On the scalability of inexact balancing domain decomposition by constraints with overlapped coarse/fine corrections

Santiago Badia[a,b], Alberto F. Martín[a,b], Javier Principe[a,b]

[a] *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`)

[b] *Universitat Politècnica de Catalunya, Jordi Girona 1-3, Edifici C1, 08034 Barcelona, Spain*

Abstract

In this work, we analyze the scalability of inexact two-level Balancing Domain Decomposition by Constraints (BDDC) preconditioners for Krylov subspace iterative solvers, when using a highly scalable asynchronous parallel implementation where fine and coarse correction computations are overlapped in time. This way, the coarse-grid problem can be fully overlapped by fine-grid computations (which are embarrassingly parallel) in a wide range of cases. Further, we consider inexact solvers to reduce the computational cost/complexity and memory consumption of coarse and local problems and boost the scalability of the solver. Out of our numerical experimentation, we conclude that the BDDC preconditioner is quite insensitive to inexact solvers. In particular, one cycle of algebraic multigrid (AMG) is enough to attain algorithmic scalability. Further, the clear reduction of computing time and memory requirements of inexact solvers compared to sparse direct ones makes possible to scale far beyond state-of-the-art BDDC implementations. Excellent weak scalability results have been obtained with the proposed inexact/overlapped implementation of the two-level BDDC preconditioner, up to 93,312 cores and 20 billion unknowns on JUQUEEN. Further, we have also applied the proposed setting to unstructured meshes and partitions for the pressure Poisson solver in the backward-facing step benchmark domain.

Keywords: Domain decomposition, inexact solvers, BDDC, parallelization, overlapping, scalability

1. Introduction

In order to deal with increasing levels of complexity in the simulation of phenomena governed by partial differential equations (PDEs), computational engineering and science must advance in the development of numerical algorithms and implementations that will efficiently exploit the ever-increasing amount of

computational resources. The growth in computational power that resulted from Moore's law passes now through increasing the number of cores in a chip, instead of making cores faster. As a result, the next generation of supercomputers, able to reach 1 exaflop/s, is expected to reach billions of cores. The efficient exploitation of billion-fold levels of concurrency is a big challenge. The advance of large scale scientific computing will be strongly related to the ability to efficiently exploit these extreme core counts [1].

The time spent in an implicit simulation at the linear solver relative to the overall execution time grows with the size of the problem and the number of cores [2]. For extreme scale implicit simulations, a massively parallel linear solver is a key component. This scenario exacerbates the need of highly scalable algorithms and implementations. Only numerical algorithms with all their components scalable will efficiently run on extreme scale supercomputers. Extreme scale solvers should be developed under the assumption that local flops are cheap and communications expensive. On extreme core counts, it will be a must to reduce communication and synchronization among processors, and overlap communication with computation. At the largest scales, linear solvers are based on preconditioned Krylov subspace methods. Algorithmically scalable preconditioners include (algebraic) multigrid (MG) [3] and some domain decomposition (DD) algorithms [4]. However, this theoretical property is not enough for practical weak scalability, since the preconditioner itself must allow for a massively scalable implementation. Today's most scalable algorithms/implementations present practical limits of parallelism, e.g., due to the small, coarse problems to be solved in the hierarchical process for DD/AMG, and the loss of sparsity and denser communication patterns at coarser levels of AMG.

DD preconditioners make explicit use of the partition of the global mesh, e.g., for the finite element (FE) integration, into sub-meshes (subdomains) and provide a natural framework for the development of fast parallel solvers tailored for distributed-memory machines. One-level DD algorithms involve the solution of local problems and nearest-neighbors communications. A (second level) coarse correction (coupling all subdomains) is required to have algorithmic scalability, but it also harms the practical (CPU time) weak scalability. Two-level DD algorithms include the Balancing Neumann-Neumann preconditioner (BNN) [5], the Balancing DD by Constraints preconditioner (BDDC) [6], and FETI-DP preconditioners [7]. In all these cases, for positive-definite matrices, a polylogarithmic expression of the condition number of the preconditioned system $\kappa = 1 + \log^2\left(\frac{H}{h}\right)$ can be proved, where h and H are the mesh and subdomain characteristic sizes, respectively, and d is the space dimension; $(\frac{H}{h})^d$ is the local problem size. Consequently, in weak scaling scenarios, i.e., increasing the linear system size and number of processors keeping $\frac{H}{h}$ fixed, the number of iterations of the preconditioned conjugate gradient (PCG) solver is (asymptotically) independent of the number of processors.

The practical scalability limits of a two-level DD implementation is determined by the coarse solver computation, whose size increases (at best) linearly with respect to the number of subdomains. The coarse problem rapidly be-

comes the bottleneck of the algorithm as we increase the number of processors, reducing weak scalability [8]. The coarse problem is several orders of magnitude smaller than the original global system, and only a very small portion of the computing cores can efficiently be exploited (assuming a parallel coarse solver). In typical DD implementations, this produces an unacceptable parallel efficiency loss, since all the cores not involved in the coarse solver computation are idling (see Figure 1). One obvious strategy to improve scalability is to reduce the wall-clock time spent at the coarse solver by using, e.g., a MPI-distributed sparse direct solver like MUMPS [9] (see [10] for BDDC and [11] for FETI-DP). However, this approach only mitigates the problem.

2. Motivation

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

(P1) It allows for a mathematically supported extremely aggressive coarsening. The ratio between the size of the global and coarse problem is of the order of the local problem size, i.e., $(\frac{H}{h})^d$. On memory-constrained supercomputers, it is in the order of 10^5 for sparse direct methods [12] and 10^6 for inexact solvers (see Section 6).

(P2) The coarse matrix has a similar sparsity pattern as the original system matrix.

(P3) The constrained Neumann and Dirichlet local problems, as well as the coarse problem, can be computed in an inexact way, e.g., using one AMG cycle without affecting the algorithmic scalability of the method [13].

(P4) Due to the fact that the coarse matrix has a similar structure as the original system matrix, a multilevel extension of the algorithm is possible [14, 15].

(P5) Coarse and fine components can be computed in parallel [6], since the basis for the coarse space is constructed in such a way that it is orthogonal to the fine component space with respect to the inner product induced by the unassembled system matrix [6, 16].

Properties (P1) and (P2) are readily exploited in any BDDC implementation. Property (P3), i.e., the algorithmic scalability of BDDC with inexact solvers, has been proved by Dohrmann in [13]. Similar inexact preconditioners have been presented in [17]. The inexact BDDC method can easily increase parallel efficiency, due to the linear complexity of the coarse solver, especially at large core counts. However, as far as we know, a practical weak scalability analysis of inexact BDDC methods (at large scales) has not been carried out so far. Besides, for FETI-DP, one cycle of the MPI-distributed AMG solver in BoomerAMG [18] has been used as inexact coarse solver in [19, 20] for 2D elasticity problems.

With regard to (P4), a multilevel BDDC algorithm has been proposed in [15], where the coarse problem at the next BDDC level is approximated by its BDDC

approximation. This way, the CPU cost of the coarse problem is reduced, but the condition number bound increases with the number of levels [15]. A high-performance implementation of the multilevel BDDC method can be found in [10].

The efficient exploitation of (P5), i.e., the orthogonality between coarse and fine spaces, is not trivial. However, this property makes possible a parallel computation of coarse and fine corrections, i.e., overlapped in time. In [12], we have classified all the duties in an *exact* (i.e., using sparse direct solvers) BDDC-PCG algorithm into fine and coarse duties. These duties have been rescheduled to achieve the maximum degree of overlapping while preserving data dependencies. The actual implementation of this idea requires significant code refactoring, since it involves a switch from a data parallelism to a task parallelism paradigm, dividing processors into those having fine grid duties and those having coarse grid duties. Clearly, this approach reduces synchronization among processors, and overlaps communications/computations, following the exascale solver paradigm [1]. It has been exploited in [12], where we have performed scalability analyses for the 3D Poisson and linear elasticity problems on a pair of state-of-the-art multicore-based distributed-memory machines (HELIOS and CURIE). Excellent weak scalability has been attained up to 27K cores for reasonably high local problem sizes, e.g., $(\frac{H}{h}) = 30$ which means 27K elements/core in the 3D Poisson problem; both local and coarse problems were solved by using the multi-threaded sparse direct solver PARDISO [21].

On the hardware front, the current trend in HPC is to increase the core count per node while reducing the memory available per core. On one hand, to reduce synchronization, as in the overlapped BDDC implementation in [12], will be crucial at extreme core counts. Further, this overlapped implementation alleviates memory requirements, since fine (resp., coarse) processors do not perform/store coarse (resp., fine) solver duties/matrices. In the same direction, linear complexity inexact solvers, much less memory intensive than sparse direct methods, will certainly be favored. They should also be favored at large core counts, since the potential loss of scalability due to the coarse solver is much less dramatic. *The current state-of-the-art in DD implementations and the supercomputing trends to reach the exascale have motivated the combined overlapped/inexact BDDC implementation proposed in this work.*

In this article, we extend the overlapped implementation in [12] for exact solvers to the inexact BDDC methods proposed in [13] (with slight modifications). First, we analyze the effect of perturbing in isolation every problem at the BDDC preconditioner. Next, we propose different inexact methods, combining different numbers of AMG cycles for each internal problem. A comprehensive weak scalability analysis of the resulting overlapped/inexact BDDC implementation has been performed up to 93,312 cores and more than 20 billion unknowns on JUQUEEN, at the Jülich Supercomputing Center (JSC). This test has been performed for structured meshes and partitions and constant physical coefficients. *As far as we know, these are the largest scale scalability analyses and simulations performed so far with DD methods.*

The proposed implementation of inexact BDDC methods has been coded in FEMPAR, a massively parallel finite element solver devoted to the LES simulation of incompressible turbulent flows and MHD on unstructured meshes (see [22, 23, 24, 25]). In these problems, the typical approach is to consider a pressure segregation technique (see [26, 27]), which leads to a momentum equation that is usually integrated explicitly and a pressure Poisson solver, the bottleneck of these simulations. An efficient and scalable pressure Poisson solver is a must. As a result, we have also applied the proposed setting to unstructured meshes and partitions for the pressure Poisson problem in the backward-facing step benchmark domain. We note that free flow solvers lead to Poisson problems with constant coefficients, a main difference with respect to the flow in highly heterogeneous porous media. In these last situations, that arise, e.g., in subsurface modelling, more advanced implementations based on an adaptive BDDC coarse space [10] are needed (see also [28] for efficient techniques for these problems). Further, the use of inexact solvers for these problems would require local AMG solvers suitable for highly varying coefficients.

This work is structured as follows. Section 3 is devoted to non-overlapping DD and the BDDC preconditioner whereas Section 4 is devoted to the introduction of inexact variants. In Section 5, we extend the highly scalable parallel distributed-memory implementation of the BDDC algorithm in [12], which overlaps fine and coarse computations, to the inexact variant. In Section 6, we report a comprehensive set of numerical experiments on structured meshes that includes a study of the influence of approximately solving each internal problem in isolation and a weak scalability analysis. Numerical examples for unstructured meshes and partitions are also provided. Finally, in Section 7, we draw some conclusions and define future lines of work.

3. Balancing Domain Decomposition

3.1. Problem setting

Let us consider a bounded polyhedral domain $\Omega \subset \mathbb{R}^d$ with $d = 2, 3$ and a quasi-uniform FE partition (mesh) with characteristic size h. As model problem, we study the Poisson problem on Ω, for an arbitrary forcing term and boundary conditions (as soon as the problem is well-posed). Let $\bar{\mathbb{V}} \subset H^1(\Omega)$ be a $\mathcal{C}^0$-continuous FE space. The Galerkin approximation of the Poisson problem with respect to $\bar{\mathbb{V}}$ leads to a linear system of equations:

$$Ax = f. \tag{1}$$

Further, we consider a quasi-uniform partition of the global mesh into n_{sbd} local meshes, which induces a non-overlapping domain decomposition of Ω into subdomains Ω_i, $i = 1, \ldots, n_{\text{sbd}}$ (of characteristic size H). 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 $\Gamma := \bigcup_{i=1}^{n_{\text{sbd}}} \Gamma_i$. For every subdomain Ω_i, we introduce the local FE space of functions $\mathbb{V}_i$. $\mathbb{V} := \mathbb{V}_1 \times \ldots \times \mathbb{V}_{n_{\text{sbd}}}$ denotes the global FE space

of functions that can be *discontinuous on* Γ; clearly, $\bar{\mathbb{V}} \subset \mathbb{V}$. Obviously, all FE spaces are isomorphic to real vector spaces.

Let us define the restriction operator $R_i : \bar{\mathbb{V}} \to \mathbb{V}_i$, that applied to a vector in $\bar{\mathbb{V}}$ provides its restriction into Ω_i, and $R := R_1 \times \ldots \times R_{n_{\text{sbd}}} : \bar{\mathbb{V}} \to \mathbb{V}$. Let us also define the operator $E_i := R_i^t D_i : \mathbb{V}_i \to \bar{\mathbb{V}}$, where $D_i : \mathbb{V}_i \to \mathbb{V}_i$ is a weighting operator. The weighting operators represent a partition of unity, in the sense that $R^t D R = I$, with $D := D_1 \times \ldots \times D_{n_{\text{sbd}}} : \mathbb{V} \to \mathbb{V}$. Further, let $E := R^t D$.

The subdomain FE matrix corresponding to $\mathbb{V}_i$ is denoted by $K^{(i)}$, and its size is denoted by n_i. $K := \text{diag}\left(K^{(1)}, \ldots, K^{(n_{\text{sbd}})}\right)$ is the global sub-assembled FE matrix on $\mathbb{V}$. (Along the paper, we denote with the letter K (partially) sub-assembled matrices and with A fully assembled ones.) Analogously, we define the local sub-assembled right-hand side $g^{(i)}$ and its global counterpart g. The system matrix A and right-hand side f can be obtained after the assembly of K as $A = R^t K R$ and g as $f = R^t g$.

The non-overlapping partition induces a reordering of FE vectors into interior and interface nodes, i.e., $u = [u_I, u_\Gamma]^t$. We also define the interior restriction operator $R_I u := u_I$. It leads to the following block reordered structure of the global assembled, global sub-assembled and local matrices:

$$A = \begin{bmatrix} A_{II} & A_{I\Gamma} \\ A_{\Gamma I} & A_{\Gamma\Gamma} \end{bmatrix}, \quad K = \begin{bmatrix} A_{II} & K_{I\Gamma} \\ K_{\Gamma I} & K_{\Gamma\Gamma} \end{bmatrix}, \quad \text{and} \quad K^{(i)} = \begin{bmatrix} A_{II}^{(i)} & A_{I\Gamma}^{(i)} \\ A_{\Gamma I}^{(i)} & K_{\Gamma\Gamma}^{(i)} \end{bmatrix},$$

respectively. Matrices A_{II}, $A_{I\Gamma}$, $A_{\Gamma I}$ and $K_{\Gamma\Gamma}$ present a block diagonal structure (very amenable to parallelization), e.g., $A_{II} = \text{diag}\left(A_{II}^{(1)}, A_{II}^{(2)}, \ldots, A_{II}^{(n_{\text{sbd}})}\right)$. Matrices $K_{I\Gamma}$ and $K_{\Gamma I}$ are simply an extension by zeros of $A_{I\Gamma}$ and $A_{\Gamma I}$, respectively.

3.2. BDDC preconditioner

The BDDC preconditioner is a two-level domain decomposition method where some local fine-grid corrections and a global coarse-grid correction (that couples all subdomains and makes the preconditioner both scalable and optimal) are combined. The idea behind 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, combined with an initial and final interior correction.

The construction of the BDDC preconditioner requires a partition of the degrees of freedom (DoFs) on Γ into objects, which can be corners, edges or faces. Next, we associate to some (or all) of these objects a *coarse* DoF. The coarse DoFs can be the values of the function at the corners, or the mean values of the function on edges/faces. We define the BDDC FE space $\mathring{\mathbb{V}}$ as the subspace of functions in $\mathbb{V}$ that are continuous on coarse DoFs; clearly, $\bar{\mathbb{V}} \subset \mathring{\mathbb{V}} \subset \mathbb{V}$. 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. We denote by $\mathring{K}$ the FE matrix related to $\mathring{\mathbb{V}}$. It can

be formally obtained from the partial assembly (at coarse DoFs only) of the global sub-assembled matrix K, even though it is never implemented this way. The invertibility of this matrix depends on the definition of the coarse DoFs. Let us define:

$$P_I := R_I^t A_{II}^{-1} R_I, \qquad P_{FC} := E\mathring{K}^{-1}E^t, \qquad H := I - P_I A = \begin{bmatrix} 0 & -A_{II}^{-1}A_{I\Gamma} \\ 0 & I_\Gamma \end{bmatrix}.$$

(FC denotes fine/coarse correction and H is the so-called *discrete harmonic extension operator*.) The BDDC preconditioner M consists in a multiplicative combination of P_I, P_{FC}, and P_I. Using the fact that $P_I A P_I = P_I$, we obtain:

$$M = P_I + HP_{FC}H^t.$$

The practical implementation of the BDDC correction P_{FC} requires some elaboration. Let us consider a decomposition of the BDDC space $\mathring{\mathbb{V}}$ into a *fine space* $\mathring{\mathbb{V}}_F$ of vectors that vanish on coarse DoFs and the $\mathring{K}$-orthogonal complement $\mathring{\mathbb{V}}_C$, denoted as the *coarse space*. As a result, the BDDC FE problem can be decomposed into fine and coarse components, i.e., $\mathring{x} = \mathring{K}^{-1}E^t r = x_F + x_C$. Since fine and coarse spaces are $\mathring{K}$-orthogonal by definition, they can be computed in parallel.

The fine space functions in $\mathring{\mathbb{V}}_F$ vanish on coarse DoFs (which are the only DoFs that involve continuity among subdomains). Due to the $\mathring{K}$-orthogonality, the fine component can be defined as $x_F := E_F K_F^{-1} E_F^t$, where K_F is the Galerkin projection of K onto $\mathring{V}_F$, i.e., functions that vanish on coarse DoFs, and E_F is the restriction of E to $\mathring{V}_F$. In order to compute this fine correction in practice, we define the local matrix of constraints C_i such that, given a local vector of unknowns, it provides its local coarse DoFs values. We refer to [29] for a detailed implementation of C_i. As a result, the fine correction x_F computation only involves *constrained Neumann* problems:

$$\begin{bmatrix} K^{(i)} & C_i^t \\ C_i & 0 \end{bmatrix} \begin{bmatrix} x_F^{(i)} \\ \lambda \end{bmatrix} = \begin{bmatrix} E_i^t r \\ 0 \end{bmatrix}. \tag{2}$$

As it is described in detail in [29], the solution of the constrained Neumann problem is performed after applying a permutation that separates coarse corner DoFs (denoted by c) from the rest of DoFs (denoted by r), i.e., $x^{(i)} = [x_c^{(i)}, x_r^{(i)}]^t \in \mathbb{V}_i$. Further, we define the restriction $R_{r,i}$ such that $R_{r,i}x^{(i)} = x_r^{(i)}$. Corner DoFs can be explicitly eliminated (in fact $x_c^{(i)} = 0$ for the fine correction), leading to the system

$$\begin{bmatrix} K_{rr}^{(i)} & C_{r,i}^t \\ C_{r,i} & 0 \end{bmatrix} \begin{bmatrix} x_r^{(i)} \\ \lambda_{r,i} \end{bmatrix} = \begin{bmatrix} R_{r,i}E_i^t r \\ 0 \end{bmatrix}. \tag{3}$$

This system is solved by computing the Schur complement for the edge/face Lagrange multipliers as

$$C_{r,i}(K_{rr}^{(i)})^{-1}C_{r,i}^t \lambda_r^{(i)} = -C_{r,i}(K_{rr}^{(i)})^{-1}R_{r,i}E_i^t r. \tag{4}$$

As a result, the fine correction involves to compute the inverse of the global matrix $K_{rr} := \mathrm{diag}(K_{rr}^{(1)}, \ldots, K_{rr}^{(n_{\mathrm{sbd}})})$. There are existing mechanisms that modify the definition of objects in order to enforce K_{rr} to be invertible (see [30, 6]).

The coarse space $\mathring{\mathbb{V}}_C \subset \mathring{\mathbb{V}}$ is built as

$$\mathring{\mathbb{V}}_C = \mathrm{span}\{\Phi_1, \Phi_2, \ldots, \Phi_{n_{\mathrm{cts}}}\},$$

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. 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.,

$$\begin{bmatrix} K^{(i)} & C_i^t \\ C_i & 0 \end{bmatrix} \begin{bmatrix} \Phi^{(i)} \\ \Lambda^{(i)} \end{bmatrix} = \begin{bmatrix} 0 \\ I \end{bmatrix}. \tag{5}$$

System (5) is solved in the same way as system (2), getting:

$$\Phi^{(i)} = \begin{bmatrix} \Phi_c^{(i)} \\ \Phi_r^{(i)} \end{bmatrix}, \quad \Phi_c^{(i)} = \begin{bmatrix} I & 0 \end{bmatrix}, \quad \Phi_r^{(i)} = -(K_{rr}^{(i)})^{-1} C_{r,i}^t \Lambda_r^i - \begin{bmatrix} K_{rc}^{(i)} & 0 \end{bmatrix}, \tag{6}$$

$$C_{r,i}(K_{rr}^{(i)})^{-1} C_{r,i}^t \Lambda_r^{(i)} = -C_{r,i}(K_{rr}^{(i)})^{-1} \begin{bmatrix} K_{rc}^{(i)} & I \end{bmatrix}. \tag{7}$$

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 K_C as

$$K_C = \Phi^t K \Phi = \sum_{i=1}^{n_{\mathrm{sbd}}} R_{C,i}^t {\Phi^{(i)}}^t K^{(i)} \Phi^{(i)} R_{C,i}.$$

where $R_{C,i}$ is the coarse matrix assembly operator, i.e., the local-to-global correspondence for coarse DoFs. The subdomain contributions ${\Phi^{(i)}}^t K^{(i)} \Phi^{(i)}$ can readily be computed (in parallel) and assembled, e.g., in one processor. The coarse residual $r_C = \Phi^t E^t r$ is computed analogously (see [29]). Once K_C and r_C are assembled, the coarse correction is obtained as $x_C = \Phi K_C^{-1} r_C$. The BDDC preconditioner can finally be stated as:

$$M = P_I + H(E\Phi K_C^{-1} \Phi^t E^t + E_F K_F^{-1} E_F^t) H^t.$$

Remark 3.1. *When considering exact Dirichlet solvers, after an initial pre-correction, we can easily check that $r_I = 0$ at all Krylov iterations. In this case, P_I and H computations can be eliminated without modifying the method, i.e.,*

$$Mr = H(E\Phi K_C^{-1} \Phi^t E^t + E_F K_F^{-1} E_F^t) r,$$

leading to one interior (Dirichlet) correction per iterations, instead of two.

We refer to [31] for a proof of the following theorem, about the condition number of the BDDC-preconditioned system matrix.

Theorem 3.1. *The maximum and minimum eigenvalues of the BDDC preconditioned system matrix are:*

$$\lambda_{\min}(MA) \geq 1, \qquad \lambda_{\max}(MA) \leq \beta\omega, \qquad \textit{with } \omega := \left(1 + \log^2\left(\frac{H}{h}\right)\right),$$

for BDDC(c) or BDDC(ce) in 2D, and BDDC(ce) and BDDC(cef) in 3D, where $\beta > 0$ does not depend on (H, h). For the BDDC(c) preconditioner in 3D,

$$\lambda_{\min}(MA) \geq 1, \qquad \lambda_{\max}(MA) \leq \beta' \frac{H}{h}\omega, \qquad \textit{for } \beta' > 0 \textit{ also independent of } (H, h).$$

Remark 3.2. *The BDDC preconditioner is quasi-optimal and algorithmically scalable, since the condition number of the preconditioned system matrix only depends on the local system size, which is fixed in a weak scaling scenario. Further, the condition number is a poly-logarithmic function of $\frac{H}{h}$, with the only exception of BDDC(c) in 3D. In this last case, the condition number is affected by an additional $\frac{H}{h}$ factor, which can be large (e.g., 60 in the numerical experiments of Section 6). It justifies the large iteration counts of this method (compared to those of BDDC(ce) and BDDC(cef)).*

4. Inexact BDDC

The exact BDDC preconditioner involves some linear systems to be solved. The action of K_{rr}^{-1}, i.e., the local constrained (on the coarse corner DoFs only) Neumann problems, is required to compute the coarse basis Φ and the fine correction, the action of A_{II}^{-1}, i.e., the local Dirichlet problems, is required for the interior corrections, and the action of K_C^{-1}, i.e., the coarse problem, must be solved to compute the coarse correction. These problems are traditionally solved via sparse direct methods [6]. However, as motivated in the introduction, the use of inexact solvers is very appealing for large-scale simulations on supercomputers, due to increasing memory restrictions and higher core counts. Dohrmann has proposed and analyzed in [13] an inexact version of the BDDC method, where the local/coarse problems are replaced by preconditioners. Let us assume that we have at our disposal an approximation $\widetilde{K}_{rr}$ of K_{rr} such that:

$$\delta_F x^t K_{rr} x \leq x^t \widetilde{K}_{rr} x \leq \Delta_F x^t K_{rr} x, \qquad \forall x. \tag{8}$$

(Along this section, we assume $\delta_{(\cdot)}$ and $\Delta_{(\cdot)}$ to be positive constants independent of (H, h) and the vector space for x can be inferred from the matrices in the inequalities.) Further, let us also introduce approximations $\widehat{K}_{rr}$ and $\widetilde{A}_{II}$ of K_{rr} and A_{II}, respectively, and the following global matrices:

$$\widehat{K} := \begin{bmatrix} \widehat{K}_{rr} & K_{rc} \\ K_{cr} & K_{cc} \end{bmatrix}, \qquad \widetilde{K} := \begin{bmatrix} \widetilde{A}_{II} & K_{I\Gamma} \\ K_{\Gamma I} & K_{\Gamma\Gamma} \end{bmatrix}. \tag{9}$$

We assume that

$$\delta_\Phi x^t K x \leq x^t \widehat{K} x \leq \Delta_\Phi x^t K x, \qquad \delta_I x^t K x \leq x^t \widetilde{K} x \leq \Delta_I x^t K x, \qquad \forall x. \quad (10)$$

(We omit the reordering operators for brevity.) In order for the matrices in (9) to be semi-positive definite, we also assume

$$x^t K_{rr} x \leq x^t \widehat{K}_{rr} x, \qquad x^t A_{II} x \leq x^t \widetilde{A}_{II} x, \qquad \forall x.$$

As noted in [13], since K is singular, the kernel of K, $\widehat{K}$, and $\widetilde{K}$ must be identical to satisfy (10). Let W be such that $\ker(K) \subseteq \text{range}(W)$, and $W_I = R_I W$. Given an arbitrary approximation $\bar{A}_{II}$, we can build $\widetilde{A}_{II}$ by solving exactly on $\text{range}(W_I)$ (see [13]), i.e.,

$$\widetilde{A}_{II}^{-1} := W_I (W_I^t \bar{A}_{II}^{-1} W_I)^{-1} W_I^t + E_I \bar{A}_{II}^{-1} E_I^t, \quad E_I := I - A_{II} W_I (W_I^t \bar{A}_{II}^{-1} W_I)^{-1} W_I^t.$$

Due to the block-diagonal nature of K, this correction is local. The definition of the kernel-correction for $\widehat{K}_{rr}$ is defined analogously. Finally, for the approximation of the coarse matrix K_C we study two different options. A difference with respect to the previous problems is the fact that K_C is not available when using inexact solvers. (It involves the exact computation of Φ.) One option is to assemble the Galerkin projection of K onto the inexact coarse basis $\widehat{\Phi}$, i.e., $\widehat{\Phi}^t K \widehat{\Phi}$, and consider an approximation of this matrix such that:

$$\delta_C x^t (\widehat{\Phi}^t K \widehat{\Phi}) x \leq x^t \widetilde{K}_C x \leq \Delta_C x^t (\widehat{\Phi}^t K \widehat{\Phi}) x, \qquad \forall x. \quad (11)$$

Another approach, the one used in [13], is to consider an approximation of the coarse matrix $\widehat{\Phi}^t \widehat{K} \widehat{\Phi}$:

$$\delta_C x^t (\widehat{\Phi}^t \widehat{K} \widehat{\Phi}) x \leq x^t \widetilde{K}_C x \leq \Delta_C x^t (\widehat{\Phi}^t \widehat{K} \widehat{\Phi}) x, \qquad \forall x. \quad (12)$$

Remark 4.1. *In general, the inexact matrices $\widetilde{K}_{rr}$, $\bar{K}_{rr}$, and $\bar{A}_{II}$ are not explicitly built, and only the action of their inverses is approximated in the algorithm, e.g., using one/several AMG cycles. Further, $\widehat{K}_{rr}$, and $\widetilde{A}_{II}$ are dense matrices, due to the kernel correction. However, when using the coarse problem approximation in (12), $\widehat{K}_{rr}$ seems to be explicitly needed. Fortunately, using the inexact version of (5) (Equations (15)-(16) below) we can easily check that $K_{rr}^{(i)} \Phi^{(i)} = -C^t \Lambda$, which makes possible to compute $\widehat{\Phi}^t \widehat{K} \widehat{\Phi}$ as $-\widehat{\Phi}^t C^t \Lambda$. In any case, when $\bar{K}_{rr}^{-1}$ stands for a preconditioned Krylov solver up to some tolerance, this approach leads to a generally nonsymmetric indefinite matrix. In these situations, it is better to use (11). Besides, we have used (11) in Section 6 due to slightly better performance.*

Finally, the inexact BDDC preconditioner reads as:

$$\widetilde{M} = \widetilde{P}_D + \widetilde{H} (E \widehat{\Phi} \widetilde{K}_C^{-1} \widehat{\Phi}^t E^t + E_F \widetilde{K}_F^{-1} E_F^t) \widetilde{H}^t, \quad (13)$$

where

$$\widetilde{H} := \begin{bmatrix} 0 & -\widetilde{A}_{II}^{-1} A_{I\Gamma} \\ 0 & I_\Gamma \end{bmatrix}, \qquad \widetilde{P}_I = R_I^t \widetilde{A}_{II}^{-1} R_I, \tag{14}$$

and the inexact coarse basis is computed as

$$\widehat{\Phi}^{(i)} = \begin{bmatrix} \widehat{\Phi}_c^{(i)} \\ \widehat{\Phi}_r^{(i)} \end{bmatrix}, \quad \widehat{\Phi}_c^{(i)} = \begin{bmatrix} I & 0 \end{bmatrix}, \quad \widehat{\Phi}_r^{(i)} = -(\widehat{K}_{rr}^{(i)})^{-1} C_{r,i}^t \Lambda_r^i - \begin{bmatrix} K_{rc}^{(i)} & 0 \end{bmatrix}, \tag{15}$$

$$C_{r,i}(\widehat{K}_{rr}^{(i)})^{-1} C_{r,i}^t \Lambda_r^{(i)} = -C_{r,i}(\widehat{K}_{rr}^{(i)})^{-1} \begin{bmatrix} K_{rc}^{(i)} & I \end{bmatrix}. \tag{16}$$

Theorem 4.1. *Let us assume that (8)-(10) hold. When $\widetilde{K}_C$ satisfies (11), we have:*

$$\lambda_{\min}(\widetilde{M}A) \geq \frac{\delta_I \min(1, \Delta_F^{-1}, \delta_\Phi \Delta_C^{-1})}{\Delta_\Phi \Delta_I}, \quad \frac{\lambda_{\max}(\widetilde{M}A)}{\lambda_{\max}(MA)} \leq \frac{\Delta_I^2 \max(1, \delta_F^{-1}, \Delta_\Phi \delta_C^{-1})}{\delta_\Phi \delta_I^2}.$$

Alternatively, when $\widetilde{K}_C$ satisfies (12), we get:

$$\lambda_{\min}(\widetilde{M}A) \geq \frac{\delta_I \min(1, \Delta_F^{-1}, \Delta_C^{-1})}{\Delta_\Phi \Delta_I}, \qquad \frac{\lambda_{\max}(\widetilde{M}A)}{\lambda_{\max}(MA)} \leq \frac{\Delta_I^2 \max(1, \delta_F^{-1}, \delta_C^{-1})}{\delta_\Phi \delta_I^2}.$$

Proof. The proof of this result readily follows from the analysis in [13], the only difference being the fact that the fine correction and Φ can in general be computed using different preconditioners. It can easily be handled using the fact that K_F is a Galerkin projection of K and a result like (11) for this matrix. Further, when the coarse preconditioner is built from $\widehat{\Phi}^t K \widehat{\Phi}$, the result is readily obtained using the fact that

$$\frac{\delta_C}{\Delta_\Phi} x^t \widehat{\Phi}^t \widehat{K} \widehat{\Phi} x \leq x^t \widetilde{K}_C x \leq \frac{\Delta_C}{\delta_\Phi} x^t \widehat{\Phi}^t \widehat{K} \widehat{\Phi} x,$$

which is obtained by combining (10)-(11).

Remark 4.2. *In the inexact preconditioner $\widetilde{M}$ we have replaced the local/coarse problems by optimal approximations. Due to their block-diagonal structure, the (possibly different) preconditioners $\widetilde{K}_{rr}^{-1}$ and $\widehat{K}_{rr}^{-1}$ are locally built from $K_{rr}^{(i)}$, that can be obtained, e.g., as one AMG cycle of this matrix. Analogously, $\widetilde{A}_{II}^{-1}$ is built from local approximations of $A_{II}^{(i)}$.*

Remark 4.3. *The preconditioners for the computation of Φ and the Dirichlet problems, i.e., $\widehat{K}_{rr}$ and $\widetilde{K}_{II}$, must include the kernel correction (see [13]). On the other hand, it is not needed for the fine and coarse correction preconditioners $\widetilde{K}_{rr}$ and $\widetilde{K}_C$.*

Remark 4.4. *To compute $\widehat{\Phi}$, we must compute $(\widehat{K}_{rr}^{(i)})^{-1}C_{r,i}^t$. It implies to apply at every subdomain the local preconditioner to as many vectors as local coarse corners. This computation can be reused to build the Schur complement matrix for the edge/face constrained Neumann problem in (4), as soon as $\widetilde{K}_{rr} = \widehat{K}_{rr}$. Otherwise, we must compute $(\widetilde{K}_{rr}^{(i)})^{-1}C_{r,i}^t$. It makes suitable to consider the same preconditioner (with kernel correction) for both Φ and the fine correction.*

5. A highly scalable distributed-memory implementation

In this section we adapt the highly scalable distributed-memory implementation of the method proposed in [12] to consider inexact solvers. The global linear system (1) is solved by means of a Krylov subspace method, where the inexact BDDC preconditioner $\widetilde{M}$ is used as a global system matrix preconditioner (see Section 4). In our implementation we can consider different Krylov subspace methods (e.g. PCG, IPCG [32], FGMRES [33]) which can be used for the solution of the global problem with a BDDC preconditioner or for the solution of the local problem with an AMG preconditioner [34, 3, 35]. This feature was exploited to test some combinations not reported here, e.g., PCG-AMG methods for local problems (with coarse tolerances), and IPCG [32] for the global system. However, *inexact variants based on Krylov methods turned out to be less efficient than a fixed number of AMG cycles in all cases.* These results have not been reported for the sake of brevity.

In a distributed-memory implementation of a BDDC preconditioned Krylov subspace solver, all data structures (i.e., matrices and vectors) and computations are split and distributed among MPI tasks in concordance with the underlying non-overlapping partition of the domain. We refer the reader to [29] for a comprehensive coverage of these implementation aspects. In the rest of the section, we only identify and briefly describe those computations and communications required to implement the BDDC preconditioner with inexact solvers.

The initial set-up of the BDDC preconditioner is in turn split into a symbolic and a numerical phase in Algorithms 1 and 2, respectively, while its application to a residual is depicted in Algorithm 3. 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 1, 2, and 3 require global gather/scatter communication, and local exchanges among nearest neighbors.

During the symbolic set-up of the BDDC preconditioner presented in Algorithm 1, the adjacency graph (denoted by G_*) of matrices $A_{II}^{(i)}$ and $K_{rr}^{(i)}$ required by the Dirichlet and constrained Neumann problems is computed in lines 12 and 13. The coarse solver tasks in lines 1- 11 are identical as for the exact BDDC method in [12].

The numerical set-up of the BDDC preconditioner is presented in Algorithm 2. The operations required during this phase depend on the inexact solvers being used. E.g., the solver set-up is an incomplete numerical factorization for ILU methods, whereas it involves the construction of the hierarchy in

Algorithm 1: $\widetilde{M}$ set-up (symbolic)

1: Identify and count (n^i_{cts}) local coarse DoFs
2: Gather n^i_{cts} GC
3: Gather global identifiers of the geometric entities corresponding to each coarse DoF GC
4: Compute a global ordering of coarse DoFs (define $R_{C,i}$ and its transpose)
5: Scatter global ordering of coarse DoFs GC
6: Fetch n^i_{cts} of/from my neighbors LC
7: Fetch global identifiers of the coarse DoFs of my neighbors LC
8: Compute row counts of G_{K_C} corresponding to local coarse DoFs
9: Gather row counts of G_{K_C} GC
10: Compute adjacency lists of G_{K_C} corresponding to local coarse DoFs
11: Gather adjacency lists of G_{K_C} GC
12: Construct $G_{K^{(i)}_{rr}}$ from $G_{K^{(i)}}$
13: Construct $G_{A^{(i)}_{II}}$ from $G_{K^{(i)}}$

AMG. The tasks in Algorithm 2 can be subdivided into fine tasks (lines 1-6) and coarse tasks (lines 7-9). Fine MPI tasks include the extraction of $A^{(i)}_{II}$ and $K^{(i)}_{rr}$ in lines 1 and 2 and the set-up of their approximations, e.g., their AMG hierarchy and (possibly) the kernel-correction set-up, in lines 3 and 4, respectively. The fine duties also involve the computation of the coarse space matrix Φ in line 5 and the coarse matrix coefficients in line 6. The MPI task (or tasks) in charge of the coarse problem then gathers these contributions and performs the matrix assembly corresponding to $R_{C,i}$ in order to build K_C in lines 7 and 8, respectively. Finally, the MPI task in charge of the coarse problem performs the coarse preconditioner, e.g., the AMG hierarchy set-up of the inexact coarse matrix (see line 9).

Algorithm 2: $\widetilde{M}$ set-up (numerical)

1: Extract $A^{(i)}_{II}$ and $A^{(i)}_{I\Gamma}$ from $K^{(i)}$
2: Extract $K^{(i)}_{rr}$ and $K^{(i)}_{rc}$ from $K^{(i)}$
3: Set-up for $\widetilde{A}^{(i)}_{II}$
4: Set-up for $\widetilde{K}^{(i)}_{rr}$ and (possibly) $\bar{K}^{(i)}_{rr}$
5: Compute $\Phi^{(i)}$ using (15)-(16)
6: Compute $K^{(i)}_C \leftarrow (\widehat{\Phi}^{(i)})^t K^{(i)} \widehat{\Phi}^{(i)}$ (or alternatively, $K^{(i)}_C \leftarrow -(\widehat{\Phi}^{(i)})^t C^{(i)} \Lambda_i$)
7: Gather $K^{(i)}_C$ GC
8: Compute $K_C \leftarrow \sum_{i=1}^{n_{\text{sbd}}} R^t_{C,i} K^{(i)}_C R_{C,i}$
9: Set-up for $\widetilde{K}_C$

Algorithm 3 describes the algorithm that applies the BDDC preconditioner to a residual. First, the computation of an interior precorrection, and corresponding residual update are performed in lines 1 and 2, respectively. Then, the updated residual is extended to the BDDC space via E^t (see line 3). On the one hand, the fine-grid tasks include the computation of the fine correction by means of constrained local Neumann problems (see line 6). Once the contributions from each subdomain to the coarse-grid residual are computed in line 4, the MPI tasks in charge of the coarse problem gather these contributions and perform the vector assembly associated to $R_{C,i}$ in order to build r_C in lines 5 and 7, respectively. Next, the coarse problem is solved in an inexact way, e.g., by one/several AMG cycle(s). Finally, the solution is scattered from this task to all subdomains, so that all subdomains get the coarse-grid correction on its local coarse DoFs. Finally, both corrections are injected into $\bar{\mathbb{V}}$ via the projection E, and corrected in the interior in line 12.

Algorithm 3: $z := \widetilde{M}r$

1: Compute $\delta_I^{(i)} \leftarrow (\widetilde{A}_{II}^{(i)})^{-1} r_I^{(i)}$
2: Compute $r_\Gamma^{(i)} \leftarrow r_\Gamma^{(i)} - A_{\Gamma I}^{(i)} \delta_I^{(i)}$
3: Compute $r^{(i)} \leftarrow E^t r$ LC
4: Compute $r_C^{(i)} \leftarrow (\widehat{\Phi}^{(i)})^t r^{(i)}$
5: Gather $r_C^{(i)}$ GC
6: Compute $x_F^{(i)}$ using (3)-(4)
7: Compute $r_C \leftarrow \sum_{i=1}^{n_{\text{sbd}}} R_{C,i}^t r_C^{(i)}$
8: Solve $z_C = \widetilde{K}_C^{-1} r_C$
9: Scatter z_C into $z_C^{(i)}$, $i = 1, 2, \ldots, n_{\text{sbd}}$
10: Compute $s_{\text{C}}^{(i)} \leftarrow \widehat{\Phi}^{(i)} z_C^{(i)}$
11: Compute $z^{(i)} \leftarrow E(s_\Gamma^{(i)} + s_{\text{C}}^{(i)})$ LC
12: Compute $z_I^{(i)} = -(\widetilde{A}_{II}^{(i)})^{-1} A_{I\Gamma}^{(i)} z_\Gamma^{(i)} + \delta_I^{(i)}$

The typical implementation of the BDDC preconditioner [36, 37] is illustrated in Figure 1 (a), where a one-to-one mapping between subdomains, MPI tasks, and computational cores is used. Fine and coarse duties are serialized. The vast majority of cores only have fine duties, and only some cores have both fine and coarse duties. This is due to the dramatic reduction of size between the original and coarse matrix. As a consequence, *there is a tremendous amount of parallel overhead caused by idling, i.e., the wall-clock time required to solve the coarse problem T_C times the number of cores with fine duties only.* Further, cores with both coarse and fine duties require more memory resources. This is a problem for current multicore-based distributed-memory architectures (in the range 1-4 GBytes per core); these memory limitations are expected to be more restrictive in the future exascale supercomputers [1].

As an alternative, we have proposed in [12] a highly scalable implementa-

tion of the exact BDDC method that solves the aforementioned problems by exploiting the algorithmic property that makes possible to compute coarse and fine duties in parallel. This technique is illustrated in Figure 1 (b). The global set of MPI tasks (i.e., the global MPI communicator) is split into fine and coarse MPI tasks, i.e., those that have fine duties only (fine MPI communicator), and those with coarse duties only (coarse MPI communicator), *so that the computation of fine and coarse corrections can be overlapped in time*. Two possible approaches for the parallel solution of the coarse-grid problem are proposed in [12]: using an OpenMP coarse-grid solver within a dedicated node, as shown in Figure 1 (b), and its generalization into a MPI-based solution that distributes the coarse-grid problem.

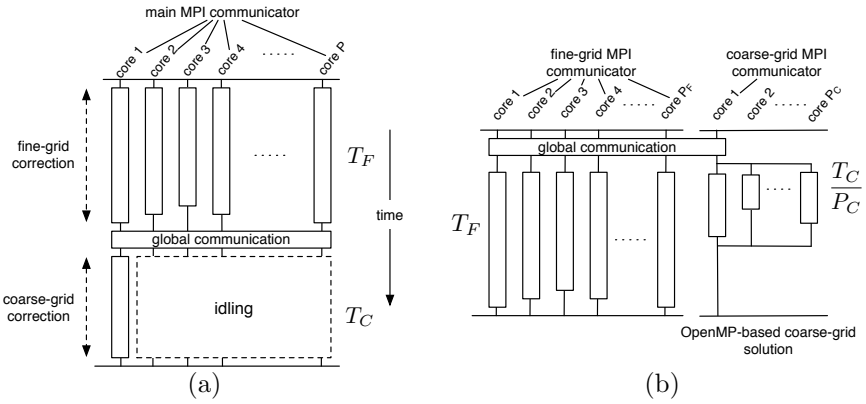


Figure 1: Comparison of (a) the typical parallel distributed-memory implementation of Algorithms 1, 2 and 3 and (b) the highly scalable one proposed in [12] implemented with multithreading.

The efficient exploitation of this idea requires an important remapping and re-scheduling of the communications and computations as well as some code refactoring, which is comprehensively described in [12]. The final result is depicted in Table 1, which is similar to the one in [12] but includes the modifications needed to use inexact solvers. Table 1 clearly evidences two areas or regions (three in the exact version [12], due to symbolic factorization), separated by global communication stages, where overlapping among fine and coarse duties is possible: the first one after gathering $K_C^{(i)}$ in line 7 of Algorithm 2 and the last one after gathering $r_C^{(i)}$ in line 5 of Algorithm 3. We stress the fact that *all coarse duties, that produce severe idling and, as a result, a loss of parallel efficiency, can be overlapped with fine duties.* Table 1 as a whole only considers the $\widetilde{M}$ set-up stages and the header of the Krylov phase. During the Krylov loop, overlapping among fine-grid/coarse-grid duties is present within each application of the preconditioner, as depicted on the region of Table 1 below the dashed horizontal line.

Fine-grid MPI tasks	Coarse-grid MPI task
Identify and count ($n^i_{\rm cts}$) local coarse DoFs	
Gather $n^i_{\rm cts}$	
Gather global identifiers of the geometric entities corresponding to each coarse DoF	
	Compute a global ordering of coarse DoFs (define $R_{C,i}$ and its transpose)
Scatter global ordering of coarse DoFs	
Fetch $n^i_{\rm cts}$ of/from my neighbors	
Fetch global identifiers of the coarse DoFs of my neighbors	
Compute row counts of G_{K_C} corresponding to local coarse DoFs	
Gather row counts of G_{K_C}	
Compute adjacency lists of G_{K_C} corresponding to local coarse DoFs	
Gather adjacency lists of G_{K_C}	
Construct $G_{A^{(i)}_{II}}$ from $G_{K^{(i)}}$ Construct $G_{K^{(i)}_{rr}}$ from $G_{K^{(i)}}$ Construct $K^{(i)}_{rr}$ and $K^{(i)}_{rc}$ from $K^{(i)}$ Set-up for $(\widehat{K}^{(i)}_{rr})$ and (possibly) $\bar{K}^{(i)}_{rr}$ Compute $\Phi^{(i)}$ using (15)-(16) Compute $K^{(i)}_C \leftarrow (\widehat{\Phi}^{(i)})^t K^{(i)} \widehat{\Phi}^{(i)}$ (or $K^{(i)}_C \leftarrow -(\widehat{\Phi}^{(i)})^t C^{(i)} \Lambda_i$)	
Gather $K^{(i)}_C$	
Construct $A^{(i)}_{II}$ and $A^{(i)}_{I\Gamma}$ from $K^{(i)}$ Set-up for $\widetilde{A}^{(i)}_{II}$ $r_0 := f - Ax_0$ $x_0 := x_0 - A^{-1}_{II} r_0$ $r_0 := f - Ax_0$	Compute $K_C \leftarrow \sum_{i=1}^{n_{\rm sbd}} (R_{C,i})^t K^{(i)}_C R_{C,i}$ Set-up for $\widetilde{K}_C$
Compute $\delta^{(i)}_I \leftarrow (\widetilde{A}^{(i)}_{II})^{-1} r^{(i)}_I$ Compute $r^{(i)}_\Gamma \leftarrow r^{(i)}_\Gamma - A^{(i)}_{\Gamma I} \delta^{(i)}_I$ Compute $r^{(i)} \leftarrow E^t r$ Compute $r^{(i)}_C \leftarrow {\Phi^{(i)}}^t r^{(i)}$	
Gather $r^{(i)}_C$	
Compute $x^{(i)}_F$ using (3)-(4)	Compute $r_C \leftarrow \sum_{i=1}^{n_{\rm sbd}} R^t_{C,i} r^{(i)}_C$ Solve $K_C z_C = r_C$
Scatter z_C into $z^{(i)}_C$, $i = 1, 2, \ldots, n_{\rm sbd}$	
Compute $s^{(i)}_{\rm C} \leftarrow \Phi^{(i)} z^{(i)}_C$ Compute $z^{(i)} \leftarrow E(s^{(i)}_{\rm F} + s^{(i)}_{\rm C})$ Solve $A^{(i)}_{II} z^{(i)}_I = -A^{(i)}_{I\Gamma} z^{(i)}_\Gamma$ $z^{(i)}_I := z^{(i)}_I + \delta^{(i)}_I$	

Table 1: Mapping of the PCG-BDDC algorithm to fine-grid and coarse-grid MPI tasks to achieve the maximum degree of overlapping in time.

6. Numerical experiments

The main goal underlying the numerical experiments section in this paper is to comprehensively assess, on state-of-the-art supercomputers, the weak scalability of the overlapped implementation of the two-level BDDC preconditioner equipped with the machinery that allows to inexactly solve the internal problems (i.e., computation of coarse-grid space basis, local Dirichlet, and constrained Neumann problems, and global coarse-grid problem) while still preserving preconditioner optimality (see [13] and Section 4). The benefit of such techniques has to be viewed in the light of future parallel architectures: the trend is that that the most scalable architectures (e.g., IBM BlueGene) will have more limited memory per core. *The study presented in the paper complements the mathematical analysis in [13] and answers how far can the overlapped/inexact BDDC codes go in the number of cores and the scale of the problem to still be within reasonable ranges of efficiency.*

This section is structured as follows. Section 6.1 briefly introduces the parallel codes, and the software/hardware stack of the supercomputers on which they are tested. In section 6.2 we comprehensively analyze the performance and scalability on JUQUEEN of the overlapped implementation of the inexact BDDC preconditioning codes when applied to a discrete 3D Poisson problem with uniform meshes and partitions, and constant physical coefficients. Section 6.2.1 in particular describes the target problem, and the mapping of the parallel codes to the underlying supercomputer. Prior to the actual raw weak scalability study, in Section 6.2.2, we evaluate the effect that the inexact solution of each internal problem in isolation has on preconditioning efficiency (i.e., number of PCG iterations). In view of the results of this evaluation, in Section 6.2.3, we define a set of inexact BDDC variants that differ in the particular solvers used for each internal problem, leading to different trade-offs among total computation time versus preconditioner efficiency. Then, the weak scalability of these variants is comprehensively studied in order to meet the objectives of the section. Finally, Section 6.3 evaluates the application on platform Marenostrum III of the algorithms/codes to the 2D and 3D (pressure) Poisson equations arising in the pressure segregation solution of the backward-facing step benchmark discretized with unstructured meshes split by means of automatic partitioners.

6.1. Code and parallel framework

The inexact/overlapped implementation of the BDDC preconditioner to be studied in this paper has been implemented in the FEMPAR (Finite Element 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 substructuring DD solvers [29]. The parallel codes in FEMPAR heavily use standard computational kernels provided by (highly-efficient vendor implementations of) the BLAS and LAPACK. 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 HSL_MI20 [38] software package for the approximate solution of these problems. HSL_MI20 is a *serial* implementation of the classical Ruge-Stüben AMG preconditioner (as described, e.g., in [39] and [40]) to be used as a convergence accelerator of Krylov subspace solvers. AMG preconditioners, while being less robust than sparse direct methods in general, are particularly well-suited for systems arising from the discretization of the Poisson problem. Indeed, AMG preconditioning leads to optimal convergence rates (i.e., independent of mesh characteristic size) with linear arithmetic/memory complexity for a number of applications.

The experiments reported in this section were obtained on a pair of PRACE Tier-0 supercomputers, namely MareNostrum III (MN-III), located in Barcelona (Spain), at the Barcelona Supercomputing Center (BSC), and JUQUEEN, located in Jülich (Germany) at the Jülich Supercomputing Center (JSC). MN-III is composed of 3,056 IBM dx360 M4 compute nodes interconnected by a FDR10 Infiniband high performance network. Each compute node is equipped with a pair of 8-core Intel E5-2670 SandyBridge-EP CPUs (i.e., 16 cores/node), running at 2.6GHz, and 32 GBytes of DDR3 memory. Each compute node runs a standard Linux SuSe Distribution (v11, SP3). On the other hand, JUQUEEN belongs to the last generation of IBM Blue Gene family of supercomputers, the so-called BG/Q supercomputer. JUQUEEN is configured as a 28-rack system, featuring a total of 28,672 compute nodes interconnected by an extremely low-latency five-dimensional (5D) torus interconnection network. Each compute node is equipped with a 16-core, 64-way threaded, IBM Power PC A2 processor, and 16 GBytes of SDRAM-DDR3 memory (i.e., 1GByte/core), and runs a lightweight proprietary CNK Linux kernel.

The codes were compiled either using the Intel Fortran compiler (14.0.2) or the IBM XLF Fortran compiler for BG/Q (v14.1) on MN-III and JUQUEEN, respectively, with recommended optimization flags. OpenMPI (1.8.1) and a customized MPICH2 were used for message-passing on MN-III and JUQUEEN, respectively. The codes were linked against the BLAS/LAPACK available on the single-threaded Intel MKL (11.1.2) and IBM ESSL library for BG/Q (v5.1), respectively, HSL_MI20 (v1.5.1), and HSL_MA87 (v2.1.1).

6.2. Inexact BDDC for 3D Poisson with structured meshes on JUQUEEN

6.2.1. Problem and parallel set-up

We consider as benchmark the solution of the Poisson problem on a rectangular prism $\overline{\Omega} = [0, 2] \times [0, 1] \times [0, 1]$ with homogeneous Dirichlet boundary conditions and a constant force term on the whole domain. A global conforming uniform mesh (partition) of $\overline{\Omega}$ into hexahedra is used for the trilinear FE discretization (i.e., Q1 FEs) of the continuous equation. 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 $m^3 = 2^3, 3^3, \ldots, 18^3$ compute nodes $(128, 432, \ldots, 93,312$ cores), with $4 \times 2 \times 2$ subdomains/MPI tasks per compute node and one MPI task per physical core.

An additional specialized MPI task is spawn in order to perform coarse-grid related computations. This task is mapped to an additional compute node, although it has only access to one core and 1 GByte of memory. This is due to limitations in the hardware/software stack of JUQUEEN, which does not allow to mix different execution modes on different compute nodes (e.g., 16 MPI tasks/1 thread per task on fine-grid nodes and 1 MPI task/16 threads per task on the coarse-grid node). Despite this, at first glance, severe restriction, we will demonstrate that these resources are already sufficient to solve very large-scale problems.[1]

The quotient among subdomain and mesh characteristic sizes, i.e., $\frac{H}{h}$, provides a measure of the local problem size. The number of FEs (i.e., hexahedra) 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 order to evaluate at which rate the computation time evolves with fixed $\frac{H}{h}$ and increasing number of cores (within the aforementioned range). As the trade-off among the factors determining the scalability of the codes depends on $\frac{H}{h}$, we perform the study with a pair of values of fixed problem size $\frac{H}{h} = 40$ and 60.

At this point, it is worth noting that an effort was done to set up the HSL_MI20 parameters to reach the fastest solution times. In particular, the same subset of values for these parameters as those considered in [38] were tested, with $\theta = 0.67$, RS1 coarsening, and Damped Jacobi smoothing being the winner combination for all internal problems. Besides, with this parameter-value combination, mesh independent convergence rates were achieved.

6.2.2. The impact of approximately solving the internal problems

In this section we evaluate the effect that the inexact solution of each internal problem in isolation has on the efficiency of the BDDC preconditioner. The objective of this section is two-fold. First, to confirm experimentally the results of the mathematical analysis presented in [13] and Section 4. Special attention will be paid on whether preconditioner optimality is preserved (i.e., number of PCG iterations asymptotically constant for fixed local problem size and increasing number of subdomains) no matter which of the internal problems is perturbed. Second, to determine to what extent there is margin for improvement (in terms of number of PCG iterations) by the usage of more accurate solvers for the internal problems. In such cases, it might be possible in practical scenarios to reach a trade-off among total computation time and preconditioner efficiency which leads to a faster solution of the problem.

Figures 2, 3, and 4 compare the number of PCG iterations of the exact BDDC preconditioner with that of its inexact variant, for BDDC(c), BDDC(ce) and BDDC(cef), respectively. Each figure provides the impact that the perturba-

[1] One way to deal with this restriction would be to distribute the coarse-grid problem among several MPI coarse-grid tasks, possibly spanning multiple compute nodes. This is not explored here, but left as future work.

tion of each internal problem in isolation has on the number of PCG iterations. For example, Figures 2 (a), 3 (a), and 4 (a) are focused on the impact of the inexact solution of the coarse-grid problem, while the rest of internal problems are solved exactly. The same applies for (b), (c), and (d), but for the solution of the Dirichlet problem, computation of coarse-grid basis, and solution of the constrained Neumann problem, respectively. For each experiment, three different inexact solvers were considered, namely "AMG(1)", "AMG(2)", and "AMG(4)", which stand for a single, two and four AMG cycles, respectively. The more AMG cycles, the more accurate the solution of the corresponding internal problem is expected to be, resulting in a beneficial impact on the number of inexact BDDC-PCG iterations. In Figures 2, 3, and 4, the global problem size was scaled linearly with the number of subdomains to keep a local problem size of $H/h = 40$, i.e., 64K FEs per core; this is the largest local problem size that can be solved provided that the exact BDDC preconditioner implementation is based on sparse direct solvers, and the 1GByte/core memory constrain on JUQUEEN. The results obtained with smaller local problem sizes ($H/h = 10, 20, 30$) are omitted for brevity; similar conclusions to the ones with $H/h = 40$ can be raised.

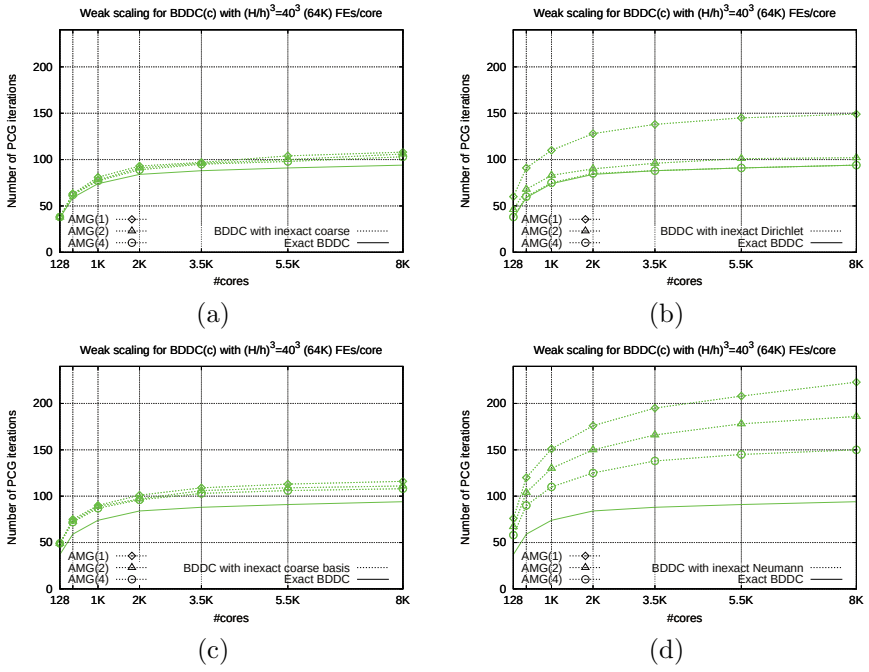


Figure 2: Sensitivity of the number of outer BDDC(c)-PCG iterations in the presence of perturbations in the solution of the (a) coarse, (b) Dirichlet, (c) coarse-grid basis, and (d) Neumann problems. Three different internal solvers, AMG(1), AMG(2) and AMG(4), were tested for the solution of these problems.

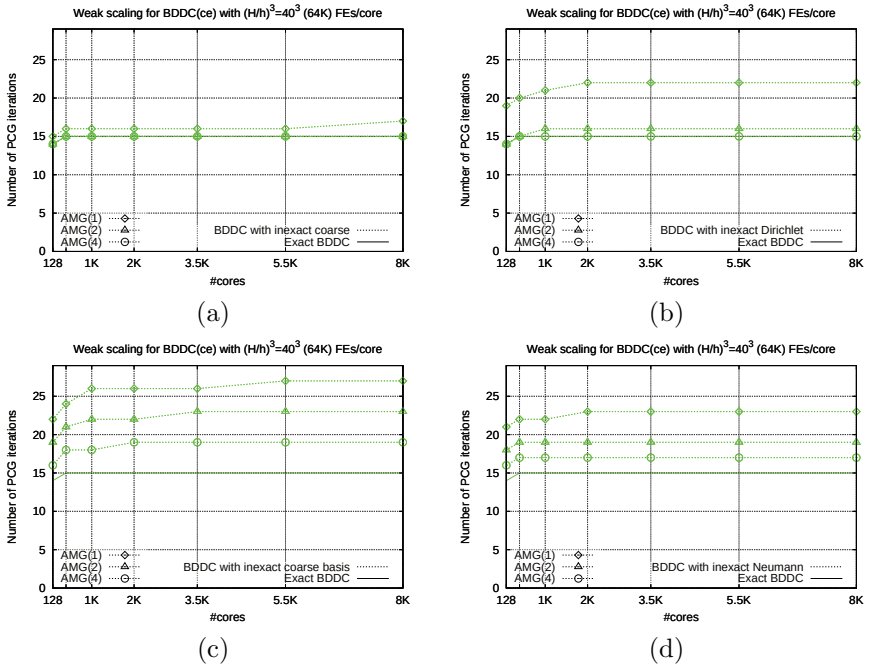


Figure 3: Sensitivity of the number of outer BDDC(ce)-PCG iterations in the presence of perturbations in the solution of the (a) coarse, (b) Dirichlet, (c) coarse-grid basis, and (d) Neumann problems. Three different internal solvers, AMG(1), AMG(2) and AMG(4), were tested for the solution of these problems.

Figures 2, 3, and 4 overall confirm the mathematical analysis in [13] and Section 4. In particular, provided that the inexact BDDC preconditioner is equipped with spectrally equivalent approximations of the Neumann and coarse problems, and a spectrally equivalent kernel preserving approximations of the Dirichlet problem and the constrained Neumann problems at the computation of the coarse basis functions, preconditioner optimality is preserved. This can be observed in Figures 2, 3, and 4 by the number of PCG iterations being asymptotically constant no matter which of the internal problems is perturbed, and to what extent it is perturbed. While this is true, it is also worth noting that the impact that the inexact solution of the internal problems has on the number of PCG iterations is highly depending on the constraints considered for the coarse space, and the internal problem being perturbed. To see this, the y-axis of the four plots in Figures 2, 3, and 4 were scaled accordingly to the one corresponding to the internal problem whose perturbation has the largest impact on the number of PCG iterations.

If we focus on Figures 2 (a) and (c), it can be observed that the inexact solution/computation of the coarse-grid problem/coarse-grid basis has a very mild impact on the number of PCG iterations. Indeed, the number of PCG

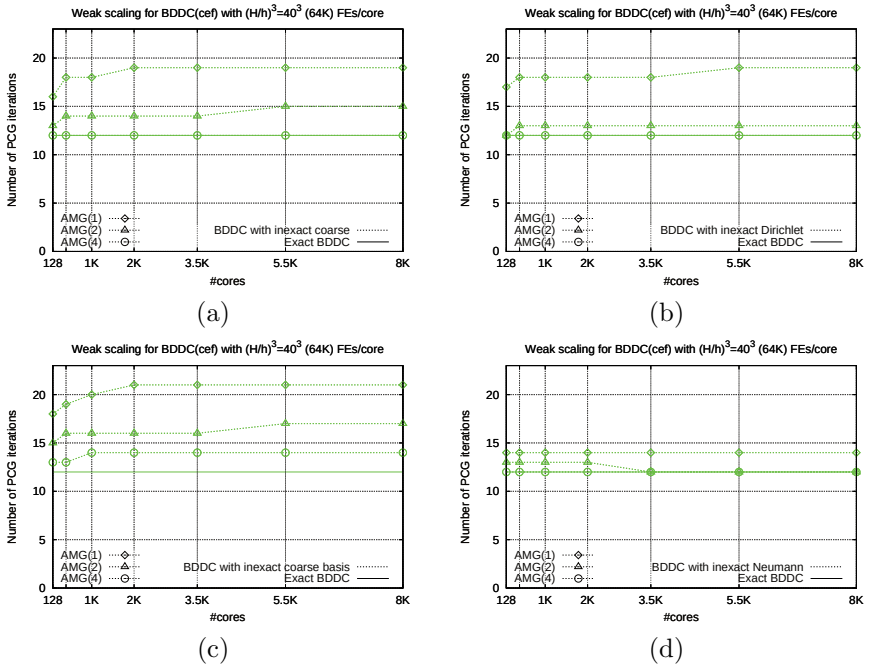


Figure 4: Sensitivity of the number of outer BDDC(cef)-PCG iterations in the presence of perturbations in the solution of the (a) coarse, (b) Dirichlet, (c) coarse-grid basis, and (d) Neumann problems. Three different internal solvers, AMG(1), AMG(2) and AMG(4), were tested for the solution of these problems.

iterations of the inexact BDDC(c) preconditioner with a single AMG cycle is very close to that of the exact BDDC(c). Increasing the number of AMG cycles leads, as expected, to a reduction of the number of PCG iterations, which is almost negligible in this case. We consider this a very nice property of the BDDC preconditioning approach provided that the coarse-grid problem is the main scalability bottleneck, so that the overall solution approach immediately benefits from any savings in memory/time that can be achieved in this part of the algorithm. However, Figures 2 (b) and (d), reveal a high impact of the inexact solution of the Dirichlet and Neumann problems, respectively, with a higher impact in the latter case. For example, with AMG(1), a roughly 50% and 100% increase in the number of PCG iterations with respect to the exact BDDC(c) preconditioner is observed, respectively. With additional AMG cycles for the approximation of the Dirichlet problem, the preconditioner efficiency of the exact BDDC(c) preconditioner can be rapidly recovered. However, in the case of the Neumann problem, still a 50% increase is observed with AMG(4).

The picture is quite different in the case of the BDDC(ce) preconditioner. While the inexact solution of the coarse-grid problem has a very mild impact on the number of PCG iterations (see Figure 3 (a)), the inexact computation of the

coarse-grid basis has the highest impact (a roughly 80% increase in the number of PCG iterations for AMG(1)) among all internal problems. The impact of the inexact solution of the Dirichlet and Neumann problems is milder than that of the inexact computation of the coarse-grid basis, and similar to each other, with a 50% increase for AMG(1) (compare Figures 3 (b) and (c)).

Figures 4 (b) and (c) reveal a very close response of the BDDC(cef) preconditioner to that of the BDDC(ce) preconditioner in the presence of perturbations of the Dirichlet problem and computation of the coarse-grid basis, respectively. However, Figure 4 (a) reveals higher sensitivity of the BDDC(cef) preconditioner under perturbations in the solution of the coarse-grid problem.[1] Indeed, with AMG(1), a 60% increase in the number of PCG iterations is observed with respect to the exact BDDC(cef) preconditioner. On the other hand, Figure 4 (d) reveals a very mild impact of the inexact solution of the Neumann problem, with AMG(1) already almost recovering the preconditioner efficiency of the exact BDDC(cef) preconditioner.

6.2.3. Scalability of the overlapped implementation with inexact solvers

In the previous section it has been shown that there is margin for improvement (at least in terms of the number of PCG iterations) by the usage of a more accurate solver than AMG(1) for the internal problems. In light of this observation, Table 2 presents a set of four selected inexact variants of the BDDC preconditioner. The columns labeled as "Φ", "Dirichlet", "Neumann", and "Coarse" refer to the computation of the coarse-grid basis vectors, Dirichlet, Neumann, and coarse-grid internal problems, respectively. "AMG(1)", and "AMG(2)" stand for a single, and a pair of AMG cycles, respectively.

	Φ	Dirichlet	Neumann	Coarse
Var. 1	AMG(1)	AMG(1)	AMG(1)	AMG(1)
Var. 2	AMG(1)	AMG(2)	AMG(1)	AMG(1)
Var. 3	AMG(2)	AMG(1)	AMG(2)	AMG(1)
Var. 4	AMG(2)	AMG(2)	AMG(2)	AMG(1)

Table 2: A set of four selected inexact variants of the two-level BDDC method.

We stress that the inner solver combinations that are shown in Table 2 are not the only ones possible, but the ones that have been selected from a much wider set after comprehensive experimentation. First, we observed that it does not pay off a more accurate solver for the coarse-grid problem (e.g., AMG(2) or even an internal PCG-AMG iteration), as the reduction of the number of outer PCG iterations did not compensate for the decreased scalability at large core counts caused by a most costly solution of the coarse-grid problem. Second, we

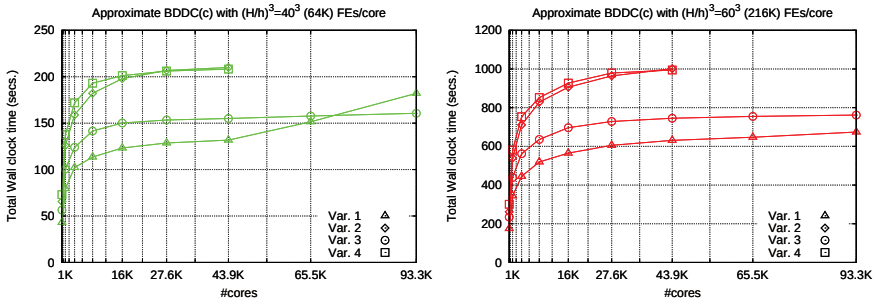
[1] We remind that the BDDC(cef) coarse matrix is denser than the one for BDDC(ce). The AMG approximation seems to be less effective due to this fact.

also considered variants where $\widetilde{K}_{rr} \neq \widehat{K}_{rr}$, e.g., AMG(2) for the Neumann problem, and AMG(1) for the computation of the coarse-grid vectors and vice versa. As stated in Remark 4.4, under this scenario one must compute $(\widetilde{K}_{rr}^{(i)})^{-1} C_{r,i}^t$ to preserve symmetry, instead of re-using $(\widehat{K}_{rr}^{(i)})^{-1} C_{r,i}^t$ from the constrained Neumann problem required for the computation of the coarse-grid basis vectors. This involves the solution of *an extra linear system with multiple right-hand sides*, as many as local coarse constraints, during preconditioner set-up. We experimentally observed that this extra computation significantly outweighs any gain derived from the usage of such variants.

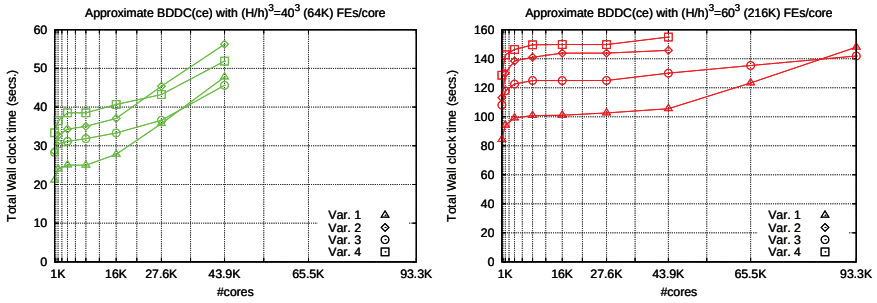
A pair of details underlying the inexact variants in Table 2 are worth noting. First, the coarse-grid problem was built using the Galerkin projection of K onto the inexact coarse basis $\widehat{\Phi}$ (see (11)) instead of the approach used in [13], that builds the coarse-grid problem as $\widehat{\Phi}^t \widehat{K} \widehat{\Phi}$ (see (12)). We consistently observed that the former approach leads at most to the same number of PCG iterations than the latter, with up to a 25% reduction in some cases (in particular, with the inexact BDDC(cef) preconditioner and the largest local problem size of $H/h = 60$). Second, although Var 3. and 4 put more effort than Var. 1 and 2, respectively, in the (more accurate) computation of the coarse-grid basis vectors and Neumann problem, note that in Var. 3 and 4 there is more potential for overlapping fine-grid and coarse-grid computations, in particular during preconditioner application at the bottom-most overlapping area of Table 1. We next study to what extent this property of Var. 3 and 4 leads to increased scalability and reduced computation times compared to those of Var. 1 and 2.

Figures 5 (a), (b), and (c) provide a comparative view of the weak scalability for the total computation time (in seconds) for the inexact variants of the BDDC(c), BDDC(ce) and BDDC(cef) solvers, respectively. The local problem sizes went from 40^3=64K (left side) to 60^3=216K FEs per core (right side), while the number of cores from 168 to 93,312 (see Section 6.2.1). For those variants, and local problem size combinations where a "high" degradation in the weak scalability was already observed up to 43.9K cores, we did not run the codes beyond because we were limited in the consumption of the underlying parallel resources. On the other hand, Figures 6 (a), (b), and (c) report the number of PCG (outer solver) iterations for the same variants, and local problem size combinations in Figure 5. We set the initial solution vector guess $x_0 = 0$ for the outer iterations, that were stopped whenever the residual r_k at a given iteration k satisfies $\|r_k\|_2 \leq 10^{-6} \|r_0\|_2$.

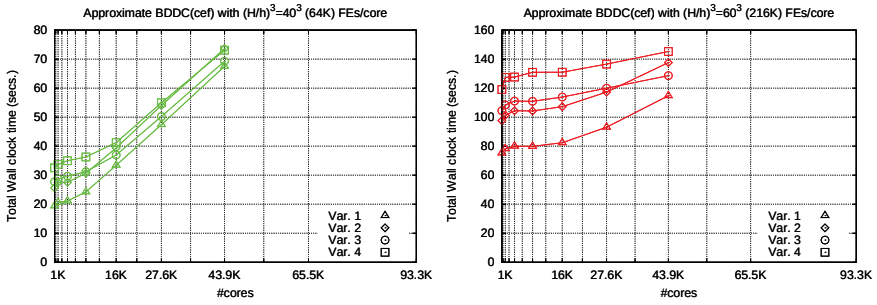
The shape of the different scalability curves shown in Figure 5 depends on the particular balance among fine-grid and coarse-grid computations achieved, for each inexact variant, in each of the three overlapping areas shown in Table 1, together with the preconditioner efficiency achieved by each variant, which determines the number of external outer solver iterations. For example, for the inexact BDDC(ce), and BDDC(cef) variants, and a load per core of 64K FEs/-core, the total computation time becomes dominated by the coarse-grid solver beyond 16K and 8K cores, respectively, rendering the overlapping technique less successful (i.e., given such load per core there is a limited potential for overlap-



(a)

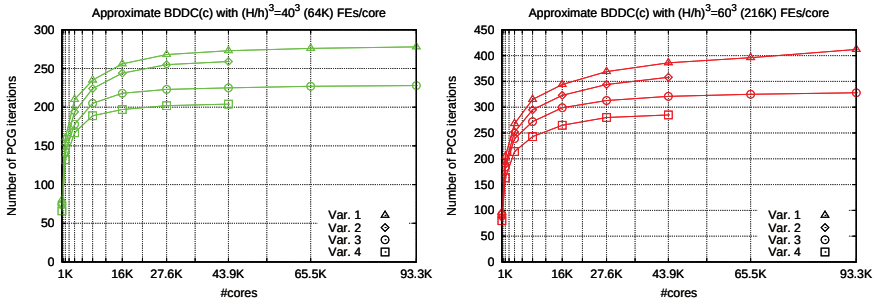


(b)

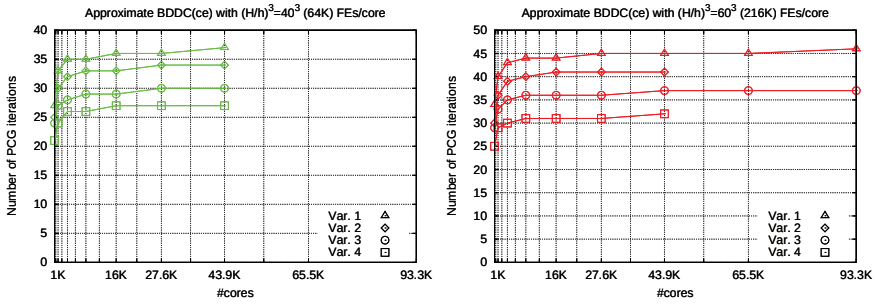


(c)

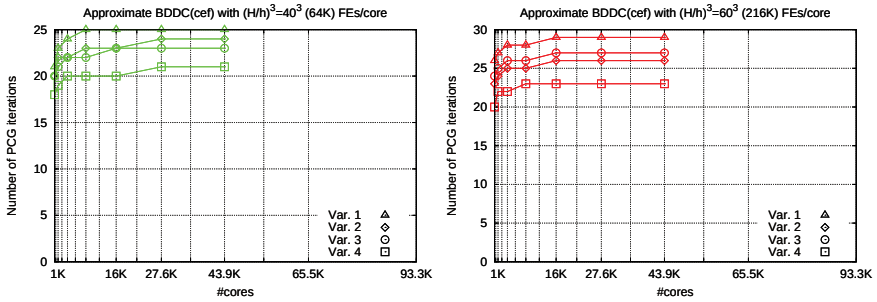
Figure 5: Weak scalability for the total computational time of the inexact variants of (a) BDDC(c), (b) BDDC(ce), and (c) BDDC(cef) solvers for the 3D Poisson problem on JUQUEEN. Left: $\frac{H}{h} = 40$. Right: $\frac{H}{h} = 60$. The solution of the coarse-grid linear system was mapped to an additional blade.



(a)



(b)



(c)

Figure 6: Weak scalability for the number of PCG iterations for the inexact variants of (a) BDDC(c), (b) BDDC(ce), and (c) BDDC(cef) solvers for the 3D Poisson problem on JUQUEEN. Left: $\frac{H}{h} = 40$. Right: $\frac{H}{h} = 60$.

ping). This can be observed in the left-hand side of Figures 5 (b), and (c) by the total computation time of all variants for increasing number of cores. As expected, the degradation in the weak scalability is linear (with a higher slope in the case of inexact BDDC(cef) due to a larger, with denser stencil, coarse-grid problem) with the number of subdomains, given the linear arithmetic complexity of AMG preconditioning.

However, as the local problem size is increased, overlapping of fine-grid/coarse-grid duties becomes progressively more successful in tackling the bottleneck associated to the coarse-grid problem. For example, for inexact BDDC(c), Var. 3, and a load per core of 64K FEs per core, and for all variants in the case of the largest load per core of 216K FEs/core, the weak scalability is solely determined by how fast the outer preconditioner solver achieves asymptotically constant converge rates with fixed problem size and increasing number of cores, meaning that the overlapping technique was successful to completely overlap coarse-grid related computations in the 128-93.3K cores range; see the right-hand side of Figures 5 (a), and 6 (a). As shown in the right-hand side of Figure 5 (b), the same holds for inexact BDDC(ce), but in the 128-43.9K cores range; a similar observation can be made for inexact BDDC(cef) with tighter core ranges in Figure 5 (c), due to a larger, with denser stencil, coarse-grid problem for the latter algorithm. Beyond 43.9K cores, the computation time of coarse-grid problem related computations in overlapping areas #1 and #2 (see Table 1) starts exceeding that of the fine-grid related computations for the inexact BDDC(ce) variants, justifying the (very) mild degradation of roughly 20% and 50% that is observed for Var. 3 and 1, respectively, in the 43.9K-93.3K cores range, rendering a third level in the hierarchy necessary.

If we now turn our attention to Figures 6 (a), (b), and (c), we can observe that Var. 4 is consistently the one that leads to a smaller number of iterations, followed by Var. 3, 2 and 1. This certainly makes sense given that Var. 4 is the one that solves more accurately all internal problems. However, in terms of computation times, and *provided that the total computation times are dominated by fine-grid related computations*, the relative rank of the variants subject of study change, with Vars. 1 and 3 being the faster, and Var. 4 the slowest; see Figures 5 (a), (b), and (c).

An interesting observation can be made, e.g., from the left hand side of Figure 5 (a) and Figure 5 (b), where Var. 3 becomes faster than Var. 1 for "sufficiently large" core counts, even with the extra computation time incurred by AMG(2) in the computation of the coarse-grid basis and the solution of the Neumann problem. As mentioned above, Var. 3 puts more computational effort in the solution of the Neumann problem. This increases the potential of the implementation to fully overlap the solution of the coarse-problem at each preconditioner application during the PCG phase, resulting in increased scalability.

Further, it is very important to note the overall effect of a more accurate computation of the coarse system matrix on scalability. When the coarse basis is computed with AMG(2) (Vars. 3 and 4) instead of AMG(1) (Vars. 1 and 2), the scalability loss is much less severe. As expected, this degradation is linear,

due to the linear complexity of AMG. However, the slope is noticeably worse for Vars. 1 and 2 than Vars. 3 and 4. As a conclusion, *for large core counts, the reduced number of PCG iterations, increased scalability, and lower degradation make Var. 3 the winning choice at large core counts.*

To conclude our study, we report in Table 3, for the exact and inexact variants of the BDDC preconditioner, and increasing values of the local problem size $\frac{H}{h}$, the memory consumption figures of the fine-grid preconditioning level; Table 4 reports those of the coarse-grid preconditioning level. [1] The exact variant was supplied with HSL_MA87 [41], a highly-efficient *parallel multi-threaded* DAG-based code implementation of the supernodal sparse direct Cholesky solver. We note that the figures reported in Tables 3 and 4 correspond to the amount of (permanent) memory consumed by the preconditioner once it has been computed, and not to the (temporary) memory used during its computation.

		$\frac{H}{h}$				
Solver	Var.	10	20	30	40	60
BDDC(c)	Inexact	20.8M	38.2M	81.2M	157.1M	516.5M
	Exact	21.9M	64.2M	218.4M	613.6M	O.M.
BDDC(ce)	Inexact	22.6M	42.3M	86.7M	158.9M	522.2M
	Exact	21.7M	64.0M	219.9M	618.8M	O.M.
BDDC(cef)	Inexact	22.6M	42.3M	86.7M	160.6M	527.5M
	Exact	21.7M	64.3M	219.6M	625.2M	O.M.

Table 3: Memory consumption of the highest memory consuming fine-grid task for the exact and inexact variants of the 2-level BDDC preconditioner. O.M.: out of memory.

	m (#subdomains=$16m^3$)						
Solver	6	8	10	12	14	16	18
Inexact BDDC(c)	17.1M	23.1M	33.9M	51.5M	63.5M	94.2M	133.9M
Inexact BDDC(ce)	30.3M	56.2M	100.3M	167.8M	263.5M	392.7M	582.7M
Inexact BDDC(cef)	54.9M	118.9M	228.5M	396.6M	604.0M	O.M.	O.M.

Table 4: Memory consumption on coarse-grid task for the inexact variant of the 2-level BDDC preconditioner. O.M.: out of memory.

As expected, the inexact variant of the BDDC preconditioner is less memory demanding than the exact one. This is clearly justified by the linear order of memory complexity of AMG solvers that the inexact variant fully exploits. In absolute terms, it consumes a moderate amount of memory, with roughly half a GByte for the largest local problem size. This is only a 50% of the memory available on JUQUEEN, meaning that larger problem sizes can still be solved on this machine (i.e., we did not still reach the memory limits of the proposed solver machinery), enabling improved scalability results. The (mild) increase of

[1] Memory consumption was obtained on JUQUEEN by a call to `malloc_stats` right after preconditioner set-up.

memory consumption with additional constraints can be easily explained by the fact that extra number of coarse-grid basis vectors have to be stored in memory. On the other hand, if we take a look at the memory consumption of the coarse-grid solver in Table 4 we observe, as expected, a moderate (linear) increase with the number of subdomains, with the higher the number of constraints, the higher the slope. In absolute terms, we can observe that for the largest number of subdomains tested (i.e., 93.3K), the inexact BDDC(ce) preconditioner consumed roughly a 58% (i.e., 582.7MBytes) of the memory available, meaning that the solver machinery proposed can still solve larger problems on larger number of subdomains.

6.3. Inexact BDDC with a complex domain and unstructured meshes on MN-III

In this section we apply the overlapped implementation of the inexact BDDC preconditioner to the solution of the Poisson problem on unstructured meshes, which is required for the numerical simulation of the (turbulent) incompressible flows using typical pressure segregation techniques [26, 27]. In particular, we analyze the applicability of the inexact BDDC method to the solution of the pressure Poisson equation in the classical benchmark of the incompressible flow over backward facing step. Unstructured FE meshes are split by means of an automatic mesh (actually dual graph) partitioner. The combination of these factors, which to some extent increases the degree of irregularity to be addressed (e.g., automatic partitioners typically lead to an irregular decomposition of the domain into subdomains), posses a challenge to the techniques used to precondition the local Dirichlet/Neumann problems, and the global coarse-grid one (in our case, classical Rüge-Stuben AMG), and therefore to the inexact BDDC preconditioning approach as a whole. We stress, however, that the purpose of the section is not to comprehensively assess the weak scalability of the code as we did with the structured test case in Section 6.2, but to grasp what to expect from the algorithmic machinery subject of study when applied to more complex test cases. A more comprehensive assessment would require, on the one hand, to consider a wider range of test cases. 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. Nevertheless, we will be able to compare the performance of both the exact and inexact variants provided that the exact BDDC preconditioner fits into available memory for the number of cores and scale of the problems considered here (in contrast to the experiments in Section 6.2).

6.3.1. 2D experiments

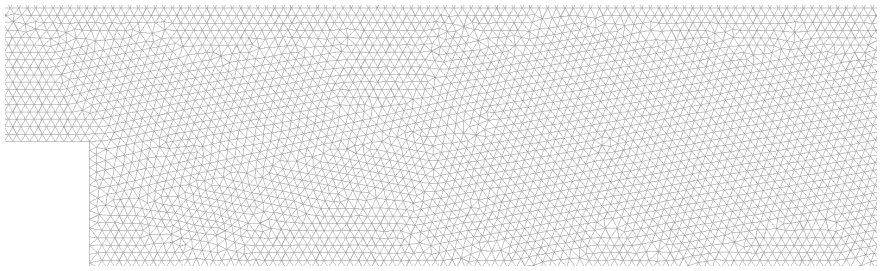
The codes subject of study were applied to the linear FE discretization (i.e., $P1$-elements) of the 2D Poisson problem on the backward-facing step domain depicted in Figure 7. Boundary conditions are set to homogeneous Dirichlet on the whole boundary except for the left-most (perpendicular to the x-axis) edge of the boundary, where the unknown is constrained to be equal to a constant function $g_D = 1$ (i.e., non-homogeneous Dirichlet boundary conditions). Finally, we set $f = 1$ (i.e., constant force term) on the whole domain.

We have considered four local problem sizes of L1 ≈ 20.9K, L2 ≈ 39.2K, L3 ≈ 80K, and L4 ≈ 157.8K triangles per subdomain, respectively. For every local problem size, we have considered four meshes, corresponding to 64, 256, and 1024 subdomains. All meshes were partitioned on a shared-memory multiprocessor with 256GBytes of main memory using the multilevel graph partitioning algorithms available in METIS 5.1.0 [42]. The (irregular) partition (resulting from METIS) of the mesh depicted in Figure 7 (b) into 64 parts is shown in Figure 7 (a) using colors to represent subdomains

	Load per core							
	L1		L2		L3		L4	
#cores	#nodes	#FEs	#nodes	#FEs	#nodes	#FEs	#nodes	#FEs
16	168K	334K	316K	628K	643K	1.28M	1.26M	2.52M
64	671K	1.34M	1.26M	2.51M	2.56M	5.12M	5.07	10.1M
256	2.68M	5.35M	5.03M	10.0M	10.3M	20.5M	20.2M	40.4M
1024	10.7M	21.4M	20.1M	40.2M	41M	82.0M	80.7M	161M

Table 5: Number of nodes and triangles (#FEs) in the unstructured computational meshes used for the scaling study. These meshes were partitioned into 16, 64, 256, and 1024 subdomains using METIS 5.1.0 [42], such that four different computational loads per subdomain of L1 ≈ 20.9K, L2 ≈ 39.2K, L3 ≈ 80K, and L4 ≈ 157.8K triangles per subdomain are considered for the study.

(a)



(b)

Figure 7: (a) 2D backward facing step domain and partition into 64 subdomains using METIS. (b) Zoomed view of the unstructured mesh for the area delimited by the grey coloured contour in Figure 7 (a).

In Table 6 we compare the number of PCG iterations of the (exact) 2-level BDDC preconditioner with that of the four variants of the inexact preconditioner sketched in Table 2. The number of cores and size of the problem were

scaled as described in Table 5. An effort was done to set up the HSL_MI20 parameters to reach the best memory/time trade-offs for this particular problem, with $\theta = 0.25$, RS1 coarsening, and Damped Jacobi smoothing being the winner combination for all internal problems (see [38] for details).

The comparison in Table 6 reflects, on the one hand, that the exact 2-level BDDC preconditioner reaches faster (with the number of cores) an asymptotically constant number of iterations compared with any of the four inexact variants. On the other hand, one may observe that, as expected, the asymptotically constant number of iterations reached is smaller for BDDC than for inexact BDDC. In any case, we stress that the difference among the number of iterations of the exact and inexact BDDC preconditioners does not depend on the number of cores, provided that all methods subject of evaluation finally reach an asymptotically constant number of iterations. Besides, we consider a remarkable property of the inexact variants that such moderate loss of preconditioner robustness came at the benefit of a reduced order of arithmetic and memory complexity (in particular, $O(n^{1.5})$ order of arithmetic complexity of sparse direct solvers versus $O(n)$ for AMG during preconditioner set-up, and $O(n\log n)$ versus $O(n)$ during preconditioner application).

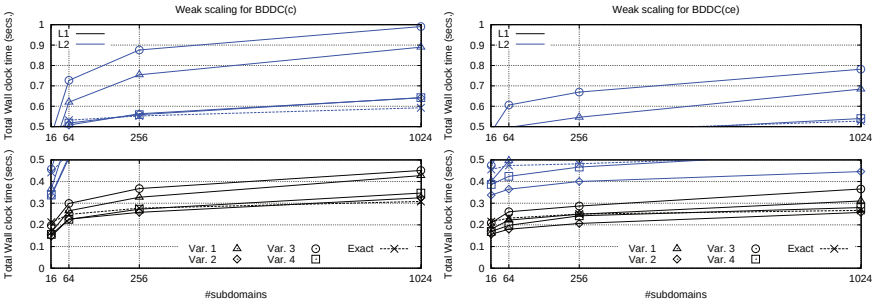
If we now turn our attention into the relative merits of the inexact variants in Table 6, we can first observe that Var 1. and Var 3. converge in (almost) the same number of iterations. This observation reveals that a more accurate computation of the coarse-grid basis vectors, and the solution of the Neumann problem at each iteration (AMG(2) for Var. 3 versus AMG(1) for Var. 1), has very little impact on preconditioner robustness for this particular problem. This can also be confirmed by comparing the number of iterations of Vars. 2 and 4, which are also (almost) coincident. However, if we compare the number of iterations of Vars. 1. and 3, to those of Vars. 2. and 4, we can observe a significant reduction in the number of iterations for the latter two variants, revealing that, for the problem subject of study, there is margin for improvement (in terms of the number of iterations) by a more accurate solution of the Dirichlet problem (AMG(2) for Vars. 2 and 3 versus AMG(1) for Vars. 1 and 4). The higher the load per core, and the number of cores, the larger the improvements.

Figure 8 reports the weak scalability for the computational time of the overlapped implementation of the (exact) 2-level BDDC preconditioner (see [12]) and those of inexact variants, with L1 and L2 (Figure 8 (a)), and L3 and L4 as fixed loads per core (Figure 8 (b)). The y-axis of the plots in the right part of Figs. 8 (a) and (b) was scaled to match the corresponding ones in the left part to simplify the comparison among BDDC(c) and BDDC(ce)-based variants. At this point, it is important to stress that, for all preconditioners subject of study, and all loads per core explored, the time spent in the coarse-grid problem could be fully overlapped in the full 16-1024 cores range. This observation justifies why any exact or inexact variant equipped with corner and edges constraints is always faster than its counterpart equipped with only corner constraints (compare the left and right parts of Figure 7, and complement this comparison with the number of iterations of BDDC(c) versus BDDC(ce) in Table 7). Overall, in Figure 7, remarkable time scalability is observed in the 16-1024 cores range

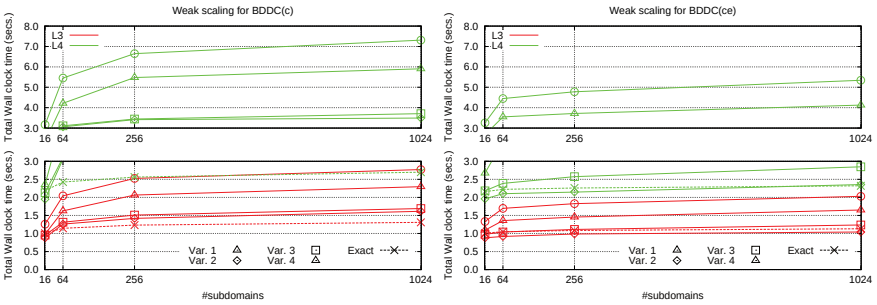
Solver		Var.	Load per core															
			L1				L2				L3				L4			
			#cores				#cores				#cores				#cores			
			16	64	256	1K	16	64	256	1K	16	64	256	1K	16	64	256	1K
BDDC(c)	Exact	*	10	14	16	17	10	14	16	17	11	15	18	18	12	16	18	19
	Inexact	1	20	34	40	41	23	41	49	50	28	48	60	60	31	58	75	76
		2	13	21	23	24	15	24	26	26	17	25	28	29	17	28	31	30
		3	21	33	38	40	25	41	48	49	30	50	61	61	35	62	75	77
		4	12	18	21	24	13	21	22	23	16	22	25	26	16	24	26	27
BDDC(ce)	Exact	*	8	8	8	8	9	8	8	8	9	8	9	9	9	9	9	9
	Inexact	1	21	25	26	27	24	29	31	32	30	36	38	39	34	45	46	48
		2	13	14	15	16	14	14	15	15	16	15	16	16	16	16	16	17
		3	21	25	25	27	24	30	32	33	30	37	39	40	34	46	48	49
		4	12	13	15	15	14	14	15	16	15	14	15	16	15	15	16	17

Table 6: Weak scalability for the number of PCG iterations of the exact and four inexact variants (see Table 2) of the BDDC(c), and BDDC(ce) preconditioners for the 2D Poisson problem on a backward facing step domain (see Figure 7).

despite the high irregularity of the problem at hand.



(a) Loads per core L1 and L2.



(b) Loads per core L3 and L4.

Figure 8: Weak scalability on MN-III for the total computation time of the exact and four inexact variants (see Table 2) of the BDDC(c) (left) and BDDC(ce) (right) preconditioners for the 2D Poisson problem on a backward facing step domain (see Figure 7).

If we compare the timings of the four inexact variants in Figure 8, we can recurrently observe that Var. 2 is the fastest variant, (closely) followed by Var. 4, then (further) by Var. 1, and, finally, (further) by Var. 3. This should not be surprising at first glance provided that: (1) Vars. 2 and 4 took significantly less iterations to converge than Vars. 1 and 3 (see Table 6); (2) Vars. 4 and 3 perform an extra AMG cycle in the computation of the coarse-grid basis vectors and the solution of the Neumann problem at each PCG iteration (compared to Vars. 2 and 1, respectively). However, note that despite Var. 2 converging in less iterations than Var. 1, Var. 2 involves the application of an extra AMG cycle for the solution of the (pre and post) Dirichlet corrections at each PCG iteration. Therefore, we conclude that a (remarkable) trade-off was reached such that the reduction in the number of iterations incurred by this extra pair of AMG cycles (at each iteration) more than compensated the extra time required for their application (at each iteration).

By comparing the computation time of the fastest inexact variant, i.e., Var. 4, to that of the exact BDDC preconditioner, we can observe that, in most cases, both methods achieved a similar trade-off among the number of iterations and time spent in the solution of the internal problems. This reflects that, for the range of local problems subject of study, any gain derived from a less costly set of inner solvers for the inexact variant was outweighed by the extra number of outer iterations resulting from the lost of robustness. (We expect, however, significant gains from the inexact solver as far as the local problem size is scaled further.) In any case, the inexact variants saved significant memory compared to the exact counterparts, due to the reduced order of memory complexity of AMG solvers over sparse direct solvers. For example, for L4 and 1K cores, the exact BDDC(ce) preconditioner consumed 411 MBytes on the highest memory demanding fine-grid task, while the inexact variants, only 309 MBytes.

6.3.2. 3D experiments

In this section we apply the inexact BDDC preconditioner to the solution of the three-dimensional extension of the problem targeted in Section 6.3.1. The computational domain of this problem is shown in Figure 9, together with one of the unstructured computational meshes of tetrahedra used in the scaling study, and its partition into 64 subdomains.

Nine meshes were generated keeping the ratio between the total number of FEs and the number of subdomains approximately equal to 20K, for 128, 256, 512, 1024, and 2048 subdomains, respectively. In Table 7 we show the minimum, maximum, and average number of nodes per subdomains. We can observe that the maximum number of nodes slightly increases, but reaches an asymptotical regime. This also applies to other factors with a (significant) impact on performance, such as the average number of coarse DoFs per subdomain. On the other hand, we were limited to a more moderate load per core than those considered in Section 6.3.1, given constraints inherent to the (mesh partitioning) code, and limited memory/disk capacity available on the underlying shared-memory computer (recall that the code we are using for mesh partitioning relies on METIS, i.e., no distributed-memory parallelization). The load per core considered is the

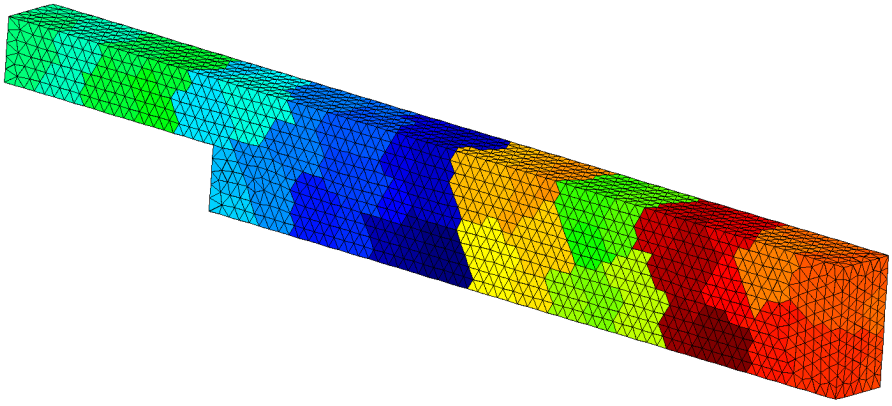


Figure 9: 3D backward facing step domain, unstructured computational mesh of tetrahedra and its partition into 64 subdomains using METIS.

largest one from those in Section 6.3.1 such that the code is able to partition the meshes required to perform the scalability study up to 4K cores.

Table 7 compares the number of PCG iterations of the (exact) 2-level BDDC preconditioner with that of the four inexact BDDC variants subject of study. The same set of values for HSL_MI20 parameters to those considered in Section 6.3.1 were considered here. To keep the presentation shorter, we omitted the results corresponding to BDDC(c)-based preconditioners. Similar observations to those already made in Section 6.2 can be made in this case, with BDDC(c)-based preconditioners being (significantly) less robust than BDDC(ce/cef)-based ones.

Solver		Var.	#cores								
			16	32	64	128	256	512	1K	2K	4K
BDDC(ce)	Exact	*	13	16	16	14	16	18	20	21	23
	Inexact	1	19	22	24	23	27	35	45	49	52
		2	16	20	21	19	23	28	35	37	39
		3	20	22	24	24	28	35	46	51	53
		4	16	18	19	18	21	27	34	37	39
BDDC(cef)	Exact	*	13	16	16	13	16	17	19	21	23
	Inexact	1	19	22	24	23	27	35	44	49	52
		2	16	20	22	21	25	31	37	41	40
		3	20	22	24	24	28	35	45	50	53
		4	16	19	22	20	23	31	36	40	41

Table 7: Weak scalability for the number of PCG iterations of the exact and four inexact variants (see Table 2) of the BDDC(ce) and BDDC(cef) preconditioners for the 3D Poisson problem on a backward facing step domain (see Figure 9).

Overall, very close observations to those made for Table 6 are derived from Table 7. The exact 2-level BDDC preconditioner reaches faster (with the number

of cores) an asymptotically constant number of iterations compared with any of the four inexact variants, and besides, the constant reached is smaller for the former. We stress that this difference becomes asymptotically constant with the number of cores (confirming preconditioner optimality for the inexact variants). For the load per core explored, a maximum increase by a factor of 2.3 and 1.7 is seen for Vars. 1 and 3, and 2 and 4, respectively. If we compare the inexact variants, we can also see that a more accurate solution of the Dirichlet problem has a significant positive impact on preconditioner quality (compare Vars. 2 with 1, and 4 with 3), while it does not for the computation of the coarse-grid basis vectors and the constrained Neumann problem at each iteration (compare Vars. 3 with 1, and 4 with 2). Finally, the results of Table 7 also confirm that, for both the exact and inexact variants, there is no gain derived from the addition of face constraints into the coarse-grid space, as the number of iterations of the BDDC(cef)-based preconditioners was (at most) the same as that of the BDDC(ce)-based counterparts.

Figure 10 reports the weak scalability for the computational time of the overlapped implementation of the (exact) 2-level BDDC(ce) (left) and BDDC(cef) (right) preconditioners (see [12]) and those of the inexact variants, with a moderate load of 20K FEs/core. The y-axis of the plot in the right part of Fig. 10 was scaled to match the corresponding one in the left part to simplify the comparison among BDDC(ce) and BDDC(cef)-based variants.

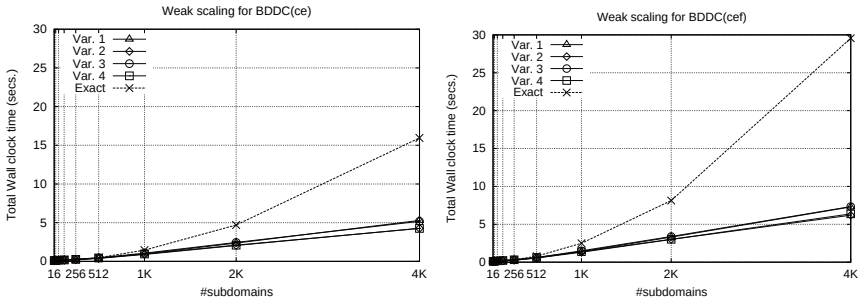


Figure 10: Weak scalability on MN-III for the total computation time of the exact and the four inexact variants (see Table 2) of the BDDC(ce) (left), and BDDC(cef) (right) preconditioners for the 3D Poisson problem on a backward facing step domain (see Figure 9) and a moderate load of 20K FEs/core.

A first (remarkable) observation that can be made from Figure 10 is that time scalability of the exact variant starts degrading at a much more higher pace beyond 512 and 256 subdomains for BDDC(ce) and BDDC(cef), respectively, compared with any of the four inexact variants. This factor also causes the exact variant being slower than the inexact variants beyond that point. Table 8 complements Figure 10 by providing the size (n) and number of nonzeros (nnz) in the coarse-grid matrix (in the columns labeled as "A_c"), and the minimum, maximum, and average local degrees of freedom among all subdomains

(in the columns labeled as "#Local DoFs"). By comparing the number of global coarse DoFs (n), with the number of local DoFs, we can see that beyond the aforementioned number of subdomains, the former already (largely) exceeds the latter. It is therefore not surprising that beyond that point the time spent in coarse-grid computations becomes larger than fine-grid ones (rendering fine/-grid overlapping no longer fully effective), and more and more dominant with the number of subdomains, such that the scalability curve finally reflects the order of complexity of the underlying coarse-grid solver (i.e., $O(n^2)$ versus $O(n)$ for sparse direct solvers compared to AMG preconditioners). The slope of the curve is higher for BDDC(cef) than for BDDC(ce), provided that the former requires additional coarse degrees of freedom (see Table 8).

Solver	#cores	A_c		#Local DoFs			#Local coarse DoFs		
		n	nnz	min.	max.	avg.	min.	max.	avg.
BDDC(ce)	16	39	487	3.04K	3.45K	3.32K	1.0	9.0	6.3
	32	89	1437	3.18K	3.78K	3.52K	1.0	13.0	7.6
	64	308	12734	3.27K	4.08K	3.63K	4.0	36.0	15.5
	128	968	66268	3.22K	4.29K	3.71K	8.0	52.0	24.9
	256	8213	180074	2.97K	4.74K	3.84K	7.0	66.0	28.7
	512	5578	621522	3.07K	4.70K	3.85K	4.0	88.0	36.1
	1024	12579	1612669	2.89K	4.86K	3.91K	6.0	102.0	40.8
	2048	28272	4021436	2.91K	4.93K	4.00K	4.0	98.0	46.0
	4096	61177	9299555	2.93K	4.86K	3.97K	6.0	96.0	49.8
BDDC(cef)	16	66	1236	3.04K	3.45K	3.32K	2.0	13.0	9.7
	32	151	3533	3.18K	3.78K	3.52K	2.0	18.0	11.4
	64	511	27323	3.27K	4.08K	3.63K	6.0	46.0	21.9
	128	1485	123815	3.22K	4.29K	3.71K	12.0	66.0	33.0
	256	3368	326426	2.97K	4.74K	3.84K	11.0	81.0	37.6
	512	8213	1059719	3.07K	4.70K	3.85K	7.0	108.0	46.4
	1024	18229	2676749	2.89K	4.86K	3.91K	10.0	125.0	51.8
	2048	40374	6538298	2.91K	4.93K	4.00K	7.0	118.0	57.8
	4096	86592	14954200	2.93K	4.86K	3.97K	10.0	118.0	62.2

Table 8: Number of coarse-grid DoFs (n) and number of nonzero elements (nnz) in the sparse coarse-grid matrix, minimum, maximum, and average number of local degrees of freedom (among all subdomains), and minimum, maximum, and average number of local coarse degrees of freedom (among all subdomains), for the BDDC(ce) and BDDC(cef) preconditioners.

7. Conclusions and future work

In this work, we have analyzed the scalability of inexact BDDC preconditioners. Inexact AMG solvers are considered, due to their linear complexity and low memory requirements. Further, a highly scalable implementation of fine/coarse duties in time has been used, which is an extension of the work in [12] to inexact solvers. It allows us to fully overlap the coarse problem tasks that harm scalability with embarrassingly parallel fine tasks and reduce both check-pointing and idling. All these choices are motivated by the future exascale scenario, with very large core counts and reduced memory per core.

We have numerically tested the overlapped/inexact implementation of the algorithms in [13] (with a slight modification for the coarse solver approximation) that complements their mathematical analysis. This work shows how far the implementation proposed herein can scale with respect to the number of cores and the size of the global problem when using a serial AMG software package like HSL_MI20 [38].

As inexact solvers, we have considered a fixed (one or two) number of AMG cycles. (The use of PCG-AMG local/coarse solvers was considered, but it turned out to be less efficient than a fixed number of AMG cycles in all cases.) First, we have carried out a sensitivity analysis, to analyze the effect of inexact solvers on iteration counts/condition numbers. Next, we have performed a comprehensive weak scalability analysis up to 93,312 cores and more than 20 billion unknowns on JUQUEEN, at the Jülich Supercomputing Center (JSC). As far as we know, these are the largest scale scalability analyses and simulations performed so far with DD methods. Even using a single core with 1 GByte of memory for the coarse-grid problem, the scalability of the inexact variants represent a dramatic improvement compared to the largest scale scalability analyses of exact BDDC methods so far (see [12]), justifying the approach considered herein.

Out of this analysis, we can conclude that for moderate core counts, less than 35K, the best option is to use one AMG cycle for all the local and coarse problems. However, as we run on larger sets of processors, to compute slightly more accurately the coarse basis and Neumann problems (using two AMG cycles) certainly pays the price; the resulting coarse problem is easier to approximate with AMG, the number of iterations is reduced, and there is more fine work load to fully overlap the coarse tasks. With respect to the three variants of BDDC methods, namely BDDC(c), BDDC(ce), and BDDC(cef), the increased number of iterations of BDDC(c) preconditioner in three dimensions does not compensate its smaller coarse problem in the scales considered herein, since the coarse problem CPU cost (at least in part) is being overlapped with fine duties. In general, the BDDC(ce) preconditioner has the best compromise between number of iterations and computational cost.

Scalable Poisson solvers on unstructured meshes are of great importance when dealing with LES simulations of turbulent flows on general geometries. Pressure segregation techniques are used, leading to a pressure Poisson problem [26, 27]. We have analyzed the effectiveness of the inexact BDDC setting on the typical backward-facing step benchmark for unstructured meshes, both in two and three dimensions.

The next step in our effort to push forward balancing DD scalability till extreme core counts is to distribute the coarse-grid problem among several MPI coarse-grid tasks, possibly spanning multiple compute nodes. It can be accomplished by linking our inexact/overlapped BDDC implementation with a MPI-distributed AMG solver like BoomerAMG [18], or alternatively, to extend the overlapping BDDC techniques described above to a multilevel setting, using a recursive use of our implementation at FEMPAR. Based on our current experience (overlapped two-level implementations can scale up to several tens of thousands of processors) and existing mathematical analyses, we can naturally

expect a three-level overlapped implementation of BDDC to perfectly scale in the largest HPC systems today. This is not explored here, but left as an exciting future line of research.

Acknowledgements

This work has been funded by the European Research Council under the FP7 Program Ideas through the Starting Grant No. 258443 - COMFUS: Computational Methods for Fusion Technology and the FP7 NUMEXAS project under grant agreement 611636. A. F. Martín was also partially funded by the Generalitat de Catalunya under the program "Ajuts per a la incorporació, amb caràcter temporal, de personal investigador júnior a les universitats públiques del sistema universitari català PDJ 2013". We acknowledge PRACE for awarding us access to resource JUQUEEN based in Germany at the Jülich Supercomputing Centre (JSC), and the Gauss Centre for Supercomputing (GCS) for providing computing time through the John von Neumann Institute for Computing (NIC) on the GCS share also on JUQUEEN. GCS is the alliance of the three national supercomputing centres HLRS (Universität Stuttgart), JSC, and LRZ (Bayerische Akademie der Wissenschaften), funded by the German Federal Ministry of Education and Research (BMBF) and the German State Ministries for Research of Baden-Württemberg (MWK), Bayern (StMWFK) and Nordrhein-Westfalen (MIWF). We gratefully acknowledge JSC's staff in general, and Dirk Broemmel in particular, for their support in porting/debugging FEMPAR and its dependencies to/on JUQUEEN. Finally, the authors thankfully acknowledge the computer resources, technical expertise and assistance provided by the Red Española de Supercomputación.

References

[1] Report on the workshop on extreme-scale solvers: Transition to future architectures, Tech. rep., U.S. Department of Energy (2012).

[2] P. T. Lin, J. N. Shadid, M. Sala, R. S. Tuminaro, G. L. Hennigan, R. J. Hoekstra, Performance of a parallel algebraic multilevel preconditioner for stabilized finite element semiconductor device modeling, Journal on Computational Physics 228 (17) (2009) 6250–6267. doi:http://dx.doi.org/10.1016/j.jcp.2009.05.024.

[3] K. Stüben, A review of algebraic multigrid, Journal of Computational and Applied Mathematics 128 (12) (2001) 281 – 309. doi:10.1016/S0377-0427(00)00516-1.
URL http://www.sciencedirect.com/science/article/pii/S0377042700005161

[4] A. Toselli, O. Widlund, Domain Decomposition Methods - Algorithms and Theory, Springer-Verlag, 2005.

[5] J. Mandel, Balancing domain decomposition, Communications in Numerical Methods in Engineering 9 (3) (1993) 233–241.
URL http://dx.doi.org/10.1002/cnm.1640090307

[6] C. R. Dohrmann, A preconditioner for substructuring based on constrained energy minimization, SIAM Journal on Scientific Computing 25 (1) (2003) 246–258. doi:10.1137/S1064827502412887.
URL http://link.aip.org/link/?SCE/25/246/1

[7] C. Farhat, K. Pierson, 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 (2–4) (2000) 333–374. doi:10.1016/S0045-7825(99)00234-0.
URL http://www.sciencedirect.com/science/article/pii/S0045782599002340

[8] S. Badia, A. F. Martín, J. Principe, Enhanced balancing Neumann-Neumann preconditioning in computational fluid and solid mechanics, International Journal for Numerical Methods in Engineering 96 (4) (2013) 203–230.

[9] P. Amestoy, I. Duff, J.-Y. L'Excellent, Multifrontal parallel distributed symmetric and unsymmetric solvers, Computer Methods in Applied Mechanics and Engineering 184 (24) (2000) 501–520. doi:10.1016/S0045-7825(99)00242-X.
URL http://www.sciencedirect.com/science/article/pii/S004578259900242X

[10] B. Sousedík, J. Šístek, J. Mandel, Adaptive-multilevel BDDC and its parallel implementation, Computing 95 (12) (2013) 1087–1119. doi:10.1007/s00607-013-0293-5.
URL http://link.springer.com/article/10.1007/s00607-013-0293-5

[11] V. Hapla, D. Horak, M. Merta, Use of direct solvers in TFETI massively parallel implementation, in: Applied Parallel and Scientific Computing, no. 7782 in Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2013, pp. 192–205.
URL http://link.springer.com/chapter/10.1007/978-3-642-36803-5_14

[12] S. Badia, A. F. Martín, J. Principe, A highly scalable parallel implementation of balancing domain decomposition by constraints, SIAM Journal on Scientific Computing (2014) C190–C218doi:10.1137/130931989.
URL http://epubs.siam.org/doi/abs/10.1137/130931989

[13] C. R. Dohrmann, An approximate BDDC preconditioner, Numerical Linear Algebra with Applications 14 (2) (2007) 149168. doi:10.1002/nla.514.

URL http://onlinelibrary.wiley.com/doi/10.1002/nla.514/abstract

[14] X. Tu, Three-level BDDC in three dimensions, SIAM Journal on Scientific Computing 29 (4) (2007) 1759–1780. doi:10.1137/050629902.
URL http://epubs.siam.org/doi/abs/10.1137/050629902

[15] J. Mandel, B. Sousedík, C. Dohrmann, Multispace and multilevel BDDC, Computing 83 (2) (2008) 55–85. doi:10.1007/s00607-008-0014-7.
URL http://www.springerlink.com/content/112v4w1821r584u0/abstract/

[16] J. Mandel, C. R. Dohrmann, R. Tezaur, An algebraic theory for primal and dual substructuring methods by constraints, Applied Numerical Mathematics 54 (2) (2005) 167–193.

[17] J. Li, O. B. Widlund, On the use of inexact subdomain solvers for BDDC algorithms, Computer Methods in Applied Mechanics and Engineering 196 (8) (2007) 1415–1428. doi:10.1016/j.cma.2006.03.011.
URL http://www.sciencedirect.com/science/article/pii/S0045782506002611

[18] V. E. Henson, U. M. Yang, BoomerAMG: a parallel algebraic multigrid solver and preconditioner, Applied Numerical Mathematics 41 (1) (2002) 155–177. doi:10.1016/S0168-9274(01)00115-5.
URL http://www.sciencedirect.com/science/article/pii/S0168927401001155

[19] O. Rheinbach, Parallel iterative substructuring in structural mechanics, Archives of Computational Methods in Engineering 16 (4) (2009) 425–463. doi:10.1007/s11831-009-9035-4.
URL http://link.springer.com/article/10.1007/s11831-009-9035-4

[20] A. Klawonn, O. Rheinbach, Highly scalable parallel domain decomposition methods with an application to biomechanics, ZAMM - Journal of Applied Mathematics and Mechanics 90 (1) (2010) 532. doi:10.1002/zamm.200900329.
URL http://onlinelibrary.wiley.com/doi/10.1002/zamm.200900329/abstract

[21] O. Schenk, K. Gärtner, On fast factorization pivoting methods for sparse symmetric indefinite systems, Electronic Transactions on Numerical Analysis 23 (2006) 158–179.

[22] O. Colomés, S. Badia, R. Codina, J. Principe, Assessment of variational multiscale models for the large eddy simulation of turbulent incompressible flows, Computer Methods in Applied Mechanics and Engineering 285 (2015) 32–63.

[23] O. Colomés, S. Badia, Segregated runge-kutta methods for the incompressible navier-stokes equations, In press.

[24] S. Badia, A. F. Martn, R. Planas, Block recursive LU preconditioners for the thermally coupled incompressible inductionless MHD problem, Journal of Computational Physics 274 (2014) 562–591. doi:10.1016/j.jcp.2014.06.028.
URL http://www.sciencedirect.com/science/article/pii/S0021999114004355

[25] S. Badia, R. Planas, J. V. Gutirrez-Santacreu, Unconditionally stable operator splitting algorithms for the incompressible magnetohydrodynamics system discretized by a stabilized finite element formulation based on projections, International Journal for Numerical Methods in Engineering 93 (3) (2013) 302–328. doi:10.1002/nme.4392.
URL http://onlinelibrary.wiley.com/doi/10.1002/nme.4392/abstract

[26] H. C. Elman, D. J. Silvester, A. J. Wathen, Finite elements and fast iterative solvers: with applications in incompressible fluid dynamics, Oxford University Press, 2005.

[27] S. Badia, R. Codina, Algebraic Pressure Segregation Methods for the Incompressible Navier-Stokes Equations, Archives of Computational Methods in Engineering 15 (3) (2008) 343–369.

[28] P. Jolivet, V. Dolean, F. Hecht, F. Nataf, C. Prud'homme, N. Spillane, High performance domain decomposition methods on massively parallel architectures with FreeFEM++, J. Numer. Math. 20 (3-4) (2012) 287–302.

[29] S. Badia, A. F. Martín, J. Principe, Implementation and scalability analysis of balancing domain decomposition methods, Archives of Computational Methods in Engineering 20 (3) (2013) 239–262. doi:10.1007/s11831-013-9086-4.

[30] J. Šístek, M. Čertíková, P. Burda, J. Novotný, Face-based selection of corners in 3D substructuring, Mathematics and Computers in Simulation 82 (10) (2012) 1799–1811. doi:10.1016/j.matcom.2011.06.007.
URL http://www.sciencedirect.com/science/article/pii/S0378475411001820

[31] J. Mandel, C. R. Dohrmann, Convergence of a balancing domain decomposition by constraints and energy minimization, Numerical Linear Algebra with Applications 10 (7) (2003) 639–659. doi:10.1002/nla.341.
URL http://dx.doi.org/10.1002/nla.341

[32] G. Golub, Q. Ye, Inexact preconditioned conjugate gradient method with inner-outer iteration, SIAM Journal on Scientific Computing 21 (4) (1999) 1305–1320. doi:10.1137/S1064827597323415.
URL http://epubs.siam.org/doi/abs/10.1137/S1064827597323415

[33] Y. Saad, A flexible inner-outer preconditioned GMRES algorithm, SIAM Journal on Scientific Computing 14 (12) (1993) 461—469.

[34] P. Vaněk, J. Mandel, M. Brezina, Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems, Computing 56 (1996) 179–196.
URL http://dx.doi.org/10.1007/BF02238511

[35] M. Sala, R. Tuminaro, A new Petrov-Galerkin smoothed aggregation preconditioner for nonsymmetric linear systems, SIAM Journal on Scientific Computing 31 (1) (2008) 143–166. doi:10.1137/060659545.

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

[37] F. Hecht, FreeFem++ User's manual. 3rd edition, Version 3.22, available at http://www.freefem.org/ff++/ftp/freefem++doc.pdf (2013).
URL http://www.freefem.org/ff++/ftp/freefem++doc.pdf

[38] J. Boyle, M. Mihajlović, J. Scott, HSL_MI20: An efficient AMG preconditioner for finite element problems in 3D, International Journal for Numerical Methods in Engineering 82 (1) (2010) 64–98. doi:10.1002/nme.2758.
URL http://onlinelibrary.wiley.com/doi/10.1002/nme.2758/abstract

[39] J. W. Ruge, K. Stüben, Algebraic multigrid (AMG), in: Multigrid Methods, S. F. McCormick Edition, Vol. 3 of Frontiers in Applied Mathematics, SIAM, Philadelphia, PA, 1987, pp. 73–130.

[40] U. Trottenberg, C. C. W. Oosterlee, A. Schüller, MULTIGRID, Academic Press, 2001.

[41] J. Hogg, J. Reid, J. Scott, Design of a multicore sparse cholesky factorization using DAGs, SIAM Journal on Scientific Computing 32 (6) (2010) 3627–3649. doi:10.1137/090757216.
URL http://epubs.siam.org/doi/abs/10.1137/090757216

[42] G. Karypis, A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. Version 5.1.0, Tech. rep., University of Minnesota, Department of Computer Science and Engineering, Minneapolis, MN, available at http://glaros.dtc.umn.edu/gkhome/fetch/sw/metis/manual.pdf (2013).
URL http://glaros.dtc.umn.edu/gkhome/fetch/sw/metis/manual.pdf