have been re-scheduled and mapped in such a way that the maximum degree of overlapping is achieved. E.g., for sparse direct solvers they correspond to symbolic factorization, numerical factorization and backward/forward substitutions. This work has been performed for exact (direct) solvers for both local and coarse problems and implemented as a MPMD model of execution. The extension of this approach to inexact (AMG) solvers can be found in [2].

The overlap between coarse and fine computations in the BDDC preconditioner application is possible due to the orthogonality with respect to the energy norm of fine and coarse duties. (This approach can also be applied to other additive Schwarz preconditioners, e.g., the FETI-DP method.) The resulting technique tackles/ameliorates the bottleneck associated with the solution of the coarse-grid problem on all possible scenarios. If the fine-grid correction is more expensive to compute than the coarse-grid correction, then the solution of the coarse-grid correction is fully masked by the effect of overlapping. Else, the coarse-grid correction computation time can be reduced using additional threads/cores. This way, even for the two-level BDDC method, *perfect* weak scalability plots can be attained up to 27,648 cores.[13] These results have been obtained with an OpenMP multi-threaded sparse direct solver (limiting the number of coarse cores to one full node). The use of distributed-memory coarse solvers (as commented above) and/or multilevel extensions of the overlapped implementation of the BDDC method will easily boost its perfect scalability to the order of hundreds of thousands of cores. This will be the objective of our future work.

References.

[1] P.R. Amestoy, I.S. Duff, and J.-Y. L'Excellent, *Multifrontal parallel distributed symmetric and unsymmetric solvers*, Computer Methods in Applied Mechanics and Engineering **184** (2000), no. 24, 501–520.

[2] S. Badia, A. F. Martín, and J. Principe, *On an overlapped coarse/fine implementation of balancing domain decomposition with inexact solvers*, In preparation.

[3] ———, *Enhanced balancing Neumann-Neumann preconditioning in computational fluid and solid mechanics*, International Journal for Numerical Methods in Engineering **96** (2013), no. 4, 203–230.

[4] ———, *Implementation and scalability analysis of balancing domain decomposition methods*, Archives of Computational Methods in Engineering **20** (2013), no. 3, 239–262.

[5] S. Balay, J. Brown, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, *PETSc Web page*, 2012. `http://www.mcs.anl.gov/petsc`.

[13] We note that perfect scalability seems to go on beyond this limit in some ranges, but we did not go further due to the limited access to the different machines used in this study.

[6] M. Bhardwaj, D. Day, C. Farhat, M. Lesoinne, K. Pierson, and D. Rixen, *Application of the FETI method to ASCI problems–scalability results on 1000 processors and discussion of highly heterogeneous problems*, International Journal for Numerical Methods in Engineering **47** (2000), no. 1-3, 513535 (en).
[7] S. C. Brenner and R. Scott, *The mathematical theory of finite element methods*, 3rd edition, Springer, 2010.
[8] T. A. Davis, *Direct methods for sparse linear systems*, Vol. 2, SIAM, 2006.
[9] C. R. Dohrmann, *A preconditioner for substructuring based on constrained energy minimization*, SIAM Journal on Scientific Computing **25** (2003), no. 1, 246–258.
[10] 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*, Computer Methods in Applied Mechanics and Engineering **184** (2000), no. 2–4, 333–374.
[11] C. Farhat and F.-X. Roux, *A method of finite element tearing and interconnecting and its parallel solution algorithm*, International Journal for Numerical Methods in Engineering **32** (1991), no. 6, 1205–1227.
[12] A. George, *Nested dissection of a regular finite element mesh*, SIAM Journal on Numerical Analysis **10** (1973), no. 2, 345–363.
[13] A. Grama, G. Karypis, V. Kumar, and A. Gupta, *Introduction to parallel computing*, 2nd ed., Addison-Wesley, 2003.
[14] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: portable parallel programming with the message passing interface*, Vol. 1, MIT press, 1999.
[15] W. Gropp, E. Lusk, and R. Thakur, *Using MPI-2: Advanced features of the message passing interface*, MIT press, 1999.
[16] V. Hapla, D. Horak, and M. Merta, *Use of direct solvers in TFETI massively parallel implementation*, Applied parallel and scientific computing, 2013, pp. 192–205.
[17] F. Hecht, *Freefem++ user's manual. 3rd edition, Version 3.22.*
[18] V. E. Henson and U. M. Yang, *BoomerAMG: a parallel algebraic multigrid solver and preconditioner*, Applied Numerical Mathematics **41** (2002), no. 1, 155–177.
[19] G. Karypis, *A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. Version 5.1.0*, University of Minnesota, Department of Computer Science and Engineering, Minneapolis, MN, 2013. Available at `http://glaros.dtc.umn.edu/gkhome/fetch/sw/metis/manual.pdf`.
[20] A. Klawonn and O. Rheinbach, *Highly scalable parallel domain decomposition methods with an application to biomechanics*, ZAMM - Journal of Applied Mathematics and Mechanics **90** (2010), no. 1, 532 (en).
[21] J. Mandel, *Balancing domain decomposition*, Communications in Numerical Methods in Engineering **9** (1993), no. 3, 233–241.
[22] J. Mandel and C. R. Dohrmann, *Convergence of a balancing domain decomposition by constraints and energy minimization*, Numerical Linear Algebra with Applications **10** (2003), no. 7, 639–659.
[23] J. Mandel, B. Sousedík, and C. Dohrmann, *Multispace and multilevel BDDC*, Computing **83** (2008), no. 2, 55–85.
[24] O. Rheinbach, *Parallel iterative substructuring in structural mechanics*, Archives of Computational Methods in Engineering **16** (2009), no. 4, 425–463 (en).
[25] Y. Saad, *Iterative methods for sparse linear systems*, 2nd ed., SIAM, 2003.
[26] O. Schenk and K. Gärtner, *Solving unsymmetric sparse systems of linear equations with PARDISO*, Future Generation Computer Systems **20** (2004), no. 3, 475 –487.
[27] ———, *On fast factorization pivoting methods for sparse symmetric indefinite systems*, Electronic Transactions on Numerical Analysis **23** (2006), 158–179.
[28] B. Sousedík, J. Šístek, and J. Mandel, *Adaptive-multilevel BDDC and its parallel implementation*, Computing (2013). In press.
[29] A. Toselli and O. Widlund, *Domain decomposition methods - algorithms and theory* (R. Bank, R. L. Graham, J. Stoer, R. Varga, and H. Yserentant, eds.), Springer-Verlag, 2005.
[30] X. Tu, *Three-level BDDC in three dimensions*, SIAM Journal on Scientific Computing **29** (2007), no. 4, 1759–1780.
[31] J. Šístek, M. Čertíková, P. Burda, and J. Novotný, *Face-based selection of corners in 3D substructuring*, Mathematics and Computers in Simulation **82** (2012), no. 10, 1799–1811.