M. Griebel, A. Schweitzer, L. Troska

A Fault-Tolerant Domain Decomposition Method Based On Space-Filling Curves

A FAULT-TOLERANT DOMAIN DECOMPOSITION METHOD BASED ON SPACE-FILLING CURVES

MICHAEL GRIEBEL, MARC-ALEXANDER SCHWEITZER, AND LUKAS TROSKA

ABSTRACT. We propose a simple domain decomposition method for d-dimensional elliptic PDEs which involves an overlapping decomposition into local subdomain problems and a global coarse problem. It relies on a space-filling curve to create equally sized subproblems and to determine a certain overlap based on the one-dimensional ordering of the space-filling curve. Furthermore we employ agglomeration and a purely algebraic Galerkin discretization in the construction of the coarse problem. This way, the use of d-dimensional geometric information is avoided. The subproblems are dealt with in an additive, parallel way, which gives rise to a subspace correction type linear iteration and a preconditioner for the conjugate gradient method. To make the algorithm fault-tolerant we store on each processor, besides the data of the associated subproblem, a copy of the coarse problem and also the data of a fixed amount of neighboring subproblems with respect to the one-dimensional ordering of the subproblems induced by the space-filling curve. This redundancy then allows to restore the necessary data if processors fail during the computation. Theory from [GO20] supports that the convergence rate of such a linear iteration method stays the same in expectation, and only its order constant deteriorates slightly due to the faults. We observe this in numerical experiments for the preconditioned conjugate gradient method in slightly weaker form as well. Altogether, we obtain a fault-tolerant, parallel and efficient domain decomposition method based on space-filling curves which is especially suited for higher-dimensional elliptic problems.

1. INTRODUCTION

Higher-dimensional problems beyond four dimensions appear in many mathematical models in medicine, finance, engineering, biology and physics. Their numerical treatment poses a challenge for modern high performance compute systems. With the advent of petascale compute systems in recent years and exascale computers to arrive in the near future, there is tremendous parallel compute power available for parallel simulations. But while higher-dimensional applications involve a huge number of degrees of freedom, their efficient parallelization by e.g. conventional domain decomposition approaches is difficult. Load balancing and scalability might be dimension-dependent, especially for geometry-based domain decomposition approaches, and necessary communication might grow strongly with increasing dimensionality. Thus one aim is to find parallel algorithms with good load balancing, good scaling properties and moderate communication costs especially for higher-dimensional problems. Moreover systems with hundreds of thousand or even millions of processor units

---

will be increasingly prone to failures, which can corrupt the results of parallel solvers or renders them obsolete at all. It is predicted that large parallel applications may suffer from faults as frequently as once every 30 minutes on future exascale platforms [SW14]. Thus a second aim is to derive not just balanced, scalable and fast parallel algorithms, but to make them fault-tolerant as well. Besides hard errors, for which hardware mitigation techniques are under development, there is the issue of soft errors [KRS13, SW14, Tre05]. For further details and literature on resilience and fault tolerance, see [A20]. Altogether, the development of fault-tolerant and numerically efficient parallel algorithms are of utmost importance for the simulation of large-scale problems.

In this article we focus on algorithm-based fault tolerance. Indeed, a fault-tolerant, parallel, iterative domain decomposition method can be interpreted as an instance of the *stochastic* subspace correction approach. For such methods there exists a general theoretical foundation for convergence, which was developed in a series of papers [GO12, GO16, GO18]. Moreover, for a conventional geometry-based domain decomposition approach, this theory was already employed in [GO20] to show algorithm-based fault tolerance theoretically and, for the two-dimensional case, also practically under independence assumptions on the random failure of subproblem solves. We now propose a simple domain decomposition method for d-dimensional elliptic PDEs which works for higher dimensions. Besides an overlapping decomposition into local subdomain problems, it also involves a global coarse problem. To create nearly equally sized subproblems, we rely on *space-filling curves*. This way, the overall number N of degrees of freedom is partitioned for P processors into N/P-sized subproblems regardless of the dimension of the PDE and the number of available processors. This is in contrast to many geometry-based domain decomposition methods, where – e.g. in the uniform grid case with mesh size h and $N \approx h^{-d}$ depending exponentially on d – the number of processors is usually to be chosen as a power of d. Furthermore, in our method, the overlap is determined based on the one-dimensional ordering of the space-filling curve as well. Moreover we employ agglomeration and a purely algebraic Galerkin discretization to again avoid d-dimensional geometric information in the construction of the coarse space problem. The subproblems and the coarse space problem are dealt with in an additive, parallel way, which leads to an additive Schwarz subspace correction method that resembles a block-Richardson-type linear iteration. To speed up convergence, we also employ this approach in a preconditioner for the conjugate gradient (CG) method for the fine grid discretization of the PDE. To this end, we store the global coarse space problem redundantly on each processor and also solve it redundantly on each processor in parallel. Moreover, to gain fault tolerance, we store on each processor not just the data of the associated subproblem and the coarse problem but also the data of a fixed amount of neighboring subproblems with respect to the *one-dimensional* ordering of the subproblems, which is induced by the space-filling curve. This results in sufficient redundancy of the stored data, whereas the amount of stored data is just enlarged by a constant. If a processor now fails in the course of the computation, a new replacement processor is invoked from a reserve batch instead of the faulty one, the corresponding necessary data is transferred to it from (one

of) the neighboring processors, and the iterative method proceeds with its computation on this processor as well. Altogether, we obtain an algorithm-based fault-tolerant, parallel, iterative algorithm, which can be interpreted as an instance of the *stochastic* subspace correction approach. Again, the theory in [GO20] supports that the convergence rate of the overall linear Richardson-type iteration stays in expectation the same, and only the order constant deteriorates slightly due to the faults, provided that the number of faults stays bounded and their occurrence among the processors is sufficiently well distributed. For the preconditioned conjugate gradient approach we do not have such a theory, but a similar, though slightly weaker behavior can nevertheless be observed in practice. Altogether, we obtain a fault-tolerant, well-balanced, parallel domain decomposition method, which is based on space-filling curves and which is thus especially suited for higher-dimensional elliptic problems.

The remainder of this paper is organized as follows: In section 2 we discuss our domain decomposition method, which is based on a space-filling curve. We first give a short overview on domain decomposition methods and their properties for elliptic PDEs. Then we discuss space-filling curves and their peculiarities. Finally, we present our algorithm and its features. In section 3 we deal with algorithmic fault tolerance. Here we recall its close relation to randomized subspace correction for our setting. Then we present a fault-tolerant variant of our domain decomposition method. In section 4 we discuss the results of our numerical experiments. We first define the model problem which we employ. Then we give convergence and parallelization results. Furthermore we show the behavior of our method under failure of processors. Finally we give some concluding remarks in section 5.

2. A domain decomposition method based on space-filling curves

2.1. Domain decomposition. The domain decomposition approach is a simple method for the solution of discretized partial differential equations and is typically used as a preconditioner for the conjugate gradient method or other Krylov iterative methods. Its idea can be traced back to Schwarz [S70]. Depending on the specific choice of the subdomains, one can distinguish between overlapping and non-overlapping domain decomposition methods, where the subdomains geometrically overlap either to a certain extent or intersect only at their common interfaces. The latter are often also called iterative substructuring methods in the engineering community. It turned out that such simple domain decomposition methods can not possess fast convergence rates and thus, starting in the mid 80s, various techniques had been developed to introduce an additional coarse scale problem, which provides a certain amount of global transfer of information across the whole domain and thus substantially speeds up the iteration. For the overlapping case it could be shown in [DW87] that the condition number of the fine grid system preconditioned by such a two-level additive Schwarz method is of the order

$$(2.1) \qquad\qquad O(1 + H/\delta),$$

where δ denotes the size of the overlap and H denotes the coarse mesh size. This bound also holds for small overlap [DW94] and can not be improved further [B00]. Thus, if the quotient of the coarse mesh size H and the overlap δ stays constant, the method is indeed optimally preconditioned and weakly scalable. For further details on domain decomposition methods see e.g. the books [SBWG96, QV99, TW04, DJN15].

We obtain a two-level additive Schwarz method as follows: Consider an elliptic differential equation

$$\mathcal{L}u = f \tag{2.2}$$

in the domain $\Omega \subset \mathbb{R}^d$, e.g. the simple Poisson problem on a d-dimensional cube. Using a conforming finite element, a direct finite difference or a finite volume discretization involving N degrees of freedom and mesh size $h \approx N^{-1/d}$, we arrive at the system of linear equations

$$Ax = b \tag{2.3}$$

with sparse stiffness matrix $A \in \mathbb{R}^{N \times N}$, right hand side vector $b \in \mathbb{R}^N$ and unknown coefficient vector $x \in \mathbb{R}^N$, which needs to be solved. Suppose that

$$\Omega = \bigcup_{i=1}^{P} \Omega_i$$

is covered by a finite number P of well-shaped subdomains Ω_i of diameter $\approx H$ which might locally overlap. It is silently assumed that $h << H$ and that the subdomains are aligned with the fine mesh. Now denote by N_i the number of grid points associated to each Ω_i, i.e. the degrees of freedom associated to the subdomains $\Omega_i, i = 1, \ldots, P$. Then denote by $R_i : \mathbb{R}^N \to \mathbb{R}^{N_i}$ the restriction operators, which map the entries of the coefficient vector x corresponding to the full grid on Ω to the coefficient vectors x_i corresponding to the local grids on the subdomains Ω_i. Analogously denote by $R_i^T : \mathbb{R}^{N_i} \to \mathbb{R}^N$ the extension operators, which map the coefficient vectors from the local grid on the subdomains Ω_i to that of the full grid on Ω via the natural extension by zero. Then the local stiffness matrices associated to the subdomains Ω_i can be denoted as $A_i \in \mathbb{R}^{N_i \times N_i}$ with $A_i := R_i A R_i^T$. Finally, we add a coarse space problem with dimension N_0 as a second level via the restriction operator $R_0 : \mathbb{R}^N \to \mathbb{R}^{N_0}$, which maps from the full grid on Ω to the respective global coarse space. The associated coarse stiffness matrix then can be generated via the Galerkin approach as $A_0 := R_0 A R_0^T$. Altogether, with the one-level additive Schwarz operator

$$C_{(1)}^{-1} := \sum_{i=1}^{P} R_i^T A_i^{-1} R_i, \tag{2.4}$$

this gives the two-level additive Schwarz operator

$$C_{(2)}^{-1} := R_0^T A_0^{-1} R_0 + C_{(1)}^{-1} = \sum_{i=0}^{P} R_i^T A_i^{-1} R_i, \tag{2.5}$$

which can be used, with a properly chosen relaxation parameter, directly for a linear iterative method or as preconditioner within a steepest descent or conjugate gradient solver for (2.3). A notational variant based on space splittings is given in [GO20]. Note that there are more sophisticated space splittings which follow the Bank-Holst technique [BH03], where the coarse problem is formally avoided by including a redundant copy of it into each of the subdomain problems with $i = 1, \ldots, P$. We will indeed follow this approach later on.

Now, if the condition number $\kappa(C_{(2)}^{-1}A) = \lambda_{\max}(C_{(2)}^{-1}A)/\lambda_{\min}(C_{(2)}^{-1}A)$ of the preconditioned system is independent of the number P of subproblems for fixed N, we obtain *strong scalability*. If it is independent of the quotient N/P, i.e. the problem size per subdomain and thus per processor stays fixed, we obtain *weak scalability*. Moreover, if it is independent of the number N of degrees of freedom, we would have an optimally preconditioned method, which however still may depend on P and might thus not be scalable. Note furthermore that we employ here for reasons of simplicity a direct solver for A_i^{-1} on all subproblems and for A_0^{-1} on the coarse scale, which involves Gaussian elimination and comes with a certain cost. However, the corresponding matrix factorization needs to be performed just once at the beginning and, in the plain linear iteration or in the preconditioned conjugate gradient iteration, only the cheaper backward and forward steps need to be employed. Alternatively, approximate iterative methods might be used as well, like the multigrid or BPX-multilevel method, which would even results in optimal linear cost for the subproblem solves. This given, to achieve a mesh-independent condition number for the preconditioned system $C_{(2)}^{-1}A$ with $C_{(2)}^{-1}$ as in (2.5), one usually chooses for the coarse problem a suitable FE space on the mesh of domain partitions, where a linear FE space will do for a second-order elliptic problem such as (2.2). Mild shape regularity assumptions on the overlapping subdomains Ω_i then guarantee robust condition number estimates of the form $\kappa \leq c(1+\frac{H}{\delta})$, see [TW04, Theorem 3.13]. Dropping the coarse grid problem, i.e. considering a one-level preconditioner as in (2.4) without the coarse problem $R_0^T A_0^{-1} R_0$, would lead to the worse bound $\kappa \leq cH^{-2}(1+H/\delta)$. Note that, even though these estimates imply a deterioration of the condition number proportional to δ^{-1} if $\delta \to h$, in practice good performance has been observed when only a few layers of fine mesh cells form the overlap at the surface of the Ω_i. With the use of an additional coarse grid problem based on piecewise linears, an optimal order of the convergence rate is then guaranteed for elliptic problems. The additional coarse grid problem results in a certain communication bottleneck, which however can by principle not be avoided and is inherent in all elliptic problems. Similar issues arise for multigrid and multilevel algorithms as well, but these methods are more complicated to parallelize in an efficient way on large compute systems. Moreover their achieved convergence rate and cost complexity is not better than for the domain decomposition approach with coarse grid, at least in order, albeit the order constant might be.

In the above two-level method, the coarse problem is mostly determined as a kind of *geometric* coarse grid or mesh, which corresponds to the specifically chosen set of subdomains and also involves a geometric coarse mesh size. Moreover, on this geometric coarse grid,

usually a set of piecewise d-linear basis functions is employed, which directly gives a geometric d-dimensional prolongation to the fine grid space by e.g. interpolation. Furthermore a conventional discretization by finite elements (FE) is invoked on this coarse grid either by direct discretization or by Galerkin coarsening. This is the basis for both the proofs of the theoretical convergence estimates and the practical implementation of the algorithms. Besides, the wire basket type constructions [BPS89, S90] and related techniques are used in theory and practice with good success. They result in somewhat more general coarse spaces, mimicking discrete harmonics to some extent, but are also based on geometric d-dimensional information of the fine mesh, the respective domain partitions and their boundaries. Altogether, such geometric coarse spaces work well for problems with up to three dimensions. Their extension to higher dimension, however, is neither straightforward nor simple and involves local costs which in general scale exponentially in d. More information on the development of coarse spaces for domain decomposition can be found in [W09].

Besides a mesh-based geometric coarse grid problem, we can derive a coarse problem in a purely *algebraic* way. To this end, let V_H be a coarse space with $N_0 := \dim(V_H)$ and let Z be a basis for it, i.e. $V_H = \operatorname{span} Z$. The restriction $R_0 : \mathbb{R}^N \to \mathbb{R}^{N_0}$ from the fine grid space V_h to the coarse space V_H can be (algebraically) defined as the matrix $R_0 := Z^T \in \mathbb{R}^{N_0 \times N}$ and the coarse space discretization is again given via the Galerkin approach as $A_0 := R_0 A R_0^T$.

There are now different specific choices for Z or R_0^T, respectively. In [N87] it was suggested to employ the kernel of the underlying differential operator as coarse space, i.e. the constant space for the Laplace operator. Thus, with $N_0 = P$,

$$R_0^T := (R_i^T D_i R_i \mathbf{1})_{1 \le i \le P}$$

with $\mathbf{1} = (1, \dots, 1)^T$ and with diagonal matrices D_i such that a partition of unity results, i.e.

$$\sum_{i=1}^{P} R_i^T D_i R_i = I.$$

Indeed, it is observed that the associated two-level preconditioner gives good results [M88, M90] and weak scaling is achieved in practice. Note that this approach can be easily generalized to the case $N_0 = qP$ such that q degrees of freedom are associated on the coarse scale to each subdomain instead of just one. Improved variants, namely the balanced method [M93] and the deflation method [VN06], had been developed subsequently. With the definitions

$$(2.6) \qquad F := R_0^T A_0^{-1} R_0 \quad \text{and} \quad G := I - AF$$

and the plain one-level additive Schwarz operator $C_{(1)}^{-1}$ from (2.4), we get the additively corrected operator $C_{(1)}^{-1} + F$ due to Nicolaides [N87], the deflated approach $G^T C_{(1)}^{-1} + F$ ([VN06]) and the balanced version [M93]

$$(2.7) \qquad C_{(2),bal}^{-1} := G^T C_{(1)}^{-1} G + F.$$

Closely related are agglomeration techniques inspired by the algebraic multigrid method, volume agglomeration methods stemming from multigrid in the context of finite volume discretizations and so-called spectral coarse space constructions, see e.g. [AAD11] and [DJN15].

2.2. **Space-filling curves.** A main question for domain decomposition methods is how to construct the partition $\{\Omega_i\}_{i=1}^P$ in the first place. To this end, for a fixed number P, one aim is surely to derive a set of subdomains which involves equal size and thus equal computational load for each of the subdomains. If we just consider uniform fine grid discretizations, a simple uniform and geometric decomposition of the mesh would need P to be a power of d, i.e. $P = \bar{P}^d$ with $\bar{P}$ being the amount of subdomain splitting in each coordinate direction. This however prohibits the use of a slowly growing number of processors and merely allows numbers of processors which are to be chosen as a power of d. Similar issues arise for anisotropic mesh sizes $h = (h_1, \ldots, h_d)$ with $h_i \neq h_j$, more general quasi uniform finite element meshes (for which often mesh partitioners like METIS or similar methods are employed) or adaptive meshes, where a well-balanced domain decomposition can be complicated to derive. In this article, we follow the lead of [GZ00, GZ01] and rely on *space-filling curves* to create nearly equally sized subproblems. This way, the overall number N of degrees of freedom is partitioned for P processors into N/P-sized subproblems (up to round-off deviations, i.e. up to a value of just one) regardless of the dimension of the PDE and the number of available processors. For further details see also [GKZ07, B13, Z03].

In general, we consider the situation where a domain X in a one-dimensional space $X \subset \mathbb{R}$ is mapped to a domain Y in a d-dimensional space $Y \subset \mathbb{R}^d$, $d \geq 1$, i.e. we consider the mapping

$$s : X \to Y, x \to y := s(x).$$

The question is: Is there a one-dimensional curve that runs through all points of the d-dimensional domain, i.e. that is indeed filling the whole domain Y? In 1878 Cantor [C78] discovered that there is a *bijective* function between any two finite-dimensional smooth manifolds independent of their dimension. This is a remarkable and surprising result: For example, it is possible to map a one-dimensional curve 1:1 onto a two-dimensional surface and, consequently, the surface contains the same amount of points as the curve. Especially the points of the unit interval $[0, 1]$ can be mapped onto the unit square $[0, 1]^2$ and vice versa. Thus such a mapping from the interval to the square is indeed filling the whole square. The second question then is: Is there a *continuous* mapping between any two finite-dimensional smooth manifolds X and Y independent of their dimension? In 1879 Netto [N79] settled this issue by proving that, if the dimensions of the two manifolds are different, such a function is necessarily discontinuous. But what happens if one sacrifices bijectivity? Does a *surjective*, continuous map from the one-dimensional interval to the higher-dimensional manifold exist? This question was positively answered in 1890 by Peano [P90]. He was the first to present such a map, which is now known as Peano's space-filling curve. It maps the unit interval into the plane where the image has positive Jordan content.

One year later, Hilbert [H91] gave another map with such properties. While Peano's curve was defined purely analytically in the first place, Hilbert's approach was given by means of a geometric construction, see below. In subsequent years, many other space-filling curves were discovered, for example the curves by Lebesgue, Moore, Sierpiński, Polya and Schoenberg. For a detailed overview on space-filling curves we refer to the book [S94].

The construction of the various space-filling curves in general follows a specific recursive process. They differ in the dimension d of the space to be filled, the shape of the space to be filled, i.e. the initial object (for example, a square or a triangle in the case $d = 2$), the refinement rule for the decomposition of the respective objects (for example, subdivision by a factor of $1/2$ or $1/3$ in each direction), and the rule how the resulting sub-objects are to be connected or ordered. The recursive construction process can be explained most easily for the two-dimensional case and for the refinement into four subsquares, i.e. for subdivision by a factor of $1/2$ in each coordinate direction. Indeed, this is the refinement rule which is used for the Hilbert curve, the Moore curve and the Lebesgue curve. Then the associated space-filling curve $s : [0, 1] \to [0, 1]^2$ is given as the limit of a sequence of curves $s_n : [0, 1] \to [0, 1]^2, n = 1, 2, 3, \ldots$. Every curve s_n connects the centers of the 4^n squares that are created by successive subdivision of the unit square by line segments in a specific way. The curve s_{n+1} results from the curve s_n as follows: Each square is subdivided, the centers of the newly created four smaller squares are connected in a given order, and all the 4^n groups of 4^{n+1} centers of smaller squares are connected in the order given by the curve s_n. In this sense, the curve s_{n+1} refines the curve s_n.

The various curves differ in the ordering of the centers in each refinement step. In the case of the Lebesgue curve, the same ordering is used everywhere, as shown in Figure 1. For the Hilbert curve, the ordering is chosen in such a way that, whenever two successive centers are connected by a straight line, only the common edge of the two squares is crossed. The construction is made clearer in Figure 2. There, from each iteration to the next, all existing subsquares are subdivided into four smaller subsquares. These four subsquares are then connected by a pattern that is obtained by rotation and/or reflection of the fundamental pattern given in Figure 2 (left). As it is known where the current discrete curve will enter and exit the existing subsquare, one can determine how to orientate the local pattern and can ensure that each square's first subsquare touches the previous square's last subsquare. One can show that the sequence s_n for Hilbert's curve converges uniformly to a curve s, which implies that this limit curve is continuous. For the Lebesgue curve, the sequence only converges pointwise and the limit is discontinuous.

An important aspect of any space-filling curve is its locality property, i.e. points that are close to each other in $[0, 1]$ should tend to be close after mapping into the higher dimensional space. But how close or apart will they be? To this end, let us consider the Hilbert curve. In [Z03] it was shown that the d-dimensional Hilbert curve $s(x)$ is Hölder continuous, but nowhere differentiable and that we have, for any $x, y \in [0, 1]$, the Hölder condition

$$|s(x) - s(y)|_2 \leq c_s |x - y|^{1/d} \qquad \text{with } c_s = 2\sqrt{d + 3}.$$
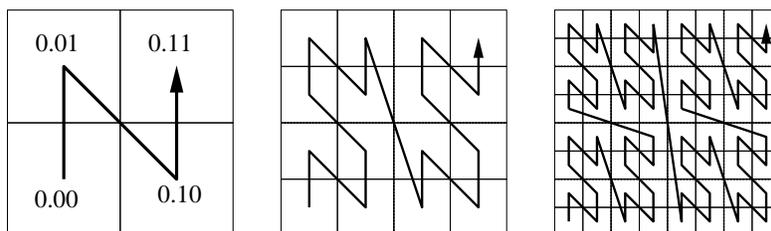
FIGURE 1. Three steps in the construction of the Lebesgue curve.
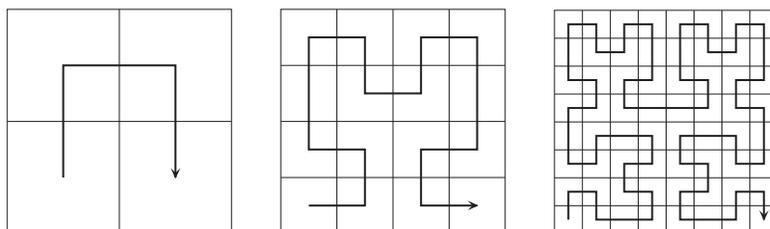
FIGURE 2. Three steps in the construction of the Hilbert curve.

The exponent $1/d$ of the Hölder condition is optimal for space-filling curves and cannot be improved. It also holds for most of the other curves (Peano, Sierpiński, Morton,...), albeit with associated curve- and dimension-dependent constants. It does not hold for the Lebesgue curve, since this curve linearly interpolates outside its defining Cantor set. The Lebesgue curve is thus non-injective, almost everywhere differentiable and its Hölder exponent is merely $1/(d \log_2 3)$.

The two-dimensional recursive constructions can be generalized to higher space dimensions d, i.e. to curves $s : [0, 1] \rightarrow [0, 1]^d$. There exist codes for certain space-filling curves, and especially for the Hilbert curve. A recursive pseudo-code for d dimensions is given in [Z03]. The approach in [B71] was further developed in [T92, M00, L00] and [S04]. Meanwhile, various code versions can be found on github.com [C20, Sh20, P20]. Our implementation of the d-dimensional Hilbert curve is based on [S04].

Space-filling curves were originally developed and studied for purely mathematical reasons. But since the 1980s they were successfully applied in numerics and computer science. This is due to the fact that in one dimension we have a *total ordering* for points whereas in higher dimensions there is only a partial ordering. Space-filling curves in principle allow to retract points from high dimensional space to one-dimensional space and to use the total order there. Moreover the implicit reduction of the dimension from d to one is helpful in many situations. Since space-filling curves preserve locality and neighbor relations to some extent, partitions that contain neighboring points in one-dimensional space will (more or less) correspond to a local point set in the image space as well and it can be expected that a parallelization based on space-filling curves is able to satisfy the two main requirements for partitions, namely uniform load distribution and compact partitions. Thus the space-filling approach is nowadays frequently used for partitioning and for static and adaptive mesh

refinement, for the parallelization and dynamic load balancing of tree-based multilevel PDE-solvers and of tree-based fast integral transforms with applications in, e.g., molecular dynamics and computational astrophysics, compare [GKZ07]. There, the underlying partitioning problem is NP-hard and can only be efficiently dealt with in an approximate heuristic manner. To this end, since space-filling curves provide a sequential order on a multidimensional computational domain and on the corresponding grid cells or particles used in simulations, the partitioning of the resulting list into parts of equal size is an easy task. A further advantage of the space-filling curve heuristic in this context is its simplicity: Parallel dynamic partitioning basically boils down to parallel bucket or radix sort, for further details see e.g. [GKZ07, Z03] and the references cited therein. Meanwhile, space-filling curves are also successfully applied in other areas of scientific computing where, besides parallelization, cache-efficient implementation plays an increasingly important role. There exist now cache-aware and cache-oblivious methods for matrix-matrix multiplication, for the handling of sparse matrices or for the stack&stream-based traversal of space-trees using space-filling curves. For further details see, e.g., [B13] and the references given there.

2.3. **Our algorithm.** Now we discuss the main features of our domain decomposition method for d-dimensional elliptic problems $\mathcal{L}u = f$. For reasons of simplicity, we consider here the unit domain $\Omega = [0,1]^d$ and employ Dirichlet boundary conditions on $\delta\Omega$. The discretization is done with a uniform but in general anisotropic mesh size $h = (h_1, \ldots, h_d)$, where $h_j = 2^{-l_j}$ with multivariate level parameter $l = (l_1, \ldots, l_d)$, which gives the global mesh Ω_h. The number of interior grid points, and thus the number of degrees of freedom is then

$$(2.8) \qquad N := \prod_{j=1}^{d} (2^{l_j} - 1).$$

The reason for considering the general anisotropic situation is the following: In a forthcoming paper, we will employ our specific domain decomposition method as a parallel and fault-tolerant inner solver within the combination method [GSZ92, GH14] for the sparse grid discretization [Z91, BG04, G12] of high-dimensional elliptic PDEs. There, problems with in general anisotropic discretizations with the level parameters

$$l = (l_1, \ldots, l_d) \in \mathbb{N}^d, \text{ where } |l|_1 := l_1 + \ldots + l_d = L + (d-1) - i, i = 0, \ldots, d-1, , l_j > 0.$$

are to be solved. The resulting solutions $u_l(x)$, $x = (x_1, \ldots, x_d)$, are then to be linearly combined as

$$(2.9) \qquad u(x) \approx u_L^{(c)}(x) := \sum_{i=0}^{d-1} (-1)^i \binom{d-1}{i} \sum_{|l|_1 = L + (d-1) - i} u_l(x).$$

Figure 3 illustrates the construction of the combination method in the two-dimensional case with $L = 3$. Then five anisotropic coarse full grids Ω_l are generated on which we solve our problem. Once the problem is solved on each of the grids, we combine these solutions with the appropriate weights (here either +1 or -1). This combination yields a

sparse grid approximation (on the grid $\Omega_L^{(c)}$), which is our approximation to the problem on the (unfeasibly large) full grid Ω_L.
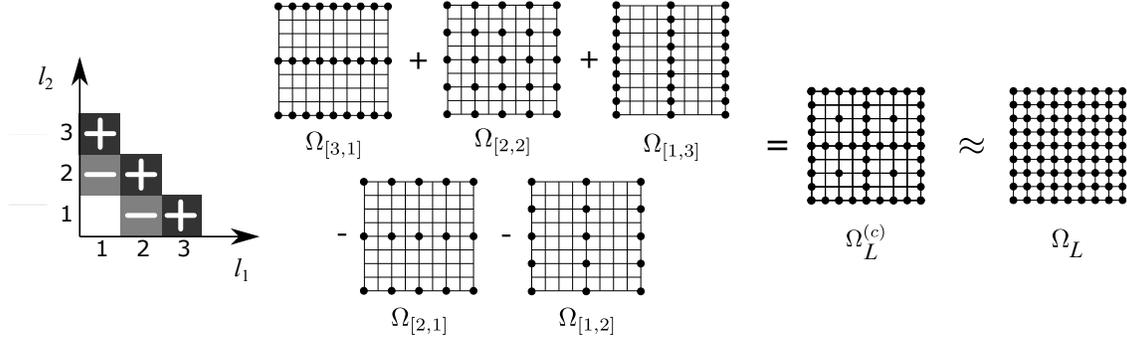
FIGURE 3. The combination method, two-dimensional case with $L = 3$.

This results in a sparse grid approximation to the high-dimensional original problem thus breaking the curse of dimension of a conventional full grid discretization, provided that a certain bounded mixed regularity is present. For details see [BG04] and the references cited therein. We were able to show that the direct hierarchical sparse grid approach and the combination method indeed possess the same order of convergence, see also [GSZ92, BGRZ94, GH14]. The combination method allows to reuse existing codes to a large extent. Moreover the various discrete subproblems can be treated independently of each other [Gri92], which introduces a second level of parallelization beyond the parallelism within each subproblem solver due to the domain decomposition. Furthermore, by means of a fault-tolerant domain decomposition method, also a fault-tolerant parallel combination method for sparse grid problems is automatically induced. Thus it is necessary to be able to deal with the various anisotropic problems arising in the combination method in a simple and efficient manner with *one* single domain decomposition code, which runs straightforwardly and automatically for all these different problems and will not need tedious modifications and adaptions by hand.

Now, for the discretization of $\mathcal{L}u = f$ on the grid Ω_h, we employ the simple finite difference method (or the usual finite element method with piecewise d-linear basis functions) on Ω_h which results in the system of linear equations $Ax = b$ with sparse system matrix $A \in \mathbb{R}^{N \times N}$ and right hand side vector $b \in \mathbb{R}^N$.

Next, we consider the case of P subdomains. To generate a partition of P overlapping subdomains $\{\Omega_i\}_{i=1}^P$ of equal size, we employ the space-filling curve approach and in principle just map our d-dimensional (interior) grid points $x_k \in \Omega_h$ by means of the inverse, discrete space-filling curve s_n^{-1} with sufficiently large n to the one-dimensional unit interval $[0, 1]$. Then we simply partition the one-dimensional, totally ordered sequence of N points into a consecutive one-dimensional set of disjoint subsets of approximate size N/P each. To this end, we first determine the remainder $r := N - P\lfloor N/P \rfloor$. This gives us the number r of subdomains which have to possess $\lfloor N/P \rfloor + 1$ grid points, whereas the remaining $P - r$

subdomains possess just $\lfloor N/P \rfloor$ grid points. Thus, with

$$(2.10) \qquad \tilde{N}_i := \lfloor N/P \rfloor + 1, i = 1, \ldots, r \text{ and } \tilde{N}_i = \lfloor N/P \rfloor, i = r+1, \ldots, P,$$

we assign the first $\tilde{N}_1$ points to the set $\tilde{\Omega}_1$, the second $\tilde{N}_2$ points to the set $\tilde{\Omega}_2$, and so on. Since the $\tilde{N}_i$ differ at most by one, we obtain a perfectly balanced partition $\{\tilde{\Omega}_i\}_{i=1}^{P}$. The basic partitioning approach by means of the Hilbert curve is shown for the two-dimensional case in Figures 4 and 5. Note here that the resolution of the discrete isotropic space-filling curve is chosen as the one which belongs to the largest value $\max_{j=1,\ldots,d} l_j$ of the entries of the level parameter $l = (l_1, \ldots, l_d)$, i.e. to the finest resolution in case of an anisotropic grid.
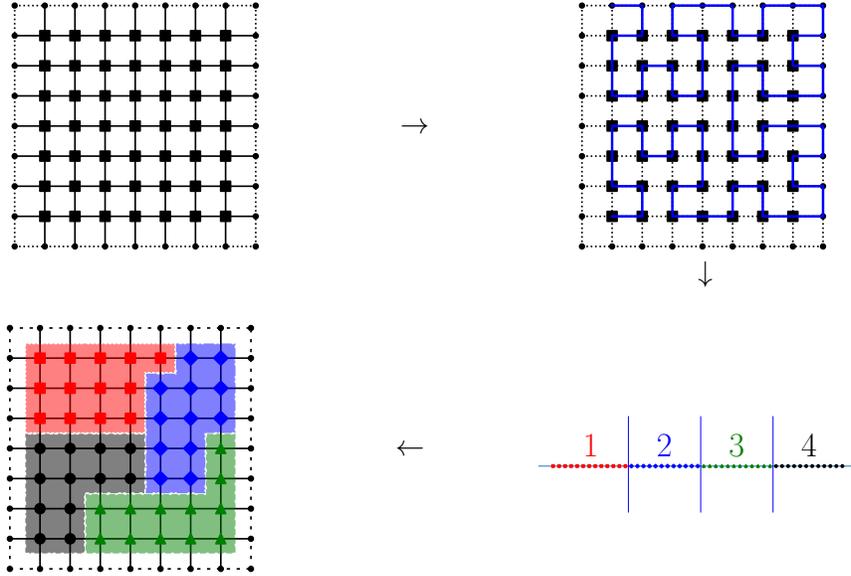
FIGURE 4. Decomposition of an isotropic grid with $l = (3,3)$ by the Hilbert curve approach.

In practice, we do not use the grid points $x_k = (x_{k_1}, \ldots, x_{k_d}), \in \mathbb{R}^d$ with $x_{k_j} = k_j 2^{-l_j}$ in the partitioning process, but merely their indices $k := (k_1, \ldots, k_d) \in \mathbb{N}^d$, $k_j = 1, \ldots 2^{l_j} - 1$, which we store in an N-sized vector, e.g. in lexicographical order. The space-filling curve mapping then boils down to a simple *sorting* of the entries of this vector with respect to the order relation of the indices induced by the space-filling curve mapping. To this end, we just need the one-dimensional relation $<$ of the space-filling curve ordering of the mapped indices. This is realized by means of a comparison function $cmp((k_1, \ldots, k_d), (k'_1, \ldots, k'_d))$ for any two indices $(k_1, \ldots, k_d)$ and $(k'_1, \ldots, k'_d)$, which returns true if the index $(k_1, \ldots, k_d)$ is situated before the index $(k'_1, \ldots, k'_d)$ on the space-filling curve and which returns false in the else clause. For the case of the Hilbert curve, the implementation follows mainly [B71, M00, Sp20, Sh20]. Our variant is based on bit-operations on the indices $(k_1, \ldots, k_d)$ and $(k'_1, \ldots, k'_d)$ only and is non-recursive. It corresponds to a 2^d-tree traversal with one
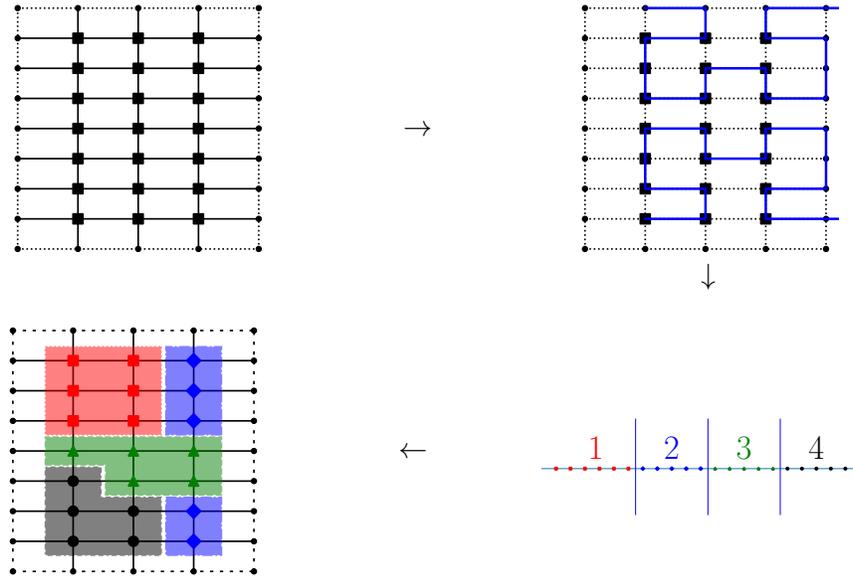
FIGURE 5. Decomposition of an anisotropic grid with $l = (2,3)$ by the Hilbert curve approach.

rotation per depth of the tree on the bit level and a bit comparison. This avoids first explicitly calculating and then comparing the associated keys. Note here that, if we would first compute the full two keys, we would have to completely traverse the tree down to the finest scale of the discrete Hilbert curve and could only then compare the associated numbers. In our direct comparison we however descend iteratively down the tree and stop the traversal as soon as we detect on the current level that the two considered indices belong to different orthants. This is much faster. Moreover it avoids a problem which might occur for strongly anisotropic grids with the isotropic Hilbert curve: For example, in the case of a grid with the multivariate level parameter $l = (L, 1, \ldots, 1)$ we still would have to deal with the universe of 2^{dL} possible indices and keys due to the *isotropy* of the d-dimensional Hilbert curve, but we employ for our anisotropic grid only 2^L indices altogether. This universe of keys for the Hilbert curve becomes, for larger d, with rising L quickly too large for any conventional data type of the associated keys. Furthermore the keys would contain large gaps and voids in this universe, since only 2^L keys are present anyway. Our approach of using just a comparison relation without explicitly computing the two keys for the two indices still allows sorting of the indices. A key is then just given as the position in the sorted index vector of length $N = 2^L$, i.e. we only need 2^L keys in our position-universe which now contains no voids or gaps at all. The sorting is done by introsort which has an average and worst-case complexity of $O(N \log(N))$. We store the full vector on each processor redundantly and perform the sorting redundantly as well. We will mainly consider the Hilbert curve ordering in our experiments.

In a next step, we enlarge the subdomains $\tilde{\Omega}_j$, i.e. the corresponding subsets of grid point indices, in a specific way to create overlap. This is not done as in conventional, geometry-based overlapping domain decomposition methods by adding a d-dimensional mesh-stripe with diameter δ of grid cells at the boundary of the d-dimensional subdomains that are associated to the sets $\tilde{\Omega}_i$. Instead, we deliberately stick to the one-dimensional setting which is induced by the space-filling curve: We choose an overlap parameter $\gamma \in \mathbb{R}, \gamma > 0$, and enlarge the index sets $\tilde{\Omega}_i$ as

$$(2.11) \qquad \Omega_i := \tilde{\Omega}_i \cup \bigcup_{k=1}^{\lfloor \gamma \rfloor} \left(\tilde{\Omega}_{i-k} \cup \tilde{\Omega}_{i+k} \right) \cup \tilde{\Omega}_{i-\lfloor\gamma\rfloor-1}^{\eta,+} \cup \tilde{\Omega}_{i+\lfloor\gamma\rfloor+1}^{\eta,-}.$$

Here, with $\eta := \gamma - \lfloor \gamma \rfloor$, the set $\tilde{\Omega}_k^{\eta,+}$ is the subset of $\tilde{\Omega}_k$ which contains its last $\lceil \eta N_k \rceil$ indices with respect to the space-filling curve ordering, while the set $\tilde{\Omega}_k^{\eta,-}$ is the subset of $\tilde{\Omega}_k$ which contains its first $\lfloor \eta N_k \rfloor$ indices. For example, for $\gamma = 1$, we add to $\tilde{\Omega}_i$ exactly the two neighboring index sets $\tilde{\Omega}_{i+1}$ and $\tilde{\Omega}_{i-1}$, for $\gamma = 2$, we add the four sets $\tilde{\Omega}_{i+1}, \tilde{\Omega}_{i+2}$ and $\tilde{\Omega}_{i-1}, \tilde{\Omega}_{i-2}$. For $\gamma = 0.5$ we would add those halves of the indices of $\tilde{\Omega}_{i+1}$ and $\tilde{\Omega}_{i-1}$ which are the ones closer to $\tilde{\Omega}_i$, etc. Moreover, to avoid any special treatment for the first and last few $\tilde{\Omega}_i, i = 1, 2, ..$ and $i = P, P-1, ..$, we cyclically close the enumeration of the subsets, i.e. the left neighbor of $\tilde{\Omega}_1$ is set as $\tilde{\Omega}_P$ and the right neighbor of $\tilde{\Omega}_P$ is set as $\tilde{\Omega}_1$. Note that, besides γ, also the specific values of the $\tilde{N}_i$ enter here. Examples of the enlargement of the index sets from $\tilde{\Omega}_i$ to Ω_i with $\gamma = 0.25$ are given in Figures 6 and 7. Here we show the induced domains Ω_1 and Ω_3 only.
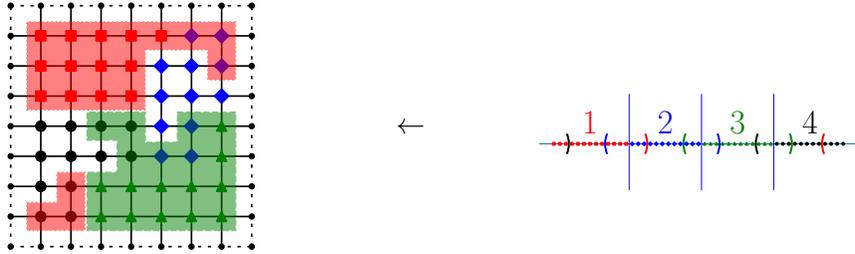
FIGURE 6. Enlargement of the subdomains with the Hilbert space-filling curve and associated overlapping domain decomposition, isotropic grid with $l = (3, 3)$.

This way, an overlapping partition $\{\Omega_i\}_{i=1}^P$ is finally constructed. Note at this point that, depending on N, P, γ and the respective space-filling curve type, each subdomain of the associated decomposition is not necessarily fully connected, i.e. there can exist some sub-domains with point distributions that geometrically indicate some further separation of the subdomain, see e.g. Ω_1 in Figure 6. But since we merely deal with index sets and not with geometric subdomains, this causes no practical issues. Note furthermore that there is not necessarily always a complete overlap but sometimes just a partial overlap between two adjacent subdomains being created by our space-filling curve approach. But this also causes
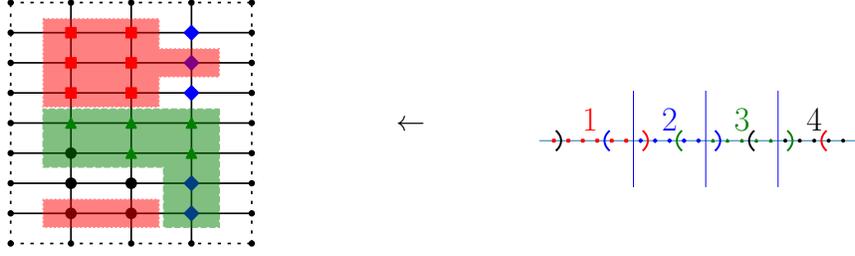
FIGURE 7. Enlargement of the subdomains with the Hilbert space-filling curve and associated overlapping domain decomposition, anisotropic grid with $l = (2, 3)$.

no practical issues. In contrast to many other domain decomposition methods, where a goal is to make the overlap as small as possible, our approach rather aims at larger overlaps along the space-filling curve, which later gives the sufficient redundancy of stored data that will be needed for fault tolerance.

Finally, we set up our coarse space problem. To this end, the size N_0 of the coarse problem is given via the number P of subdomains and the number of degrees of freedom n_i considered on the coarse scale per subdomain, i.e. $N_0 := \sum_{i=1}^{P} n_i$. If we let all n_i be equal to a fixed integer $q \in \{1, \ldots, \lfloor \frac{N}{P} \rfloor\}$, i.e. $n_i = q$, then $N_0 = q \cdot P$. The mapping from the fine grid to the coarse space is now given by means of the restriction matrix $R_0 \in \mathbb{R}^{N_0 \times N}$ and its entries. Again, we avoid any geometric information here, i.e. we do not follow the conventional approach of a coarse grid with associated coarse piecewise linear basis functions. Instead we derive the coarse scale problem in a purely *algebraic* way. For that, we resort to the non-overlapping partition $\{\tilde{\Omega}_i\}_{i=1}^{P}$ and assign the values of zero and one to the entries of the restriction matrix as follows: Let q be the constant number of coarse level degrees of freedom per subdomain. With $\tilde{N}_i = \lfloor N/P \rfloor + 1$ if $i \leq (N \mod P)$ and $\tilde{N}_i = \lfloor N/P \rfloor$ otherwise, which denotes the size of $\tilde{\Omega}_i$, we have the index sets $\tilde{\Omega}_i = \left\{ \sum_{j=1}^{i-1} \tilde{N}_j + 1, \ldots, \sum_{j=1}^{i} \tilde{N}_j \right\}$. Now let $\tilde{\Omega}_{i,m}$ be the m-th subset of $\tilde{\Omega}_i$ with respect to the size q in the space-filling curve ordering, i.e.

$$\tilde{\Omega}_{i,m} = \left\{ \sum_{j=1}^{i-1} \tilde{N}_j + \sum_{n=1}^{m-1} \tilde{N}_{i,n} + 1, ..., \sum_{j=1}^{i-1} \tilde{N}_j + \sum_{n=1}^{m} \tilde{N}_{i,n} \right\},$$

with associated size $\tilde{N}_{i,m}$ for which, with q coarse points per domain, we have

$$\tilde{N}_{i,m} = \left(\begin{array}{ll} \lfloor \tilde{N}_i/q \rfloor + 1 & \text{if } m \leq (\tilde{N}_i \mod q) \\ \lfloor \tilde{N}_i/q \rfloor & \text{otherwise.} \end{array} \right.$$

Then, for $i = 1, ..., P, m = 1, ..., q, j = 1, ..., N$, we have

(2.12) $$(R_0)_{((i-1)q+m,j)} = \left(\begin{array}{ll} 1 & \text{if } j \in \tilde{\Omega}_{i,m} \\ 0 & \text{otherwise.} \end{array} \right.$$

This way a basis is implicitly generated on the coarse scale: Each basis function is constant in the part of each subdomain of the non-overlapping partition which belongs to the $\tilde{\Omega}_{i,m}$, where q piecewise constant basis functions with support on $\tilde{\Omega}_{i,m}$ are associated to each $\tilde{\Omega}_i$.

The coarse scale problem is then set up via the Galerkin approach as

$$(2.13) \qquad\qquad A_0 = R_0 A R_0^T.$$

Here we follow the Bank-Holst technique [BH03] and store a redundant copy of it on each of the P processors together with the respective subdomain problem. This way, the coarse problem is formally avoided. Moreover the coarse problem is redundantly solved on each processor. It interacts with the respective subproblem solver in an additive, parallel way, i.e. we solve the global coarse scale problem and the local fine subdomain problem independently of each other on the processor.

Finally, we have to deal with the overcounting of unknowns due to the overlapping of fine grid subdomains. To this end, we resort to the partition of unity on the fine level

$$(2.14) \qquad\qquad I = \sum_{i=1}^{P} R_i^T D_i R_i$$

with properly chosen diagonal matrices $D_i \in \mathbb{R}^{N_i \times N_i}$. This leads to the two-level domain decomposition operator

$$(2.15) \qquad C_{(2),D}^{-1} := R_0^T A_0^{-1} R_0 + C_{(1),D}^{-1} = \sum_{i=0}^{P} R_i^T D_i A_i^{-1} R_i, \quad D_0 = I,$$

with weighted one-level operator

$$(2.16) \qquad\qquad C_{(1),D}^{-1} := \sum_{i=1}^{P} R_i^T D_i A_i^{-1} R_i,$$

and to the iteration in the k-th step

$$x^{k+1} = x^k + C_{(2),D}^{-1}(b - Ax^k).$$

Up to the coarse scale part, this resembles just the restricted additive Schwarz method of [CS99], which is basically a weighted overlapping block-Jacobi/Richardson method. With the choice $D_0 = I$, the associated weighted balanced variant is then

$$C_{(2),D,bal}^{-1} := G^T C_{(1),D}^{-1} G + F,$$

with G and F from (2.6). The modifications of F, G and thus $C_{(2),D,bal}^{-1}$ for general D_0 are obvious.

There are different choices of the D_i's since the above condition (2.14) does not have a unique solution. A natural choice is here

$$(2.17) \qquad\qquad (D_i)_{j,j} = 1/|\left\{\Omega_{i'}, i' = 1, \ldots P : j \in \Omega_{i'}\right\}|,$$

which locally takes for each index $j \in \Omega_i$ the number of domains that overlap this index into account. Note that, for such general diagonal matrices D_i, the associated $C_{(2),D}$ is not

a symmetric operator. If however the D_i are chosen as $\omega_i I_i$ with scalar positive weights ω_i, then, on the one hand, the partition of unity property is lost, but, on the other hand, symmetry is regained and we have a weighted, two-level overlapping additive Schwarz method, which can also used as a preconditioner for the conjugate gradient method. To this end, a sensible choice is

$$(2.18) \qquad \omega_i = \max_j (D_i)_{(j,j)}$$

with D_i from (2.17).

At this point a closer inspection of the relation of the overlap parameter γ to the entries of D_i or ω_i, respectively, is in order. Of course we may choose any value for γ and we may select quite freely the D_i or ω_i. However, if we want to achieve fault tolerance and thus employ a larger overlap of the subdomains, which are created as described above via our space-filling curve construction to allow for proper redundant storage and for data recovery, it is sensible to restrict ourselves to values of γ that are integer multiples of 0.5. In this case, every fine grid point is overlapped by exactly $2\gamma + 1$ subdomains, whereas, if γ is not an integer multiple of 0.5, the number of subdomains that overlap a point can indeed be different for different points of the same subdomain. Additionally, integer multiples of 0.5 for γ are the natural redundancy thresholds of our fault-tolerant space-filling curve algorithm. In particular for $\frac{1}{2}n \leq \gamma < \frac{1}{2}(n+1), n \in \mathbb{N}$, our fault-tolerant algorithm can recover from faults occurring for at most n neighboring processors in the same iteration. Thus, with these considerations, overlap parameter values of the form $\gamma = \frac{1}{2}n, n \in \mathbb{N}$, are the ones that are most relevant for proper redundant storage, for data recovery and thus for fault tolerance in practice. Additionally, such a specific choice of γ has a direct consequence on the resulting ω_i and D_i. We have the following Lemma.

Lemma 2.1. *Let $d \geq 1$ be arbitrary and let $\gamma = \frac{1}{2}n$, where $n \in \mathbb{N}, n \leq P - 1$. Then, with $c := 2\gamma + 1$, there holds*

$$D_i = \omega_i I = \frac{1}{c} I$$

for all $i = 1, \ldots, P$ and any type of space filling curve employed.

Proof. With the notation from above we need to show that we have

$$| \{ \Omega_i, i = 1, \ldots P : j \in \Omega_i \} | = c = 2\gamma + 1$$

for all $j = 1, \ldots, N$, if $\gamma = \frac{1}{2}n$. To this end, let j be arbitrary and $m \in \{1, \ldots, P\}$ such that $j \in \tilde{\Omega}_m$. Since the $\tilde{\Omega}_i$ are disjoint, there is exactly one such m. Now recall (2.11), i.e.

$$(2.19) \qquad \Omega_i := \tilde{\Omega}_i \cup \bigcup_{k=1}^{\lfloor \gamma \rfloor} \left(\tilde{\Omega}_{i-k} \cup \tilde{\Omega}_{i+k} \right) \cup \tilde{\Omega}_{i-\lfloor \gamma \rfloor - 1}^{\eta,+} \cup \tilde{\Omega}_{i+\lfloor \gamma \rfloor + 1}^{\eta,-},$$

where $\eta = \gamma - \lfloor \gamma \rfloor$.

First consider the case where n is even. This implies that γ is an integer and $\lfloor \gamma \rfloor = \gamma$. We obtain $\eta = 0$ and therefore $\tilde{\Omega}_{i-\lfloor \gamma \rfloor - 1}^{\eta,+} = \tilde{\Omega}_{i+\lfloor \gamma \rfloor + 1}^{\eta,-} = \emptyset$ for all i. Furthermore, due to the

assumption $n \leq P - 1$, we have $\gamma \leq \frac{P-1}{2}$ and, since the $\tilde{\Omega}_i$ are disjoint by construction, the sets of the union in (2.19) are disjoint. This implies that $j \in \Omega_i$ if and only if $j \in \tilde{\Omega}_i$ or $j \in \tilde{\Omega}_{i-k}$ or $j \in \tilde{\Omega}_{i-k}$ for some $k = 1, \ldots, \gamma$. Equivalently, $j \in \tilde{\Omega}_i$ if and only if $i = m, i - k = m$ or $i + k = m$ for $k = 1, \ldots, \gamma$. Hence there are exactly $1 + \gamma + \gamma = 2\gamma + 1$ values for i, namely m and $m + 1, \ldots, m + \gamma$ and $m - 1, \ldots, m - \gamma$. Therefore, we obtain

$$| \{\Omega_i, i = 1, \ldots P : j \in \Omega_i\} | = 2\gamma + 1.$$

Now consider the case where n is odd. Then $\eta = \frac{1}{2}$ and we have $\tilde{\Omega}_i^{\eta,-} \cup \Omega_i^{\eta,+} = \tilde{\Omega}_i$ and $\tilde{\Omega}_i^{\eta,-} \cap \tilde{\Omega}_i^{\eta,+} = \emptyset$ for all i, since $\tilde{\Omega}_i^{\eta,-}$ contains the first $\lfloor \eta N_i \rfloor = \lfloor \frac{N_i}{2} \rfloor$ and $\tilde{\Omega}_i^{\eta,+}$ contains the last $\lceil \eta N_i \rceil = \lceil \frac{N_i}{2} \rceil$ indices of $\tilde{\Omega}_i$, respectively. Thus there is exactly one i such that $j \in \left(\tilde{\Omega}_{i-\lfloor \gamma \rfloor - 1}^{\eta,+} \cup \tilde{\Omega}_{i+\lfloor \gamma \rfloor + 1}^{\eta,-} \right)$. Furthermore, since $\gamma \leq \frac{P-1}{2}$, the sets of the union in (2.19) are again disjoint. Using the same argument as for the previous case where γ was an integer, there are now exactly $2\lfloor \gamma \rfloor + 1$ choices for i such that $j \in \left(\tilde{\Omega}_i \cup \bigcup_{k=1}^{\lfloor \gamma \rfloor} \left(\tilde{\Omega}_{i-k} \cup \tilde{\Omega}_{i+k} \right) \right)$. Consequently, we also obtain for the case where n is odd that

$$| \{\Omega_i, i = 1, \ldots P : j \in \Omega_i\} | = 2\lfloor \gamma \rfloor + 1 + 1 = 2(\lfloor \gamma \rfloor + 0.5) + 1$$
$$= 2(\lfloor \gamma \rfloor + \eta) + 1 = 2\gamma + 1.$$

Altogether, this proves the assertion. $\square$

We thus have seen that the general weightings D_i and ω_i for the fine scale subdomains are the *same* and even boil down to a simple constant global scaling with the factor $1/c = 1/(2\gamma + 1) = 1/(n + 1)$ uniformly for all subdomains $i = 1, \ldots, P$, if we a priorily choose $\gamma = \frac{1}{2}n$, $n \in \mathbb{N}$. Recall that the single weight for the coarse scale problem was set to one in the first place. Thus we regain symmetry of the corresponding operator also for the D_i-choice, as long as we employ values for γ that are integer multiples of 0.5. This is a direct consequence of our one-dimensional domain decomposition construction due to the space filling curve approach involving the proper rounding in the definition of the overlapping subdomains Ω_i and using the value of γ not absolutely but *relatively* with respect to the size of the non-overlapping subdomains $\tilde{\Omega}_i$, which are still allowed to be of different size. Moreover this property is independent of the respective type of space-filling curve and merely a consequence of our construction of the overlaps. Such symmetry of the operator can not be obtained so easily for the general D_i-weighting within the conventional geometric domain decomposition approach for $d > 1$. Note also that the result of Lemma 2.1 holds analogously for more general geometries of Ω beyond the d-dimensional tensor product domain. Note furthermore that, for the choice $\gamma = \frac{1}{2}n$, $n \in \mathbb{N}$, the weighted one-level operator (2.16) becomes just a scaled version of the conventional one-level operator (2.4), i.e.

$$C_{(1),D}^{-1} = \frac{1}{2\gamma + 1} C_{(1)}^{-1}.$$

Consequently, if we choose $D_0 = I$, we obtain

$$C_{(2),D}^{-1} = R_0^T A_0^{-1} R_0 + C_{(1),D}^{-1} = R_0^T A_0^{-1} R_0 + \frac{1}{2\gamma + 1} C_{(1)}^{-1}.$$

and

$$C_{(2),D,bal}^{-1} := G^T C_{(1),D}^{-1} G + F = \frac{1}{2\gamma + 1} G^T C_{(1)}^{-1} G + F,$$

which now just resemble a fine-level-rescaled variant of the conventional two-level operator and of its balanced version, respectively.

In any case, we obtain the damped linear two-level Schwarz/Richardson-type iteration as given in Algorithm 2, where the setup phase, which is executed just once before the iteration starts, is described in Algorithm 1. Its convergence property is well known, since it is a special case of the additive Schwarz iteration as studied in [GO95]. There, in Theorem 2, it is stated that the damped additive method associated to, e.g., our decomposition (2.15) indeed converges in the energy norm for $0 < \xi < 2/\lambda_{\max}$ with convergence rate

$$\rho = \max\{|1 - \xi\lambda_{\min}|, |1 - \xi\lambda_{\max}|\},$$

where $\lambda_{\min}$ and $\lambda_{\max}$ are the smallest and largest eigenvalues of $C_{(2),D}^{-1} A$, provided that $C_{(2),D}^{-1}$ is a symmetric operator, which for example is the case for the choice (2.18). Moreover the convergence rate takes the minimum

$$(2.20) \qquad\qquad \rho^* = 1 - \frac{2}{1 + \kappa} \quad \text{for} \quad \xi^* = \frac{2}{\lambda_{\min} + \lambda_{\max}}$$

with $\kappa = \lambda_{\max}/\lambda_{\min}$. The proof is exactly the same as for the conventional Richardson iteration. To this end, the numbers $\lambda_{\max}$ and $\lambda_{\min}$ need to be explicitly known to have the optimal damping parameter ξ^*, which is of course not the case in most practical applications. Then good estimates, especially for $\lambda_{\max}$, are needed to still obtain a convergent iteration scheme, albeit with a somewhat reduced convergence rate. Note at this point that for the general non-symmetric case, i.e. for the general choice (2.17), this convergence theory does not apply. In practice however, convergence can still be observed.

Here the following remarks are in order: We store and sort on each processor redundantly the full index vector sfc_index of size N. The sorting could be done in parallel and non-redundantly, but the redundant sorting is very cheap anyway. The two numbers ta_i, tb_i need, depending on the overlap factor γ, to be interpreted properly for the first and last few i, since we cyclically close the enumeration of the subsets Ω_i modulo P to avoid any special treatment there. The relevant parts of A that belong to Ω_i are the full rows of A with indices $j \in [ta_i, tb_i]$. They are stored in compressed row storage (CRS) format. But these full rows can also contain entries whose *column* index might belong to another processor, i.e. the column indices of such entries of the matrix need to be known to processor i to be able to set up the full rows in the first place. Therefore, we store on each processor a map of the global indices which this processor does not own but which are geometric neighbors tied to an index on this processor due to the non-zero matrix entries of the respective rows. This information is then used to determine the non-zero entries of the complete rows associated to Ω_i in the next step and can be deleted after that. It is not necessary to store the relevant parts of $R_i, i = 1, \ldots, P$. The matrices $R_i^T, i = 1, \ldots, P$, and the the corresponding restriction matrices $R_i, i = 1, \ldots, P$, are implicitly available

---

Algorithm 1 Overlapping two-level additive Schwarz iteration with space-filling curve: Setup phase.

1: **on** every processor $i = 1, \ldots, P$ **do**

2:     Set input parameters: d, $l = (l_1, \ldots, l_d)$, P, γ, q, ξ, type of space-filling curve.

3:     Derive N from (2.8), set $\tilde{N}_i, i = 1, \ldots, P$ as in (2.10) and set $N_0 = q \cdot P$.

4:     Compute the index vector sfc_index of length N from the d-dimensional grid point indices $k = (k_1, \ldots, k_d)$, $k_j = 1, \ldots, 2^{l_j} - 1$, $j = 1, \ldots d$, according to the space-filling curve by means of $cmp((k_1, ..., k_d), (k'_1, ..., k'_d))$ and introsort.

5:     Derive the disjoint subdomain index sets $\{\tilde{\Omega}_i\}_{i=1}^P$ by splitting the overall index set into P subsets $\tilde{\Omega}_i$ of consecutive indices, each of size $\tilde{N}_i$. This is simply done by storing two integers $\tilde{ta}_i, \tilde{tb}_i$, which indicate where the local index sequence of $\tilde{\Omega}_i$ starts and ends in sfc_index.

6:     Derive the overlapping subdomain index sets $\{\Omega_i\}_{i=1}^P$ by enlarging the $\tilde{\Omega}_i$ with γ as in (2.11). Again, this is simply done by storing two integers ta_i, tb_i, which indicate where the local index sequence of Ω_i starts and ends in sfc_index.

7:     Set up a map to neighboring grid points that are not in Ω_i, i.e. store their global indices, to later determine the column entries of the stiffness matrix that are situated outside of Ω_i.

8:     Set the rows of A that belong to Ω_i, i.e. store the rows of A with indices $j \in [ta_i, tb_i]$ in CRS format.

9:     Initialize the part of the starting iterate x^0 and the part of b that belong to Ω_i.

10:     Derive the rows of the matrix R_0 from (2.12) with indices $j \in [(i-1)q + 1, \ldots, iq]$ and store them in CRS format.

11:     Compute the rows of the coarse scale matrix A_0 as in (2.13) that belong to Ω_i, i.e. with the indices $j \in [(i-1)q + 1, \ldots, iq]$, and store them in CRS format.

12: **end on**

---

using the local part of Ω_i of the sfc_index vector, since R_i^T is just the extension-by-zero map and R_i the corresponding restriction. Note here that the product of two matrices in CRS format can easily be stored in CRS format as well. This facilitates the setup of the coarse scale matrix. In this step, the q rows of A_0 that belong to Ω_i, i.e. those with the indices $j \in [(i-1)q + 1, \ldots, iq]$, are first locally generated. Then, to create the complete A_0 redundantly on all processors i, an all-to-all communication step is necessary. For Algorithm 2 we have three types of communication: First, an overlap-based data exchange between processors i and $i \pm 1, \ldots, i \pm \lceil \gamma \rceil$, which are neighbors with respect to the space-filling curve ordering, is needed for the update of the $R^T x^{k+1}$. Second, an A-based data exchange between processors is necessary for the parallel matrix-vector product Ax^k. And third, all-to-all communcation is to be employed for the exchange of the coarse residual data and the coarse update data due to the Bank-Holst technique. Note finally that we store on each processor the local part of x and b, which belongs to Ω_i and not just to $\tilde{\Omega}_i$. This avoids one communication step of the overlap data, which would be necessary otherwise.

Algorithm 2 Overlapping two-level additive Schwarz iteration with space-filling curve: Linear iteration.

1: Set k=1.
2: **while** not converged **do**
3:     **on** every processor $i = 1, \ldots, P$ **do**
4:         Compute the part $r_i^k = R_i r^k$ of the residual $r^k = b - Ax^k$ that belongs to Ω_i.
5:         Solve the local subproblems

$$A_i d_i^k = r_i^k.$$

6:         Solve redundantly the coarse scale problem

$$A_0 d_0^k = R_0 r^k.$$

7:         Compute the part $x_i^{k+1} = R_i x^{k+1}$ of the correction

$$x^{k+1} = x^k + \sum_{i=0}^{P} \xi R_i^T D_i d_i^k$$

that belongs to Ω_i.

8:     **end on**
9: **end while**

Thus, in contrast to most conventional parallel domain decomposition implementations, we trade communication for (moderate) storage costs.

This linear two-level additive Schwarz iteration can also be used as a preconditioner for the conjugate gradient iteration, which results in a substantial improvement in convergence. In the symmetric case, an error reduction factor of $2(1 - 2/(1 + \sqrt{\kappa}))$ per iteration step is then obtained in contrast to the reduction factor of $1 - 2/(1 + \kappa)$ from (2.20). Moreover a damping parameter is no longer necessary, since the respective two-level additive Schwarz operator from (2.15) with the weights (2.18) is merely used to improve the condition number and no longer needs to be convergent by itself.

Then the basic conjugate gradient iteration must additionally be implemented in parallel. This can easily be done in a way analogous to the overlapping Schwarz iteration above, involving a further parallel vector product based on the overlapping decomposition $\{\Omega_i\}_{i=1}^{P}$. The details are obvious and are left to the reader. Note here again that, for general diagonal matrices D_i, the associated preconditioner is no longer symmetric, while it is in the case $D_i = \omega_i I$. This can cause both theoretical and practical problems for the conjugate gradient iteration. Then, instead of the conventional conjugate gradient method, we could resort to the *flexible* conjugate gradient method, which provable works also in the non-symmetric case, see [BDK15] and the references cited therein. But, as already shown in Lemma 2.1, this issue is completely avoided with the choice $\gamma = \frac{1}{2}n, n \in \mathbb{N}$.

3. A fault-tolerant domain decomposition method

3.1. **Fault tolerance and randomized subspace correction.** Now we will focus on algorithm-based fault tolerance. Usually, to obtain fault-tolerant parallel methods, techniques are employed which are based on functional repetition and on check pointing. They are however prone to scaling issues, i.e. naive versions of conventional resilience techniques will not scale to the peta- or exascale regime, and more advanced techniques need to be devised. An overview of the state of the art for resiliency in numerical algorithm design for extreme scale simulations is given in [A20]. Depending on the PDE-problem to be treated, certain error-oblivious algorithms had been developed that can recover from errors without assistance as long as these errors do not occur too frequently. Besides time stepping methods with check pointing for parabolic and hyperbolic problems, various iterative linear methods and fixed-point solvers are able to execute to completion, especially in the setting of elliptic PDEs. To this end, the view of asynchronous iterative methods can be taken. But to make, for example, a parallel multigrid or multilevel solver fault-tolerant is still a challenging task, see [HGRW16, Sta19] for the two- and three-dimensional case. Furthermore, albeit numerical experiments often show good convergence and impressive fault-tolerance properties, most asynchronous iterative methods are not just simple fixed-point iterations anymore and the development of a sound convergence theory for such algorithms is an issue.

For higher-dimensional time-dependent problems, fault mitigation on the level of the combination method has been experimentally tried for a linear advection equation in [HHLS14], for the Lattice Boltzmann method and a solid fuel ignition problem in [ASHH16], and for a gyrokinetic electromagnetic plasma application in [PBG14, OHHBP17, LOP20], where existing (parallel) codes were used as black box solver for each of the subproblems and adaptive time stepping methods were employed. But again, the considered numerical approaches (Lax-Wendroff finite differences, the method of lines and Runge-Kutta 4th order in the GENE code, the two-step splitting procedure in the Taxila Lattice Boltzmann code) in general do not allow for a simple and clean convergence theory, nor do they allow for a Hilbert space structure due to the involved time and advection operators.

However, if there is a direct Hilbert space structure as for elliptic problems, algorithm-based fault tolerance can be interpreted in the framework of *stochastic subspace correction* algorithms for which in [GO95, GO12, GO16, GO18] we recently developed a general theoretical foundation for their convergence rates in expectation. Indeed, for a conventional domain decomposition approach, we employed our stochastic subspace correction theory to show algorithm-based fault tolerance theoretically and in practice under independence assumptions for the random failure of subdomain solves in [GO20]. The main idea is to switch from deterministic error reduction estimates of additive and multiplicative Schwarz methods as subspace splitting techniques to error contraction and thus to convergence *in expectation*. This way, convergence behavior and convergence rates can indeed be shown

for certain iterative methods in a faulty environment, provided that specific assumptions on the occurrence and the distribution of faults and errors are fulfilled.

To be precise, in [GO20], we considered linear iterative, additive Schwarz methods for general stable space splittings, which we considered as subspace correction methods. For the setting of the two-level domain decomposition of (2.5), the linear iteration reads as follows in our notation: In the k-th iteration step, a certain index set $I_k \subset \{0, 1, \ldots, P\}$ is selected. For each $i \in I_k$ the corresponding subproblem

$$(3.21) \qquad\qquad A_i d_i^k = R_i(b - Ax^k)$$

is solved for d_i^k, and an update of the form

$$(3.22) \qquad\qquad x^{k+1} = x^k + \sum_{i \in I_k} \xi_{k,i} R_i^T d_i^k \qquad k = 0, 1, \ldots$$

is performed. At the beginning we may set x^0 to zero without loss of generality. In the simplest case the relaxation parameters $\xi_{k,i}$ are chosen independently of $x^{(k)}$ and we then obtain an, in general, non-stationary but linear iteration scheme. The iteration (3.22) subsumes different standard algorithms such as the multiplicative (or sequential) Schwarz method, where in each step a single subproblem (3.21) is solved ($|I_k| = 1$), the additive (or parallel) Schwarz method, where all $P + 1$ subproblems are solved simultaneously ($I_k = \{0, 1, \ldots, P\}$), and intermediate block-iterative schemes ($1 < |I_k| < P + 1$). Here and in the following, $|I|$ denotes the cardinality of the index set I. The recursion (3.22) therefore represents a one-step iterative method, i.e. only the current iterate x^k needs to be available for the update step.

In [GO20], we specifically discussed stochastic versions of (3.22), where the sets I_k are chosen *randomly*. To this end, we assumed that

A I_k is a uniformly at random chosen subset of size p_k in $\{0, 1, \ldots, P\}$, i.e., $|I_k| = p_k$ and $\mathbb{P}(i \in I_m) = \mathbb{P}(i' \in I_k)$ for all $i, i' \in \{0, 1, \ldots, P\}$.

B The choice of I_k is independent for different k.

We then considered expectations of squared energy error norms for iterations with any fixed but arbitrary sequence $\{p_k\}$ and derived the following result:

Theorem 3.1. *Let the relaxation parameters in (3.22) be given by $\xi_{k,i} := \xi\omega_i$, $i = 0, 1, \ldots, P$, where $0 < \xi < 2/\lambda_{\max}$. Furthermore let the random sets I_k of size p_k be selected in agreement with **A** and let **B** hold. Then the algorithm (3.22) converges in expectation for any $x \in \mathbb{R}^N$ and*

$$(3.23) \quad \mathbb{E}(\|x - x^k\|^2) \le \prod_{s=0}^{k-1} \left(1 - \frac{\lambda_{\max} \, \xi \, (2 - \lambda_{\max} \, \xi) \, p_s}{\kappa \, (P+1)}\right) \|x - x^0\|^2, \qquad k = 1, 2, \ldots$$

where $\kappa := \lambda_{\max}/\lambda_{\min}$ is the condition number of the underlying ω_i-weighted decomposition.

Here $\|x\| := \sqrt{x^T A x}$ denotes the discrete energy norm which is associated to the FEM-matrix A of the discretized Laplace operator. For a detailed proof in the case of general space splittings see [GO20]. Here the norm equivalency

$$(3.24) \qquad \lambda_{\min} \, v^T C_{(2),\omega} A v \le v^T A v \le \lambda_{\max} \, v^T C_{(2),\omega} A v, \qquad v \in \mathbb{R}^N$$

has to hold with $0 < \lambda_{\min} \le \lambda_{\max} \le \infty$ and positive weights $\omega = \{\omega_i > 0\}_{i=0,1,\dots,P}$, where we employ the ω-weighted, symmetric, additive Schwarz operator

$$(3.25) \qquad C_{(2),\omega}^{-1} := \sum_{i=0}^{P} \omega_i R_i^T A_i^{-1} R_i,$$

compare (2.15) with $D_i = \omega_i I$. Now any application of the linear iteration (3.22) with theoretical guarantees according to Theorem 3.1 requires knowledge of suitable weights ω_i, and an upper bound $\bar{\lambda}$ for the stability constant $\lambda_{\max}$ in order to choose the value of ξ, whereas information about p_k, i.e. the size of I_k, is not crucial. Numerical experiments for model problems with different values $\xi \in (0, 2/\lambda_{\max})$ suggest that the iteration count is sensitive to the choice of ξ and that overrelaxation often gives better results. The weights ω_i can be considered as scaling parameters that can be used to improve the stability constants $\lambda_{\max}, \lambda_{\min}$, and thus the condition number κ of the splitting.

The application of the above convergence estimate to fault tolerance will focus on the situation

$$1 << p_k \le p \le P + 1,$$

where p is the number of processors available in the compute network, and p_k is a sequence of random integers denoting the number of correctly working processors in iteration k. For such a setting, the average reduction of the expectation of the squared error per iteration is approximately given by

$$(3.26) \quad \left(\prod_{s=0}^{k-1} \left(1 - \frac{p_s}{(P+1)\,\kappa} \right) \right)^{1/k} \approx 1 - \frac{\sum_{s=0}^{k-1} p_s}{k\,(P+1)\,\kappa} \approx 1 - \frac{r_p}{\kappa}, \qquad r_p := \mathbb{E}(p_k)/(P+1),$$

if we set $\xi = 1/\lambda_{\max}$ and take a sufficiently large k. The number r_p can be interpreted as the average rate of subproblem solves per iteration (3.22) and the linear dependence on κ is what can be expected from a linear iteration in the best case. Altogether, this convergence theory covers a stochastic version of Schwarz methods based on generic splittings, where in each iteration a random subset of subproblem solves is used. On the one hand, this theory shows that randomized Schwarz iterative methods are competitive with their deterministic counterparts. On the other hand, there are situations where randomness in the subproblem selection is naturally occurring and is not a matter of choice in the numerical method. An important example is given by algorithm-based methods for achieving fault tolerance in large-scale distributed and parallel computing applications.

This theory was already successfully applied to two different domain decomposition methods in a faulty environment, see [GO20] for further details. There, concerning the nature of faults, we assumed that faults are detectable and represent subproblem solves that were

unreturned or declared as incorrect, i.e., we ignored soft errors such as bit flips in floating point numbers even if they were detectable. Moreover we assumed that the occurrence of a fault is not related to some load imbalance, i.e., slightly longer execution or communication times for a particular subproblem solve do not increase the chance of declaring such a process as faulty. Furthermore it should not matter if faults are due to node crashes or communication failures, nor did we pose any restrictions on spatial patterns (which and how many nodes fail) or temporal correlations of faults (distribution of starting points and idle times of failing nodes). Then meaningful convergence results under such a weak fault model follow almost directly from the results above, as long as we can select, uniformly at random and independently of previous iterations, p subproblems out of the $P+1$ available ones at the start of each iteration, assign them in a one-to-one fashion to the p nodes, and send the necessary data and instructions for processing the assigned subproblem solve to each of the p nodes. Indeed, if the time available for a solve step is tuned such that there is no correlation between faults and individual subproblem solves, then one can safely assume that, with f_k denoting the number of faulty subproblem solves, the index set I_k corresponding to the $p_k = p - f_k$ subproblem solutions detected as non-faulty at the end of a cycle is still a uniformly at random chosen subset of $\{0, 1, \ldots, P\}$ that is independent of the index sets $I_0, \ldots, I_{k-1}$ used in the updates of the previous iterations. The latter independence property is the consequence of our scheme of randomly assigning subproblems to processor nodes, and not an assumption on the fault model. Thus Theorem 3.1 applies and yields the estimate

$$(3.27) \qquad \mathbb{E}(\|x - x^k\|^2) \leq \prod_{s=0}^{k-1} \left(1 - \frac{p_s}{\kappa(P+1)}\right) \|x - x^0\|^2, \qquad k = 1, 2, \ldots,$$

for the expected squared error if we formally set $\xi = \lambda_{\max}^{-1}$.

In [GO20], this approach was first applied for a simple manager-worker network example with a reliable manager node that possesses enough storage capacity to keep all necessary data, and a fixed number p of unreliable worker nodes, which perform the calculations. Here communication only takes place between the manager and worker nodes, but not between worker nodes. There, the P subproblems and also the coarse scale problem are randomly assigned to one of the available worker nodes for each iteration, the worker node receives the necessary data from the manager node for the corresponding subproblem, solves it and sends the solution back to the manager node. For simplicity p was set to $P + 1$. For example, a constant failure rate r_f then results in a constant $p^* = p_k = \lfloor(1 - r_f)(P + 1)\rfloor$ and in each iteration a fixed number $f^* = P + 1 - p^*$ of compute nodes fail to return correct subproblem solutions, which finally means that the index set I_k in the iteration was selected as a random subset of size p^* from $\{0, 1, \ldots, P\}$ and our assumptions in Theorem 3.1 are indeed fulfilled.

Subsequently, our theory was also employed to a more general local communication network with fixed assignment of the subproblems $i = 1, \ldots, P$ to P unreliable compute nodes, decentralized parallel data storage with local redundancy, and treatment of the

coarse scale problem on an additional reliable server node. Moreover communication was possible between the server node and the compute nodes but now also between geometrically neighboring compute nodes. Altogether, there were thus $p = P + 1$ nodes available for the computation. Moreover the data of the subdomain problem associated to a processor were also redundantly stored on a fixed number of neighboring processors. This allowed, in case of a fault occurring on a processor, to proceed with the associated computations on one of these neighboring processors while delaying the computations of the neighboring processor. This behavior of the iterative method then could again be matched to our convergence theory with minor modifications, compare Corollary 1 in [GO20].

The deterioration of the convergence rate with the condition number κ of the associated weighted splitting in (3.25) is typical for one-step iterations such as (3.22). Note that the convergence rate can be improved to a dependence on only $\sqrt{\kappa}$ rather than on κ by using multi-step strategies. This was indeed shown in [GO20] for a two-step variant of the basic linear iteration. Note however that there is presently no similar theory for the conjugate gradient method with a stochastic two-level Schwarz preconditioner. Moreover we are presently not aware of simple, tight estimates for the values $\lambda_{\min}$ and $\lambda_{\max}$ that are associated to our specific splitting from Subsection 2.3 of overlapping subdomains based on space-filling curves in the case of discretizations for general level parameters $l = (l_1, \ldots, l_d)$ and a coarse scale problem associated to agglomeration like (2.12). This is future work. In the following, we nevertheless apply our algorithm and report its behavior.

3.2. **Our fault-tolerant algorithm.** Now we discuss the main features of our two-level domain decomposition algorithm based on space-filling curves in a faulty environment. The main idea is again to exploit redundancy to recover data in case of a fault and to resume computation. This redundancy is now provided by means of the γ-overlap, which is present in our construction of overlapping domains Ω_i based on space-filling curves. Furthermore recall that we employ the Bank-Holst paradigm, i.e. the coarse scale problem is stored and treated redundantly on each of the P processors together with the respective subdomain problem. In the linear iteration, it interacts with the respective subproblem in an additive, parallel way, i.e. we solve the global coarse scale problem *and* the local fine subdomain problem independently of each other on one processor. The coarse scale problem is thus completely protected against faults due to this redundancy, i.e. as long as there is at least one processor running, it is always solved. Thus, for reasons of simplicity, we focus on failures in the computation of the local subproblems and just record if a processor fails within its local subproblem solver in Algorithm 2. In case of a fault of a certain processor we have to recover the necessary data of the associated subproblem from some other processor, which is a γ-dependent neighbor with respect to the one-dimensional space-filling curve ordering of the subdomains. This way, we can proceed with the iteration albeit delaying the corrections of faulty processors to a certain extent. Again, our above theory can be modified to cover this situation for the linear iteration case provided that faults are detectable, represent subproblem solves that were declared as unsuccessful or

incorrect, and are happening independently of each other. We then have the following result:

Corollary 3.2. *Let $I^1 = \{0\}$ and $I^2 = \{1, \ldots, P\}$. For fixed x^k, let x^{k+1} be given by (3.22) with $I_k = I^1 \cup I_k^2$, where the I_k^2 are uniformly at random selected subsets of I^2 of size $p_k \leq |I^2|$. Let furthermore $\xi_{k,i} := \xi_k \omega_i, i = 0, \ldots, P$. Then taking $\xi_k = p_k/(P \lambda_{\max})$ yields the estimate*

$$(3.28) \qquad \mathbb{E}(\|x - x^{k+1}\|^2 | x^k) \leq \left(1 - \frac{p_k^2}{\kappa \, P^2}\right) \|x - x^k\|^2.$$

This is a simple instance of the result in Corollary 1 in [GO20] for the local communication network adapted to our specific situation, see also the bound (32) in [GO20]. Here $\mathbb{E}(\|x - x^{k+1}\|^2 | x^k)$ denotes the expectation of $\|x - x^{k+1}\|^2$ conditioned on x^k. If no faults occur in the k-th iteration, then the contraction rate stays $1 - 1/\kappa$ of course.

If we now assume, as it is often done in the literature, that a faulty processor in iteration k comes back to life (or is replaced by a new processor) in a relatively short time and thus is active again in the next iteration, and if we assume that faults occur independently of each other, we obtain the following result for our algorithm involving the Bank-Holst paradigm:

Theorem 3.3. *Let $\xi_{k,i} := \xi_k \omega_i, i = 0, \ldots, P$ where $0 < \xi_k < 2/\lambda_{\max}$. Furthermore let $I^1 = \{0\}$ and $I^2 = \{1, \ldots, P\}$ and let $I_k = I^1 \cup I_k^2$, where in each iteration k of (3.22) the I_k^2 are uniformly at random selected subsets of I^2 of size $p_k \leq |I^2|$, i.e. let I_k^2 be selected in agreement with* **A** *and* **B**. *Then the algorithm (3.22) converges in expectation for any $x \in \mathbb{R}^N$ with*

$$(3.29) \qquad \mathbb{E}(\|x - x^k\|^2) \leq \prod_{s=0}^{k-1} \left(1 - \frac{p_s^2}{\kappa \, P^2}\right) \|x - x^0\|^2, \qquad k = 1, 2, \ldots$$

where $\kappa := \lambda_{\max}/\lambda_{\min}$ is the condition number of the underlying ω-weighted decomposition.

Now, for our setting, the average reduction of the expectation of the squared error per iteration is approximately given by

$$\left(\prod_{s=0}^{k-1} \left(1 - \frac{p_s^2}{(\kappa \, P^2)}\right)\right)^{1/k} \approx 1 - \frac{\sum_{s=0}^{k-1} p_s^2}{k \, P^2 \, \kappa} \approx 1 - \frac{\hat{r}_p}{\kappa}, \qquad \hat{r}_p := \mathbb{E}(p_k^2)/P^2,$$

if we take k to be sufficiently large. Compared to r_p in (3.26) we have lost the simple interpretation as the average rate of subproblem solves per iteration. The squaring of the value p_k/P in the expectation is due to the imbalance of fault probabilities between the coarse problem (never faulty) and fine level local problems (potentially faulty) in comparison to those in Corollary 1 of [GO20], which no longer allows to reduce the fraction and thus gives a slightly weaker upper bound.

Altogether, we now have an estimate for the conditional expectation of the squared error in one iteration with possible faults for some of the P processors and thus an asymptotic convergence theory for our specific additive, two-level domain decomposition method based

on space filling curves for the faulty situation which involves the Bank-Holst approach. This is under the assumption that we have a good upper estimate of the value $\lambda_{\max}$ of the associated Schwarz operator, i.e. the associated splitting, and that we have independent occurrences of faults in each iteration and also independent occurrences between two different iterations. Note here that this gives – as already discussed in Theorem 3.1 – not necessarily an optimal rate as for example for the non-faulty iteration method in (2.20) with optimal damping parameter, but merely an upper bound. It nevertheless shows that the asymptotic convergence rate depends on κ in a linear fashion, which is as good as we can hope for with a linear iterative method after all.

We now describe the details of the fault-tolerant version of our domain decomposition method of Algorithm 1 and 2. At the beginning, we assume to have P processors available in our compute system, where each processor will be assigned to treat one of the P subdomain problems (plus the redundant coarse scale problem) in our parallel domain decomposition method. This way, subdomain i is uniquely associated to the processor with number $proc_i$. Furthermore we have two different components in our approach, namely the failure process and the reconstruction process, which can be treated independently of each other.

In the *failure process* we decide which of the P processors fail and which stay active. Here, for reasons of simplicity, we focus on failures in the computation of the local subproblems. Note that if a fault would occur during another part of the overall algorithm, for example in the coarse scale problem or in the computation of the residual, the data can be recovered directly. This is due to the redundancy of the coarse scale problem with the Bank-Holst approach in the first case and due to the redundancy induced by the overlap and the global nature of the involved operations in the second case. This is different for the treatment of the local subproblems in our domain decomposition approach. In the following we measure the failures in terms of cycles. To this end, we define one cycle as one application of the additive Schwarz operator. For each processor we assume that the processor fails in a *random* fashion or successfully completes this cycle. Once a processor is discovered to be faulty, its local subproblem will not contribute to the iteration for that cycle.

Furthermore we assume that a faulty processor will be instantly repaired and is available again for the next iteration, i.e. a processor only stays faulty for the iteration in which the fault was detected. Indeed, that faulty processors come back to life relatively quickly or are quickly replaced is a common assumption in many algorithm-based fault-tolerant methods, see e.g. [HHLS14, LOP20, OHHBP17, PBG14]. Note here that this does in principle not exclude longer fault times of a processor since it may happen that the same processor is faulty again in the next iteration. But this happens with the same probability p_{fault} in each iteration, which results in the much smaller overall probability p_{fault}^s for a processor to be faulty for s successive iterations. This way, we ensure the independence of faults between iterations as required for Corollary 3.2 and Theorem 3.3. The more realistic scenario of a longer lasting fault is not directly treated by our theory, but, nevertheless, our assumptions

and thus our theory still give a crude worst case bound for such cases by adjusting the failure rate p_{fault} accordingly.

To simulate the failures of processors described above in our real compute system, we do the following: In each cycle we determine the faulty processors via P independently drawn Bernoulli-distributed random variables with fault probability p_{fault}, which is the same across all processors.

Once a failed processor $proc_i$ returns to the computation in the next cycle, we first employ the *reconstruction process* to restore its local data. To this end, we reassign the interval limits of processor $proc_i$. This can be done without problem since, even as $proc_i$ has failed and its data is lost, its interval information can be taken from any other running processor since we stored the index vector sfc_index and the interval limits $ta_i, tb_i, \tilde{ta}_i, \tilde{tb}_i$ redundantly on all processors. The processor $proc_i$ then locally recalculates his part of the index vector, i.e. it executes steps (4)-(6) of Algorithm 1 with the input data of processor $proc_i$. Consequently, processor $proc_i$ is aware which other processor(s) possess(es) indices which correspond to its lost local data. There exists at least one such healthy processor if $\gamma \geq 0.5$, since this ensures that at least two processors cover each index. These other processors then also possess, due to the overlapping nature of the local subproblem, the rows of A as well as the entries of all vectors (iterate, residual, etc.) that correspond to the data lost by processor $proc_i$. Here we always choose the first processor that is not $proc_i$ along the space-filling curve ordering on which the required data is available. Additionally, the redundant coarse level problem, in particular the matrices R_0, P_0 and A_0, can be fetched from any living processor since it is stored redundantly on all of them. Alternatively, it can be locally recalculated as in step (10) of Algorithm 1 to avoid communication.

Altogether, we obtain the failure and recovery process as given in Algorithms 3 and 4.

Algorithm 3 Overlapping two-level additive Schwarz using space-filling curves: Failure Cycle.

    Input: number of subproblems P, cycle number k

1: **on** all processors $proc_i$ independently **do**
2:     Determine if $proc_i$ has failed in cycle k.
3:     **if** $proc_i$ has failed in cycle k **then**
4:         Remove $proc_i$ from the computation for cycle k.
5:     **else if** $proc_i$ has not failed in cycle k but has failed in cycle $k-1$ **then**
6:         *Reconstruction* of the data of processor $proc_i$.
7:     **end if**
8: **end on**

The determination of failures in Line 2 of Algorithm 3 will simply be done by drawing independent random numbers from a Bernoulli distribution, i.e. $X \sim Bernoulli(p_{\text{fault}})$

with parameter $p_{\text{fault}} \in [0, 1]$. Moreover line 4 in Algorithm 3 in the context of our DDM Algorithm 2 essentially boils down to setting the local subproblem update d_i^k to zero if the processor corresponding to subproblem i failed in the current iteration k.

Algorithm 4 Overlapping two-level additive Schwarz using space-filling curves: Reconstruction.

Input: processor $proc_i$

1: **on** processor $proc_i$ **do**
2:     Recover interval bounds from a processor $proc_s \neq proc_i$ and
        recompute locally owned indices.
3:     **for** locally owned indices $j = ta_i, \ldots, tb_i$ **do**
4:         Find some processor $proc_t \neq proc_i$ which also owns index j.
5:         Copy necessary data (vector entries, matrix rows) corresponding to j from
            processor $proc_t$ to processor $proc_i$.
6:     **end for**
7: **end on**

Here the following remarks are in order: We employ the Bernoulli distribution which directly gives the independence of faults between two different iterations and thus fulfills the prerequisites of Corollary 3.2 and Theorem 3.3. This is in contrast to the usual assumption on the temporal distribution of faults where a Weibull distribution for the failure arrival times is used instead, see [HHLS14, ASHH16, PAS14, LOP20]. The usage of a Weibull distribution stems from the observations in [SG10]. There, in 2010, failure data had been collected over 9 years at Los Alamos National Laboratory, which include about 23.000 failures recorded on more than 20 different compute systems. The subsequent analysis of inter-arrival times of failures then gave a good agreement with the Weibull distribution, especially when the system was already in production for some time and the processors had "burnt in". Not so much is known about the length of failures of a processor (it could be Weibull-, Gamma- or even lognormally-distributed). More recent studies can be found in [YWWYZ12, JYS19]. There however, the picture is much less clear. Furthermore the data of fault distributions for current, substantially larger compute systems are not publicly known.

Note that a Weibull distribution is not memoryless and involves dependencies of the fault time points. Thus the prerequisites of our Corollary 3.2 and our Theorem 3.3 are not fulfilled for faults with inter-arrival times according to a Weibull distribution. In contrast, the use of a Bernoulli distribution for the failures adheres to the prerequisites of our Corollary 3.2 and our Theorem 3.3 and allows proven convergence rates in the faulty situation. If the true fault distribution would indeed be Weibull-distributed, our approach using a Bernoulli distribution nevertheless gives a crude upper bound on the convergence rate of a fault-tolerant method, since it can be seen as a worst case scenario (but with existing convergence theory at least for the linear iteration case, while for fault-tolerant conjugate

gradient iterations no convergence theory is presently available anyway). Moreover the time scale for the occurrence of faults is unrealistically large in most Weibull-based fault models. For more realistic fault values (like one fault a day on a big compute system) we would not see much difference at all, as the occurrence of faults during the relatively short computation time needed in our parallel domain decomposition method for any subdomain problem within the sparse grid combination method is scarce in case of an elliptic problem. The benefit of the fault-tolerant repair mechanism will rather be visible for time-dependent parabolic problems with long time horizons and long compute times, where elliptic DDM solvers for the subproblems arising in the combination method are invoked in each time step of an implicit time discretization method. But even then the number of faults during a long time simulation is rather small and there is thus not much difference between the choice of a Bernoulli or a Weibull distribution for the fault model. In any case, the Bernoulli choice gives an upper bound for the convergence and the usage of a Weibull distribution can only lead to better results in practice.

4. Numerical experiments

4.1. **Model problem.** We will consider the elliptic diffusion-type model problem

$$-\nabla \alpha(\mathbf{x}) \nabla u(\mathbf{x}) = f(\mathbf{x}) \quad \text{in } \Omega = [0, 1]^d$$

with right hand side $f(\mathbf{x})$ and appropriate boundary conditions on $\partial\Omega$. Since we are merely interested in the convergence behavior, the scaling properties and the fault tolerance quality of our approach and not so much in the solution itself, we resort to the simple Laplace problem, i.e. we set $\alpha = I$, $f = 0$, and employ zero Dirichlet boundary conditions. Consequently, the solution is zero as well. For the discretization we employ finite differences on a grid with level parameter $l = (l_1, \ldots, l_d)$, which leads to N interior grid points and thus N degrees of freedom, compare (2.8), and which results in the associated matrix A. Now any approximation x^k during an iterative process directly gives the respective error in each iteration. We measure the error in the discrete energy norm associated to the matrix A that stems from the finite difference discretization , i.e. we track

$$\|x^k\|_A := \sqrt{(x^k)^T A x^k}$$

for each iteration of the considered methods. Note here that we run the iterative algorithms for the symmetrically transformed linear system $\hat{A}\hat{x} = \hat{b}$ with $\hat{A} = T^T A T$, $\hat{b} = T^T b$, $\hat{x} = T^{-1}x$ and $T = diag(A)^{-1/2}$, whereas we measure the error in the untransformed representation, i.e. for $x - x^k$. For the initial iterate x^0 we uniformly at random select the entries $\tilde{x}_i^0, i = 1 \ldots, N$, of $\tilde{x}^0$ from $[-1, 1]$ and rescale them via $x_i^0 := \tilde{x}_i^0/\|\tilde{x}^0\|_A$ such that $\|x^0\|_A = 1$ holds. To this end, we employed the routine $uniform\_real\_distribution$ of the C++ STL (Standard Template Library). We then run our different methods until an error reduction of the initial error x^0 by at least a factor 10^{-8} is obtained and record the necessary number K of iterations. We employ two types of convergence measures: First, we consider the average convergence rate $\rho^{ave} = (\|x^K\|_A/\|x^0\|_A)^{1/K}$, which contains both a

fast initial error reduction due to smoothing effects of the employed domain decomposition iteration on the highly oscillating random initial error in the first iterations and the asymptotic linear error reduction later on. Secondly, we consider the asymptotic convergence rate ρ^{asy}. To this end, we use the maximum of the last 5 iterations and the last 5 percent of the iterations, i.e. we set $\tilde{K} := \max(5, \lceil 0.05 \cdot K \rceil)$ and define $\rho^{asy} := (\|x^K\|_A / \|x^{K-\tilde{K}}\|_A)^{1/\tilde{K}}$ which gives the average convergence rate over the last $\tilde{K}$ iterations. Note that we usually have $\rho^{ave} \leq \rho^{asy}$. The quotient ρ^{ave}/ρ^{asy} reflects the influence of the preasymptotic regime on the convergence. If the quotient is close to one, we have an almost linear decay of the iteration error starting from the very beginning, whereas, if the quotient is much smaller than one, we have a large and fast preasymptotic regime before the linear asymptotic error reduction sets in. Note furthermore that instead of ρ^{ave} we may just give the necessary number K of iterations. It bears the same amount of information as ρ^{ave}, since $K = \lceil \log(10^{-8})/\log(\rho^{ave}) \rceil$. Note finally that, instead of the random initial iterate and zero right hand side, we could have chosen zero as initial guess and a right hand side which results from the application of the discrete Laplacian to an a priorily given non-zero solution. We then would converge to the discrete solution instead of zero. This approach might be more suitable from a practical point of view, since it eliminates the randomness of the initial guess altogether and also eliminates certain smoothing effects of our algorithms in the first few iterations (as discussed in more detail later on). Therefore, it almost completely eliminates the corresponding preasymptotic regime in the convergence and directly gives the asymptotic properties of our algorithms. However, picking a sufficiently general smooth solution in an arbitrary dimension is not an easy task and it may even happen that, for a too simple solution candidate, the convergence may look deceptively good. For our choice of a random initial guess we will observe the strongest influence of the preasymptotic regime in any case.

In the following, we present the results of our numerical experiments. First, in Subsection 4.2, we study the convergence and parallel scaling behavior of both the linear two-level Schwarz/Richardson-type iteration as given in Algorithm 2 and the associated preconditioned conjugate gradient method. Then, in Subsection 4.3, we report on the behavior of the fault-tolerant version of these two iterative methods in the presence of faults.

All calculations have been performed on the parallel system *Drachenfels* of the Fraunhofer Institute for Algorithms and Scientific Computing (SCAI). It provides, among others, 1.824 Intel Sandy Bridge cores on 114 compute nodes, each one with 2 Xeon E5-2660 processors (8 cores per CPU, disabled SMT) at 2.20 GHz and 32 GB RAM, i.e. 2GB RAM per core, and 2.272 Ivy Bridge (Xeon E5-2650 v2) cores on 142 compute nodes, each one with 2 Xeon E5-2650 processors (8 cores per CPU, disabled SMT) at 2.60 GHz and 64 GB RAM, i.e. 4GB RAM per core. Drachenfels is equipped with a Mellanox Technologies MT27500 Family [ConnectX-3] 4x (56Gbps) connection. Altogether, we may use 4.096 cores on this system but we restricted ourselves to 256 cores and thus subdomains in our experiments for practical reasons.

4.2. **Convergence and parallelization results.** We are initially interested in the weak scaling situation, i.e. we set

$$N := 2^S P,$$

where S denotes the weak scaling parameter. This results in the size 2^S for each subproblem, whereas P denotes the number of subproblems. Thus the size of each subproblem stays fixed for rising P and the overall number N of unknowns grows linearly with P. Before we turn to the results of our numerical experiments, let us shortly put this weak scaling situation into the conventional geometric perspective: There, with the isotropic fine scale mesh width $h \approx N^{-1/d}$ and the isotropic coarse grid mesh size $H \approx (q \cdot P)^{-1/d}$, we get

$$(4.30) \qquad \frac{H}{h} \approx \left(\frac{q \cdot P}{N} \right)^{-1/d} = \left(q \cdot 2^{-S} \right)^{-1/d},$$

which is independent of P. From conventional domain decomposition theory we have (for geometric coarse grid problems) a condition number of the order $O(1 + H/h)$, compare (2.1) with $\delta = ch$. Now, with (4.30), we obtain a condition number and a convergence rate which is independent of N and P, i.e. we achieve weak scaling of the respective domain decomposition method. Moreover (4.30) suggests the choice

$$(4.31) \qquad\qquad\qquad q := c \cdot 2^S,$$

which then gives $H/h \approx c^{-1/d}$ independent of N, P and S. We are interested in also observing such a weak scaling behavior for our non-geometric overlapping domain decomposition method, which is based on the space-filling curve approach and which involves a coarse scale problem set up purely algebraically via agglomeration.

We study the convergence behavior of the two-level additive Schwarz/Richardson-type iteration from Algorithm 2 with optimal damping parameter $\xi^* = 2/(\lambda_{min} + \lambda_{max})$ as a solver. To this end, we numerically determine the two eigenvalues by the Krylov-Schur method beforehand. Note that the eigenvalues correspond to their specific subspace splitting, i.e. to the specific two-level domain decomposition operator under consideration. The splitting depends on the dimension d, on the discretization level $l = (l_1, \ldots, l_d)$, on the number P of subdomains, on the weak scaling parameter S, on the coarse level parameter q, on the overlap parameter γ, on the respective scalings D_i, on the specific space-filling curve, and finally on the specific type of Schwarz operator (plain additive, balanced).

First, we consider the one-dimensional case in our numerical experiments. Surely there is no need to employ a parallel iterative domain decomposition solver and a direct sequential Gaussian elimination would be sufficient. However, this is a good starting point to study the convergence and parallel scaling properties of the various algorithmic variants. Moreover it will turn out that the one-dimensional case is indeed the most difficult one for good convergence and scaling results. This behavior stems from the relative "distance" of the fine scale to the coarse scale, which is maximal for $d = 1$. Here we study the convergence and scaleup behavior of the standard additive two-level Schwarz/Richardson-type approach (2.15) as well as its balanced variant (2.7). Furthermore we consider the conjugate gradient

method with these two associated variants of the respective two-level domain decomposition operators as preconditioner. We set $N := 2^S P$ where $S := 8$. Thus the size of each subproblem stays fixed at 2^8 with growing P, whereas the overall number N of unknowns grows linearly with P. Moreover we fix $q = 16$ and $\gamma = 0.5$. We compare the different methods for the three scalings $D_i = I$ (no scaling at all), $D_i = \omega_i I$ with ω_i according to (2.18), and D_i according to (2.17), $i = 1, \ldots, P$. For our special choice of γ we know from Lemma 2.1 that the weighting with ω_i and the weighting with D_i are indeed the same and differ from the unweighted case by just the constant scaling $\frac{1}{c} I$ with $c = 2\gamma + 1$. For the coarse scale problem, we always set $D_0 = I$. The results for the two types of convergence rates of the standard additive two-level Schwarz/Richardson-type approach (2.15) and the associated conjugate gradient method are shown in Figure 8. The results for the two types of convergence rates of the balanced variants according to (2.7) are given in Figure 9.
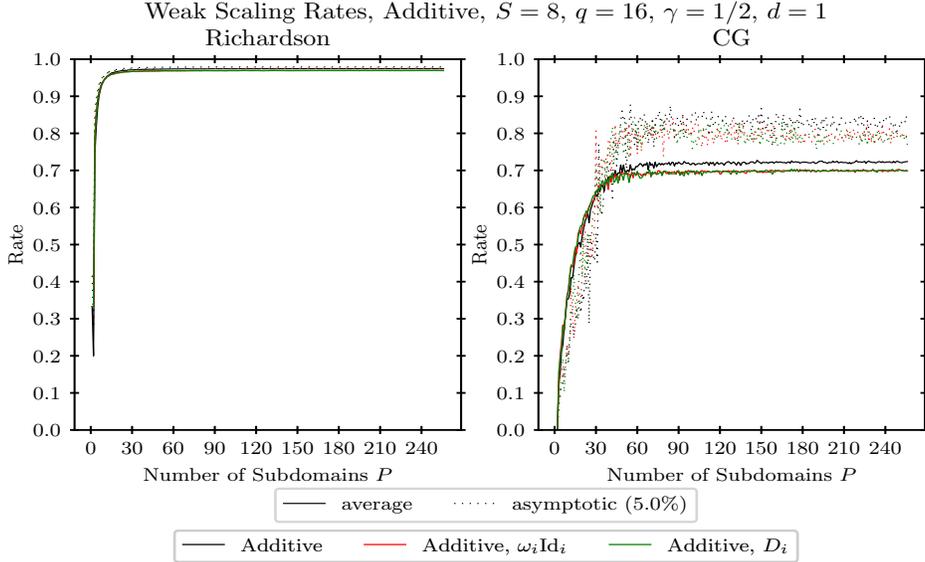
FIGURE 8. Weak scaling: Convergence rates ρ^{ave} and ρ^{asy} versus number of subdomains for the three weighted versions of the *plain* two-level additive Schwarz/Richardson iteration (left) and the associated preconditioned conjugate gradient method (right), $d = 1$, $q = 16$, $\gamma = 0.5$.

We see that the scaling with ω_i and the scaling with D_i indeed give the same results, as was expected. Moreover the case of no scaling at all only gives slightly worse results for the plain additive variant and even gives the same results for the balanced variant for this one-dimensional situation. Furthermore we observe that ρ^{ave} and ρ^{asy} are approximately the same in the Richardson case, i.e. there is not much difference to be seen. This is surely due to the large number of iterations needed to reach an error reduction of 10^{-8} and a moderate preasymptotic regime. The rates are quite high with a value of about 0.98 for larger values of P. Moreover, in the conjugate gradient case, we see a slight difference between ρ^{ave} and ρ^{asy} of about 0.08, which is due to the preasymptotic regime incorporated in ρ^{ave} and the lower number of iterations involved. For the balanced variants, there is still not much
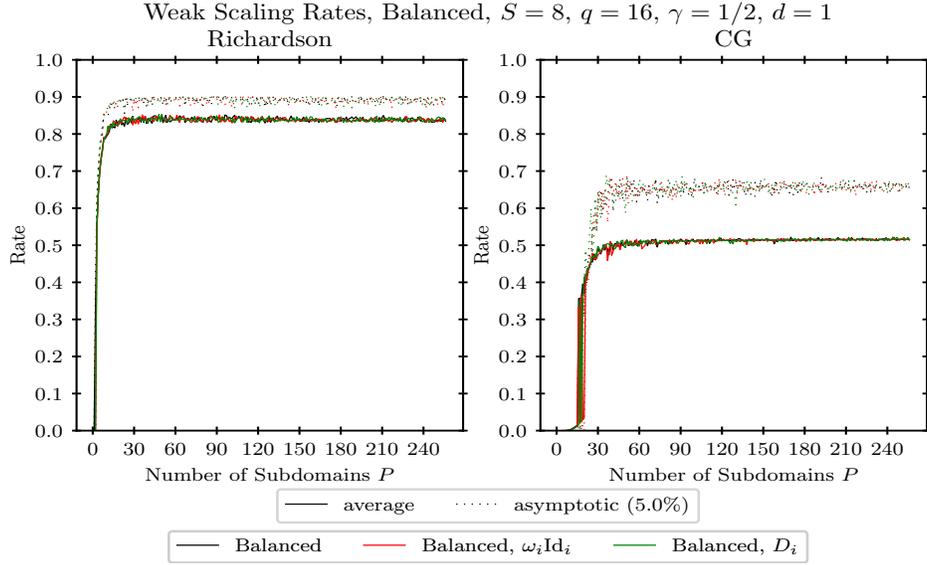
FIGURE 9. Weak scaling: Convergence rates ρ^{ave} and ρ^{asy} versus number of subdomains for the three weighted versions of the *balanced* two-level additive Schwarz/Richardson iteration (left) and the associated preconditioned conjugate gradient method (right), $d = 1$, $q = 16$, $\gamma = 0.5$.

difference for the Richardson iteration, i.e. there is just a difference of about 0.04 between the ω_i-scaling and no-scaling at all, but the rates are a improved with values of about 0.9. In the conjugate gradient case, we now observe an improvement to a rate of 0.6 for ρ^{ave} due to balancing. Thus balancing indeed improves convergence substantially. Moreover we obtain a more distinct difference between ρ^{ave} and ρ^{asy} with a value of 0.13, which is again due to the preasymptotic regime incorporated in ρ^{ave} and the lower number of iterations involved. Nevertheless, for rising values of P, constant convergence rates are obtained in any case, which indicates weak scaling.

We conclude that the average convergence rate ρ^{ave} is a sufficient measure to unveil the weak scaling properties of our various algorithms, even though the asymptotic convergence rate ρ^{asy} is a bit larger. Since ρ^{ave} is related via $K = \lceil \log(10^{-8}) / \log(\rho^{ave}) \rceil$ to the number K of required iterations, we will report the number K of iterations instead of ρ^{ave} for the remainder of this subsection. As we are interested in weak scaling properties and since a fixed reduction of the initial error is considered anyway, the necessary number K of iterations is sufficient for our purpose and we may neglect the influence of the preasymptotic regime. This will be different in the next Subsection 4.3, where we will consider the behavior of our algorithms for different fault probabilities. Then the size of the preasymptotic regime can be influenced by faulty subproblem solves and plays a certain role, as we will see later on.

Thus, from now on in this subsection, the number K of iterations necessary to reduce the initial error by a factor of 10^{-8} is shown. For the plain additive two-level Richardson-type

approach (2.15) and the associated conjugate gradient method, they are given in Figure 10. For the balanced variants according to (2.7), they are given in Figure 11.
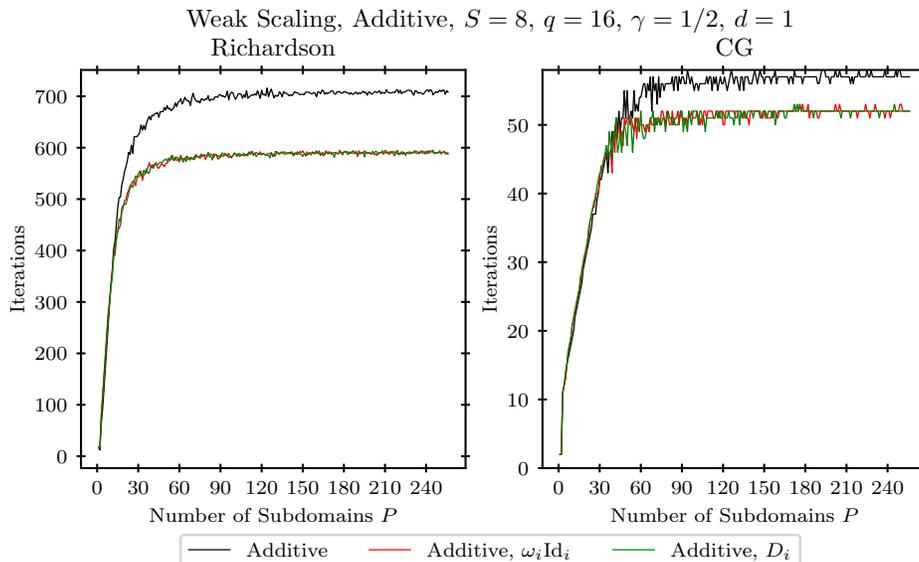
FIGURE 10. Weak scaling: Number of iterations K versus number of subdomains for the three differently weighted versions of the *plain* two-level additive Schwarz/Richardson iteration (left) and the associated preconditioned conjugate gradient method (right), $d = 1$, $q = 16$, $\gamma = 0.5$.
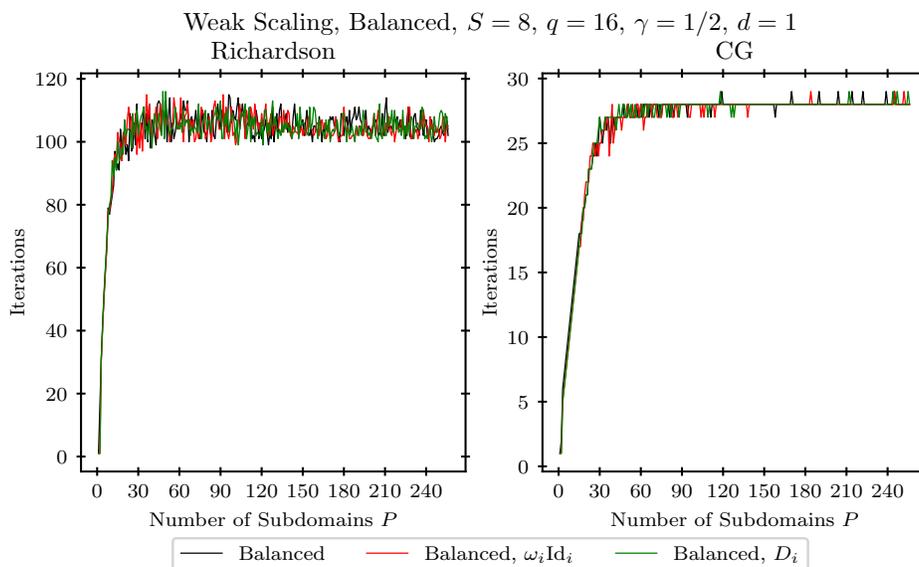
FIGURE 11. Weak scaling: Number of iterations K versus number of subdomains for the three differently weighted versions of the *balanced* two-level additive Schwarz/Richardson iteration (left) and the associated preconditioned conjugate gradient method (right), $d = 1$, $q = 16$, $\gamma = 0.5$.

Of course we observe the same behavior as in the two previous figures for ρ^{ave}, albeit now in a different representation which is due to the measured number of iterations. Moreover we again obtain weak scaling behavior in all cases, i.e. the necessary number of iterations stays constant for growing values of P. This constant depends on the respective splitting: For the plain additive Schwarz/Richardson iteration (left) we see that a scaling with ω_i (and equally with D_i) reduces this constant compared to the no-scaling case, albeit a large number of iterations is still needed. Moreover the results get substantially improved by the balanced variants: All three scalings now give the same results and the weak scaleup constant is reduced by a factor of approximately seven for the unweighted case and by a factor of approximately six for the other two variants. For the associated preconditioned conjugate gradient methods (right) we observe a further reduction of the necessary number of iterations. This reflects roughly the κ-versus-$\sqrt{\kappa}$ effect of the conjugate gradient method in its convergence rate. For the balanced version, we additionally see a substantial improvement of the scaleup constant compared to the plain additive preconditioned conjugate gradient method by a factor of nearly one half. Note here that for values of γ that are not integer multiples of $1/2$, the D_i-scaling does not lead to a symmetric operator, which renders a sound and robust convergence theoretically questionable and in further experiments gave considerably worse iteration numbers with oscillating behavior for the corresponding preconditioned conjugate gradient method in practice.

We conclude that the balanced variant of both the Schwarz/Richardson iteration and the associated preconditioned conjugate gradient method is substantially faster than the plain version. We also see that, for our choice $\gamma = 1/2$, balancing eliminates the difference of the unscaled and the ω_i-scaled (and D_i-scaled) cases. Moreover the preconditioned conjugate gradient version is nearly quadratically faster and gives good weak scaling constants. In further comparisons with the Nicolaides method (2.6) it was found that the balanced approach was again superior. Also, a comparison with the deflated variant (2.6) revealed that the balanced method was more robust. Therefore, we will from now on focus on the optimally damped, *balanced* Schwarz iteration and the associated preconditioned conjugate gradient method. The type of damping we will choose, i.e. none at all, ω_i according to (2.18) or D_i according to (2.17) is still to be determined. We refrain from employing the D_i-weighting for general choices of γ, since in general this results in a non-symmetric operator for which our theory is not valid anymore. It now remains to study the behavior of the unweighted and the ω_i-weighted algorithms in more detail.

So far, we kept the value of the overlap parameter γ fixed. Now we vary γ and consider its influence on the weak scaling behavior of our two algorithms in the balanced case. First we consider the one-dimensional situation, where we set $S = 8$, $q = 16$ and vary the number P of subdomains. The resulting number of iterations for different values of γ ranging from $1/5$ up to 5 are shown in Figure 12 for the unweighted case and in Figure 13 for the ω_i-weighted case.
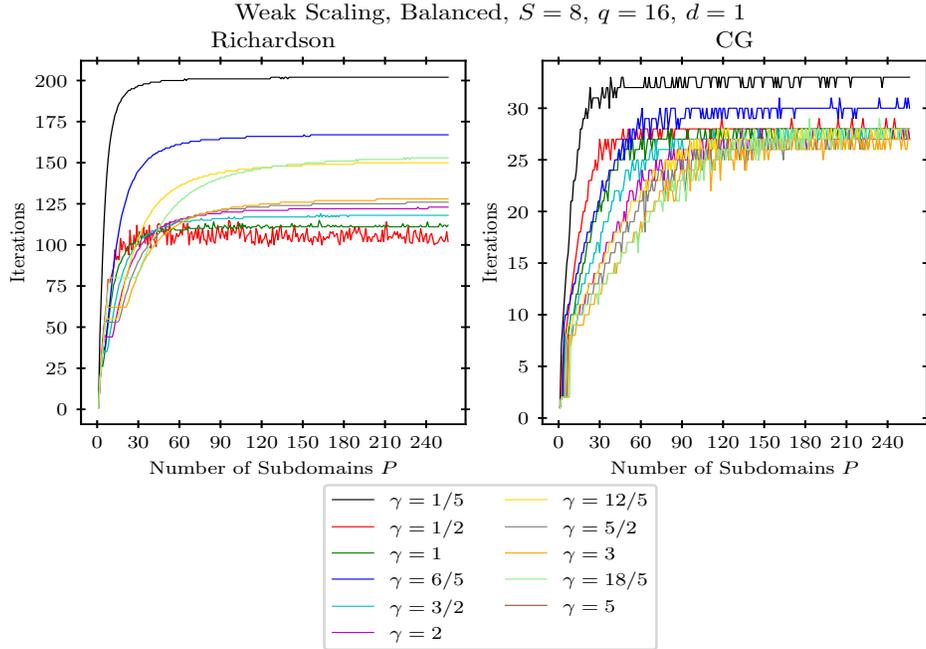
FIGURE 12. Weak scaling: Number of iterations versus number of subdomains for the unweighted balanced two-level additive Schwarz/Richardson iteration (left) and the associated preconditioned conjugate gradient method (right) for different values of γ with varying P, $d = 1$, $q = 16$, $S = 8$.

Comparing the ω_i-weighted case to the unweighted case, there is not much visible difference at all. We again clearly see weak scaling behavior. The scaling constant now depends on the respective value of γ. In the Richardson case, it is interesting to observe that, in any case, a constant number of iterations is quickly reached for rising values of P. Moreover, for a small overlap value of $\gamma = 1/5$, it is quite bad. The number of iterations is seen to be optimal for $\gamma = 1/2$ and then deteriorates for larger values of γ. Note at this point that, starting with $\gamma = 1/2$, we observe a slight deterioration of the convergence curves and of the weak scaling constant, where this deterioration is *monotone* in n if we restrict ourselves to values of γ that are integer multiples of 0.5, i.e. $\gamma = \frac{1}{2}n, n \in \mathbb{N}_+$. The non-integer multiples give worse results. In the conjugate gradient case, the scaling constant is reached increasingly later for rising values of P (which is desirable). It is improved in a nearly monotonic way for rising values of γ. Furthermore the absolute value of necessary iterations is again much smaller than in the Richardson case.

Next, we consider the three-dimensional case and again vary the value of γ. Then, in contrast to the one-dimensional situation, the Hilbert curve structure comes into play. The resulting iteration numbers for different values of γ in the unweighted and the ω_i-weighted case are shown in Figures 14 and 15, respectively.

For the Richardson iteration we again see an analogous behavior for the weak scaling constant but with a much lower number of iterations compared to the one-dimensional
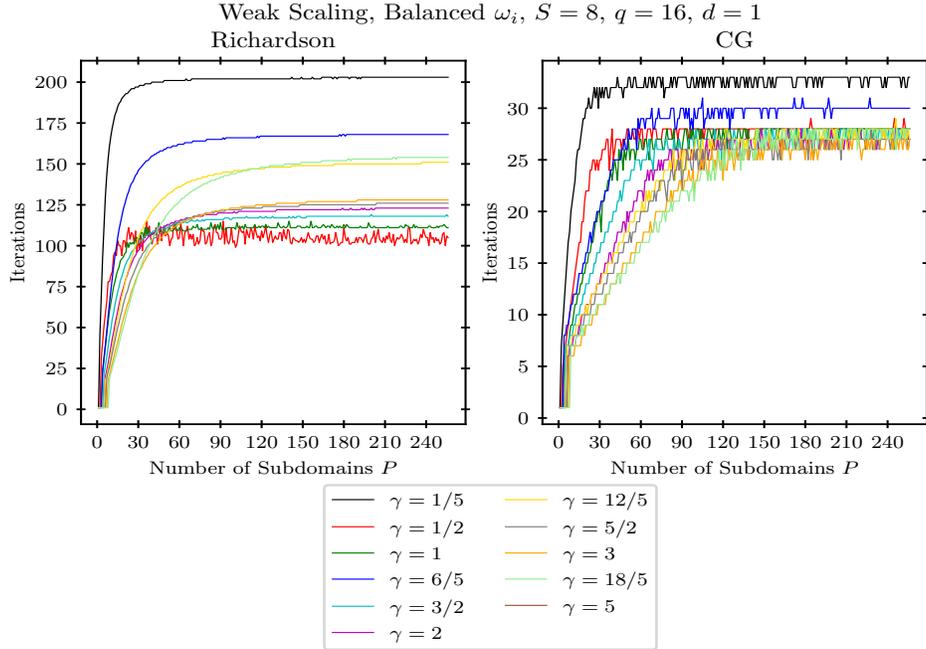
FIGURE 13. Weak scaling: Number of iterations versus number of subdomains for the ω_i-weighted balanced two-level additive Schwarz/Richardson iteration (left) and the associated preconditioned conjugate gradient method (right) for different values of γ with varying P, $d = 1$, $q = 16$, $S = 8$.

case. Furthermore we observe that the ω_i-scaling improves the convergence: Except for $\gamma = 1/5$, all required numbers of iterations are now close together for both the values of γ that are integer multiples of $1/2$ and the other values, and their respective number of iterations are in any case reduced to around 50. This shows that ω_i-weighting is able to deal with the decomposition based on the Hilbert curve, which appears for $d > 1$, in a proper way. In the unweighted conjugate gradient case, the curves are approximately the same for all values of γ, except for $\gamma = 1/5$ which is too small again. They are successively improved for rising values of γ in the ω_i-weighted case, as is expected intuitively. Again, the conjugate gradient method is much faster than the Richardson scheme. Similar observations could be made for other dimensions.

Altogether, ω_i-weighting stabilizes the iteration numbers against variations of γ and improves the convergence behavior. In light of the larger costs involved for higher values of γ, a good choice for γ is given by the value $1/2$ in both the Richardson and the conjugate gradient case. However, larger values of γ may be needed to attain fault tolerance due to redundancy later on. The ω_i-weighted case then indeed results in slightly better convergence results for larger values of γ and shows no deterioration, which the unweighted case does. From now on we therefore will focus on the optimally damped, ω_i-weighted balanced Schwarz iteration and the associated preconditioned conjugate gradient method. Moreover we will restrict ourselves in this subsection to the value $\gamma = 1/2$. In the next subsection,
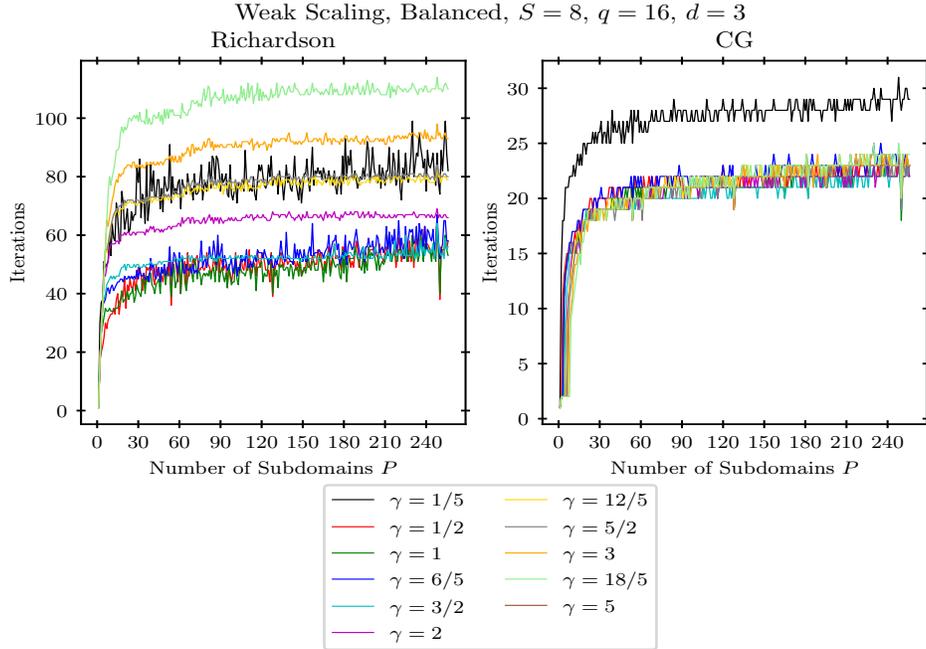
FIGURE 14. Weak scaling: Number of iterations versus number of subdomains for the unweighted balanced two-level additive Schwarz/Richardson iteration (left) and the associated preconditioned conjugate gradient method (right) for different values of γ with varying P, $d = 3$, $q = 16$, $S = 8$.

where we deal with fault tolerance, we will also employ values of γ that are larger integer multiples of $1/2$.

So far, we kept the weak scaling parameter S, i.e. the size 2^S of each subproblem, fixed and only varied the number P of subdomains. But what happens if we also vary the subproblem size? For fixed $q = 16$, the results are shown in Figure 16.

We again clearly see weak scaling behavior. However the weak scaling constant now depends on the subproblem size, i.e. it grows with rising S. This holds for both the Richardson-type iteration and the conjugate gradient approach. This behavior stems from the fixed value of q and thus the fixed size of the coarse scale problem for fixed P. In this case the difference between fine scale and coarse scale increases with growing S and, consequently, the additive coarse scale correction is weakened compared to the fine scale.

We now let the coarse scale parameter q be dependent on S, compare also the discussion leading to (4.31). To be precise, we set $q = 2^{S-4}$, which gives a coarse problem of size $P \cdot 2^{S-4}$, where we vary the parameter S. This way, we double q while doubling the subdomain size 2^S, i.e. N/P. The obtained results are shown in Figure 17.

In all cases, we obtain substantially improved results compared to the fixed choice $q = 16$ from Figure 16. This was to be expected since now the coarse scale correction is improved for rising S. We again observe an asymptotically constant number of iterations for growing values of P and we obtain a weak scaling constant which is, compared to the fixed
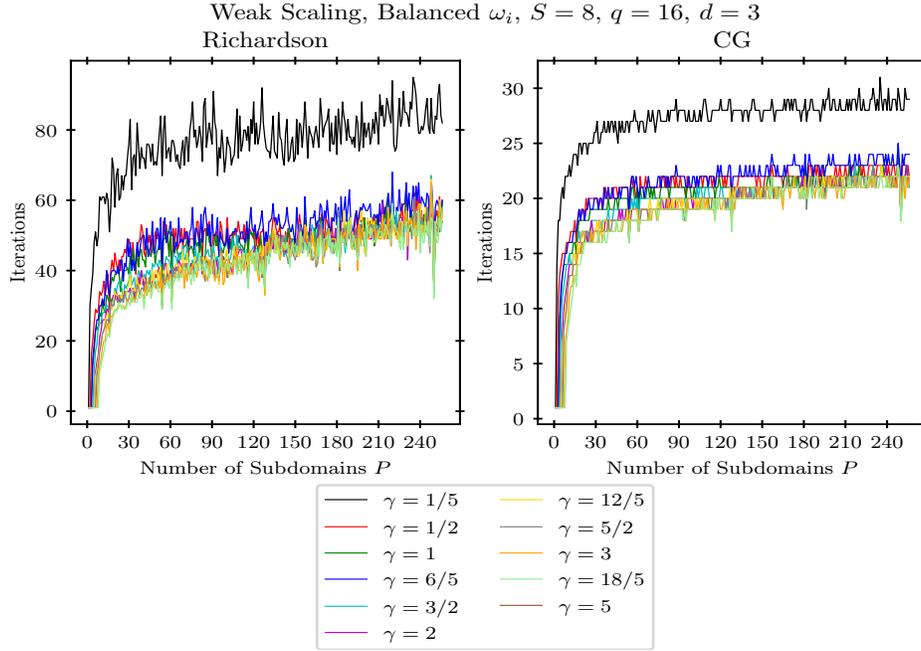
FIGURE 15. Weak scaling: Number of iterations versus number of subdomains for the ω_i-weighted balanced two-level additive Schwarz/Richardson iteration (left) and the associated preconditioned conjugate gradient method (right) for different values of γ with varying P, $d = 3$, $q = 16$, $S = 8$.
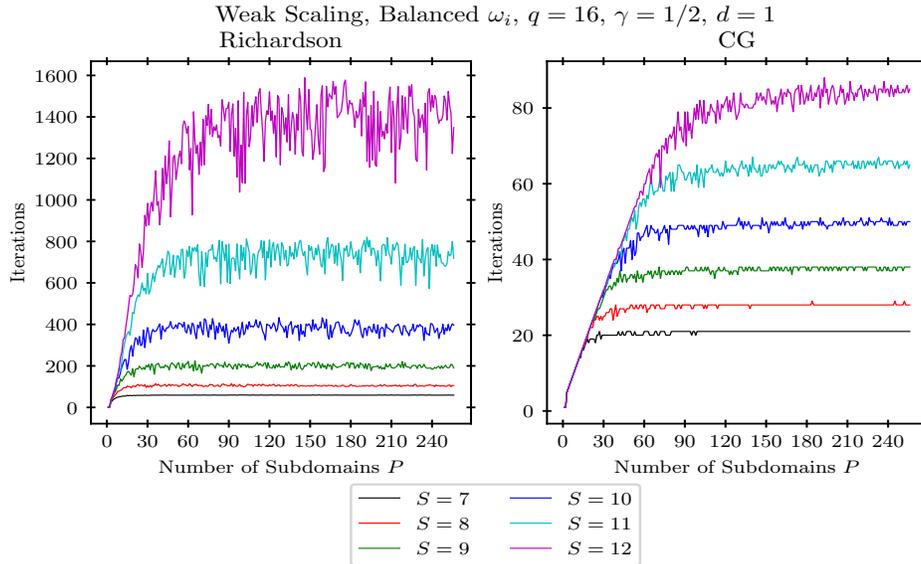
FIGURE 16. Weak scaling: Number of iterations versus number of subdomains for the ω_i-weighted balanced two-level additive Schwarz/Richardson iteration (left) and the associated preconditioned conjugate gradient method (right) for different values of S, $d = 1$, $q = 16$, $\gamma = 0.5$.
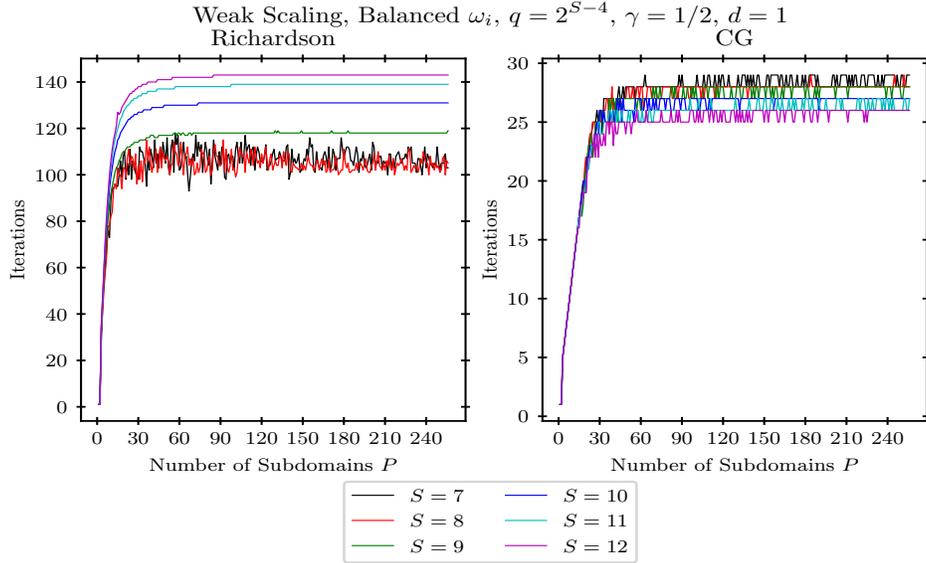
FIGURE 17. Weak scaling: Number of iterations versus number of subdomains for the ω_i-weighted balanced two-level additive Schwarz/Richardson iteration (left) and the associated preconditioned conjugate gradient method (right) for different values S, $d = 1$, $q = 2^{S-4}$, $\gamma = 0.5$.

choice $q = 16$, now only slightly growing with S for the Richardson iteration and seems to approach a limit of about 145 for rising values of S. Moreover it is now completely independent of S for the conjugate gradient approach, for which we need at most 29 iterations in all cases. This shows that q should scale proportional to N/P. In further experiments we quadrupled q while doubling N/P and found that the weak scaling constant then even shrank with growing S.

Note at this point that the size of the coarse scale problem is now $2^{-4} \cdot P \cdot 2^S$, while the size of each subdomain problem is 2^S (whereas the size of the overall problem is $N = P \cdot 2^S$). Thus the cost of solving the coarse scale problem tends to dominate the overall cost with rising P, which is the price to pay for a more uniform convergence behavior. This calls for further parallelization of the coarse scale problem itself via our P processors to remedy this issue. Thus, in contrast to the present implementation via the Bank-Holst paradigm where we redundantly keep the coarse grid problem on each processor (besides the associated subdomain problem), we should partition the coarse scale matrix to the P processors and therefore solve the coarse scale problem in a parallel way. This however will be future work.

Next, we consider the weak scaling behavior for varying dimensions $d = 1, 2, 3, 4, 5, 6$. For the discretization we stick to the isotropic situation, i.e. we set

$$l = (\lfloor (S + \log(P))/d \rfloor, \ldots, \lfloor (S + \log(P))/d \rfloor).$$

Thus the overall number of degrees of freedom is independent of d, since

$$N \approx \prod_{j=1}^{d} 2^{(S+\log(P))/d} = 2^{S+\log(P)} = P \cdot 2^S,$$

and the size of each subdomain is again approximately 2^S for all values of P. Furthermore we choose $\gamma = 0.5$, $S = 8$ and $q = 2^{S-4}$ and consider only the ω_i-weighted balanced methods. The resulting weak scaling behavior is shown in Figure 18.
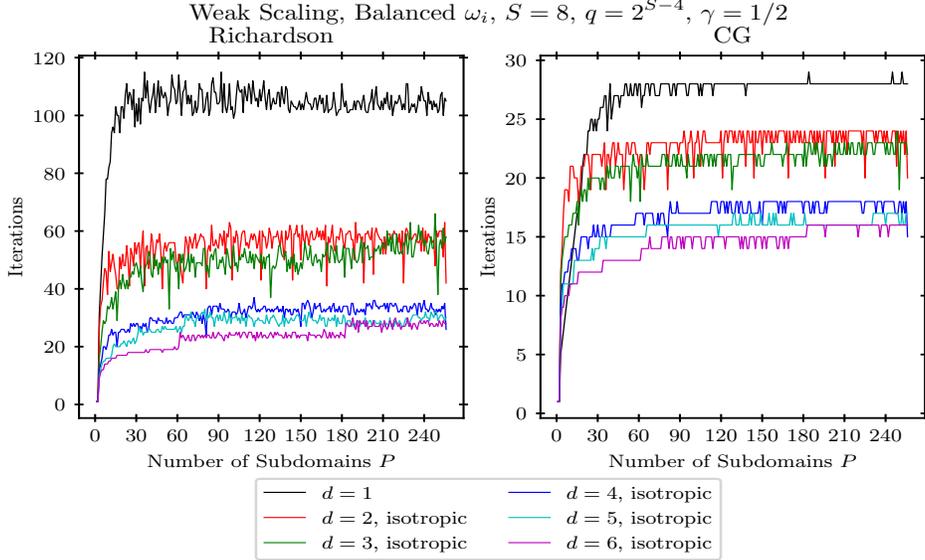
FIGURE 18. Weak scaling: Number of iterations versus number of subdomains for the ω_i-weighted balanced two-level additive Schwarz/Richardson iteration (left) and the associated preconditioned conjugate gradient method (right) for different dimensions d, $q = 2^{S-4}$, $\gamma = 0.5$.

We always obtain weak scaling behavior, where now the constant depends on d. But it improves for growing values of d and we see that the one-dimensional case is indeed the most difficult one. This is due to the relative "distance" of the fine scale to the coarse scale in the two-level domain decomposition method, which is largest for $d = 1$ and decreases for larger values of d. Furthermore a stable limit of 26 and 16 iterations, respectively, is reached for $d \geq 6$. Such a behavior could be observed not only for the isotropic discretizations in d dimensions but also for all the various anisotropic discretizations, which arise from (2.9) in the sparse grid combination method. This becomes clear when we consider the simple case of the anisotropic discretization $l = (\lfloor S + \log(P) \rfloor, 1, \ldots, 1)$ in d dimensions: With our homogeneous Dirichlet boundary conditions we obtain the same fine scale finite difference discretization as for the case $d = 1$ with differential operator $-\partial^2/\partial_{x_1}^2 + 8(d-1) \cdot I_{x_1}$. We then have an additional reaction term of size $8(d-1)$, which merely improves the condition number compared to the purely one-dimensional Laplacian. Consequently, the one-dimensional convergence results impose an upper limit to the number of iterations needed for all the subproblems arising in the combination method.

Now let us shortly consider the strong scaling situation as well. There we have $N = 2^L$, where the size of each subdomain is $2^L/P$, i.e. it decreases with growing values of P. Moreover values of P larger than 2^L are not feasible. We consider again the one-dimensional situation, set $q = 2^{L-12}$, $\gamma = 0.5$, and vary the number P of subdomains. The resulting strong scaling behavior is shown for $L = 16, 18, 20$ in Figure 19.
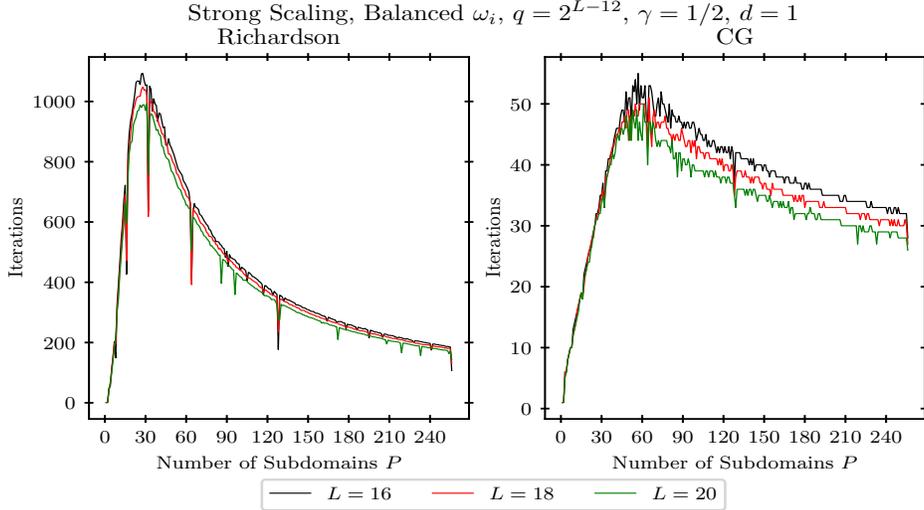
Figure 19. Strong scaling: Number of iterations versus number of subdomains for the ω_i-weighted balanced two-level additive Schwarz/Richardson iteration (left) and the associated preconditioned conjugate gradient method (right), $d = 1$, $L = 16, 18, 20$, $q = 2^{L-12}$, $\gamma = 0.5$.

We see that the necessary number of iterations first grows with rising values of P and then, after its peak, steadily declines, as is expected. This is due to the fact that now the size of each subproblem shrinks with rising P, whereas the size $P \cdot 2^{L-12}$ of the coarse scale problem grows linearly with P. There is not much of a difference for the three curves $L = 16$, $L = 18$ and $L = 20$ since for each P the coarse scale problem has the same relative distance to the fine grid for all values of L, i.e. $2^{16}/(2^{16-12} \cdot P) = 2^{18}/(2^{18-12} \cdot P) = 2^{20}/(2^{20-12} \cdot P)$. Note here that the downward peaks of the curves in Figure 19 correspond to the situation where all subdomains are perfectly balanced, i.e. where $N \mod P = 0$. Furthermore the parameter q was chosen such that the case $P = 256$ for $L = 16, 18, 20$ in Figure 19 results in exactly the same situation as the case $P = 256$ for $S = 8, 10, 12$ in Figure 17, respectively.

We finally consider strong scaling for the situation where $q = 2^{\lfloor \log(N/P) \rfloor - 4}$, i.e. where the choice of q is P-dependent and is chosen such that the relative distance between the coarse and the fine grid is roughly four levels for all P, as was the case in the weak scaling experiments in Figure 17. The results are given in Figure 20. Due to the involved round off there are now jumps at the points where $\log_2(P)$ is an integer. However we observe the expected steady decline of the necessary number of iterations in between these jump points. Note that, for this choice of q, the necessary numbers of iterations for the end points of these intervals are approximately the same.
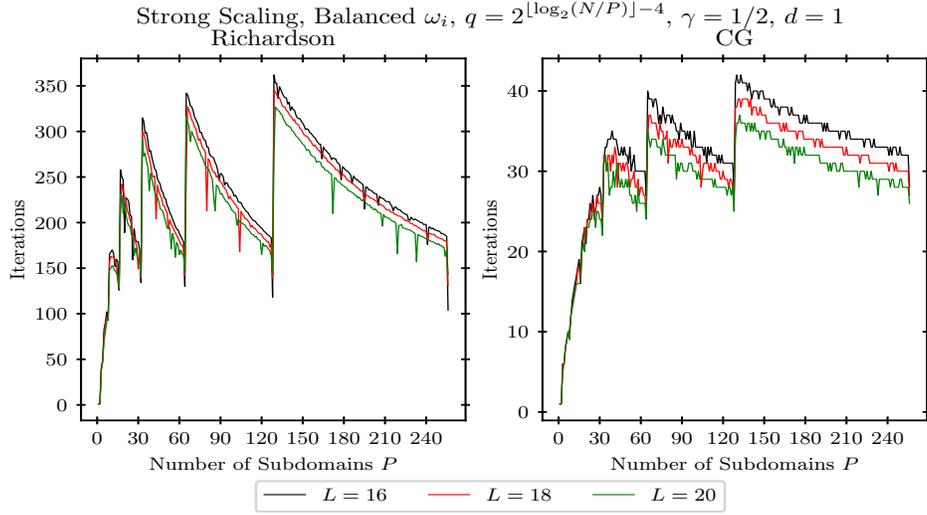
FIGURE 20. Strong scaling: Number of iterations versus number of subdomains for the ω_i-weighted balanced two-level additive Schwarz (left) and the associated preconditioned conjugate gradient (right) method, $d = 1$, $L = 16, 18, 20$, $q = 2^{\lfloor \log(N/P) \rfloor - 4}$, $\gamma = 0.5$.

4.3. **Fault tolerance results.** We now consider the results of our fault tolerance experiments. As already mentioned, we assume fault locality (in space and time) which is often done in the literature for analysis purposes. As a model for faults we employ the Bernoulli distribution to inject faults into our compute system, as described in detail in Subsection 3.2. This directly gives independent occurrences of faults per cycle. Furthermore we assume that a faulty processor will be instantly repaired and is available again in the next iteration, i.e. a processor only stays faulty for the iteration in which the fault was detected. We exploit a sufficiently large γ of overlap and thus *redundancy* on the stored data in our algorithm to be able to recover lost data due to a faulty processor. This assumption on the model adheres to the prerequisites of our Corollary 3.2 and our Theorem 3.3 and thus enables an estimate for the expectation of the squared error for our specific additive, two-level domain decomposition method based on space filling curves in the faulty situation. Now we consider the ω_i-weighted balanced two-level additive Schwarz/Richardson iteration with damping parameter ξ^* under faults. Moreover we are interested in the behavior of the associated preconditioned conjugate gradient methods under faults, even though we do not have a proper convergence theory for this case as for the simpler linear iteration. Note to this end that the energy norm based on the finite element matrix, for which our theory is valid, and the energy norm based on the finite difference matrix, which we use in practice, are, up to a scaling of $h^{-d/2} = 2^{(dL)/2}$, spectrally equivalent due to the well-known L_2-stability of the piecewise d-linear finite element basis.

First we study the convergence behavior of these different methods. To this end, we consider the case $d = 1$, we set $q = 16$, $S = 8$ and $P = 100$. Thus the size N of the overall problem is fixed to $100 \cdot 2^8$ with a size of 2^8 for each subproblem. Now we study the error reduction

properties of our fault-tolerant algorithms. To this end we consider, besides the case of no fault at all, the cases where the faults are randomly generated in each cycle and for each processor according to the Bernoulli distribution with different fault-probabilities $p_{\text{fault}} = 0.01, 0.02, 0.05, 0.1$ and 0.2. This way, on average, one, two, five, ten and twenty processors fail in each iteration. Since the faults are randomly distributed, we run the algorithms ten times and show the resulting convergence curves for each run in transparent colors. We additionally show the averaged convergence curve in bold color. Note here also that, if the overlap parameter γ is too small compared to the fault rate p_{fault}, a proper fault recovery might no longer be possible and, depending on the specific realization of random picks of faulty processors, the respective run of the algorithm may no longer be able to recover and may fail. These failed runs are discarded in the computation of the average convergence rate. For the overlap parameter $\gamma = 0.5, 1, 1.5, 2$, the results for the ω_i-weighted balanced two-level Richardson/Schwarz iteration and the correspondingly preconditioned conjugate gradient method are shown in Figure 21.

We see that in the ω_i-weighted balanced Richardson case there is a slight preasymptotic reduction of the initial error in the first iterations, which shortens but is more profound the larger the values of p_{fault} and of γ are. After that an asymptotic error reduction can be observed which is nearly the same as for the non-faulty case. A close inspection reveals that, depending on p_{fault}, this asymptotic rate becomes slightly larger in the end compared to the non-faulty case. This is however to be expected from our theory, compare also the bound for the (asymptotic) rate in Theorem 3.3. The preasymptotic behavior stems from the fact that a moderate number of faults indeed restores a certain smoothing effect in the preasymptotic regime. This observed (small) preasymptotic smoothing effect in the faulty cases can best be explained by looking at all the fine grid points that belong to a failing processor. For example, the overlap $\gamma = 2$ and the associated weighting $\omega_i I = \frac{1}{2\gamma+1} I = \frac{1}{5} I$ yields an (asymptotically) optimal damping parameter of $\xi_{\text{opt}} = 2/(\lambda_{\min} + \lambda_{\max}) \approx 1.86$ for the balanced preconditioner. Recall that ω_0 is chosen as one here. Then the effective preconditioner, including damping, can be seen to be

$$
\begin{aligned}
\xi_{\text{opt}} C_{(2),D,bal}^{-1} &= \xi_{\text{opt}}(G^T C_{(1),D}^{-1} G + F) = \xi_{\text{opt}}(\frac{1}{5} G^T C_{(1)}^{-1} G + F) \\
&= \xi_{\text{opt}} \frac{1}{5} G^T C_{(1)}^{-1} G + \xi_{\text{opt}} F \approx 0.37\, G^T C_{(1)}^{-1} G + 1.86\, F.
\end{aligned}
\tag{4.32}
$$

On the other hand, an intuitive way of weighting the respective fine grid corrections in the iteration would be to average just over all fine grid contributions per point, which resembles a certain smoothing of higher frequency error components. For our example, this would yield a weighting factor of $\frac{1}{5}$, i.e. the associated preconditioner would then be $\frac{1}{5} G^T C_{(1)}^{-1} G + F$. A comparison of this preconditioner to the above effective preconditioner (4.32), which involves the asymptotically optimal dampening parameter ξ_{opt}, shows that the optimally damped version shifts the balance between fine and coarse level more towards the coarse side to achieve optimal convergence in the asymptotic regime. But this in turn yields worse smoothing properties in the preasymptotic regime compared to the dampening

Fault Tolerance, Balanced ω_i, $S = 8$, $P = 100$, $q = 16$, $d = 1$
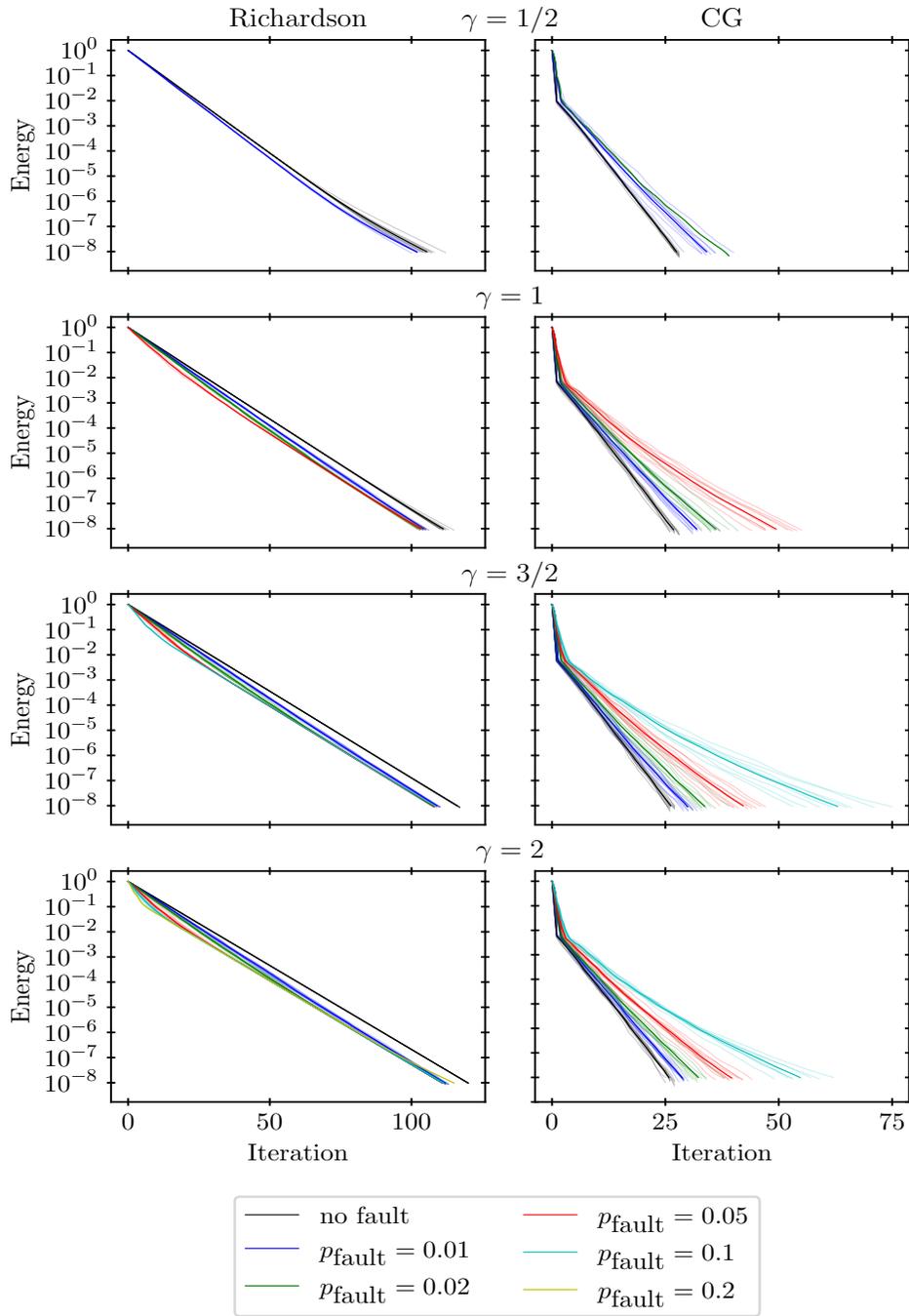
FIGURE 21. Number of iterations (single runs and mean) of the fault-tolerant version of the ω_i-weighted balanced two-level additive Schwarz/Richardson iteration (left) and the associated preconditioned conjugate gradient method (right) for different fault probabilities and different overlaps $\gamma = 0.5, 1, 1.5, 2, q = 16, d = 1$.

due to averaging, since the factor 0.37 of the fine grid contribution does not match the fact that each point is overlapped by five fine subdomains. However, if we now consider

the faulty cases, the occurrence of a few faults brings the expected factor due to averaging closer to this factor 0.37. For example, if one of five neighboring processors fails, one would expect an averaging factor of $\frac{1}{4}$, since now only $4 = 5 - 1$ processors contribute to the correction at all points belonging to the failing processor. In the case of two neighboring faults the expected weighting factor due to averaging of $\frac{1}{3}$ in the points belonging to both failing processors is now even closer to our asymptotically optimal fine grid weighting of 0.37. Therefore we see an improved initial smoothing behavior for a moderate number of neighboring faults compared to the non-faulty case simply due to the fact that the effective weighting parameter for the fine grid, i.e. $\xi_{\mathrm{opt}} \cdot \omega_i$, more closely resembles the intuitive weighting factor due to averaging when a small number of neighboring faults is present. Note however that, for larger values of p_{fault}, the faulty cases still produce asymptotic rates that are slightly worse than those of the non-faulty case. The somewhat better convergence of the faulty cases is therefore only a result of the better initial smoothing. Hence one could potentially improve the overall iteration scheme by starting with the dampening parameter due to averaging and then (gradually) switching to the optimal damping parameter after a certain number of iterations. Further experiments showed that the non-faulty case $p_{\mathrm{fault}} = 0$ then indeed exhibits better overall average convergence than the faulty cases and that the convergence for the faulty cases gradually deteriorates with rising values of p_{fault}. However, a proper switching between these two damping parameters is not straightforward and the convergence analysis of such a linear iterative method with dynamically changing damping is quite involved. This is future work.

For the associated conjugate gradient method the convergence behavior is different: We see that there is a sudden reduction of the initial error in the first few iterations by a factor of about 100, which even grows slightly with rising values of γ. This holds for all values of p_{fault}. Here indeed the same smoothing effect due to averaging as in the faulty Richardson case comes into play. However, instead of the fixed ξ_{opt}, the conjugate gradient iteration determines the optimal step size in the preconditioned search direction in the first iteration step. This leads to a substantial smoothing effect in the first (few) iteration(s) and thus to the steep drop of the error as seen in Figure 21 (right). After that the asymptotic error reduction rate gradually sets in. It becomes monotonously worse for rising values p_{fault}, as is intuitively to be expected. Again, the necessary number of iterations is substantially smaller due to the κ-versus-$\sqrt{\kappa}$ effect of the conjugate gradient method compared to the simple linear Richardson iteration. The conjugate gradient approach needs approximately a quarter of the iterations of the Richardson approach in the non-faulty case. But it can be seen that the conjugate gradient method tends to lose its advantage for larger values of p_{fault}. For example, for $\gamma = 2$ and $p_{\mathrm{fault}} = 0.1$, it merely needs approximately half the number of iterations compared to the Richardson approach. Moreover recall that runs, for which the recovery failed due to a larger number of faults than can be repaired with the given values of γ, are discarded in the computation of the average convergence rate. This is the reason why for example for $\gamma = 1/2$ in Figure 21 only averaged convergence plots for $p_{\mathrm{fault}} = 0, 0.01, 0.02$ could be displayed, whereas the curves for the larger values of p_{fault} are

missing. We furthermore see that for the larger overlap $\gamma = 2$ also faults with larger fault probabilities of p_{fault} up to 0.1 could be recovered from and are treated successfully. We finally see that, if p_{fault} is larger than 0, we gradually obtain better results for larger overlap values γ as long as the runs are successful at all. For example, the number of iterations improves with $p_{\text{fault}} = 0.05$ from 50 for $\gamma = 1$ to 43 for $\gamma = 1.5$ and to 37 for $\gamma = 2$. In the case $p_{\text{fault}} = 0$ such a gain for rising values of γ is still there, but it is substantially smaller. In further experiments with larger values of P, like for example $P = 200$, we could observe analogous results with a further improvement in convergence for the faulty situations. This is due to the larger subdomain sizes and the exact subdomain solves, while now larger costs are involved of course.

Altogether, we see that the occurrence of a moderate number of faults does not have much influence on the convergence rate of the fault-tolerant version of the balanced two-level additive Richardson iteration, as already predicted by our convergence theory in Subsection 3.1. We observe that, due to the error reduction in the preasymptotic regime, the necessary number of iterations even decreases compared to the non-faulty case. Analogous observations could be made for the plain Richardson iteration. Note that a similar behavior was already present in the numerical experiments of [GO20], Figure 3, for the case of a failure rate of eight percent using a Weibull distribution. For one, two, five, ten and twenty failing processors in each iteration, which is indeed already an extremely large number of failures regarding the short computation time and the employed number of 100 processors, the influence on the number of iterations is deteriorating slightly and is thus only marginal.

This is somewhat different when considering the associated preconditioned conjugate gradient method. Its convergence in the non-faulty case is substantially faster (i.e. it is dependent on $\sqrt{\kappa}$ instead of just κ), however we no longer have a sound convergence theory in the faulty situation. Indeed, the behavior of the conjugate gradient approach is much more sensitive to the occurrence of faults, as can be seen on the right side of the Figure 21. Now the deviation from the non-faulty convergence rate grows substantially with rising values of p_{fault}. For example, for $\gamma = 2$, the necessary number of iterations grows in the ω_i-weighted balanced conjugate gradient case with $p_{\text{fault}} = 0, 0.01, 0.02, 0.05, 0.1$ with $25, 28, 31, 37, 54$. Note furthermore that this dependence on the fault probability decreases with rising values of the overlap parameter γ. But, all in all, the fault-tolerant version of our conjugate gradient method still exhibits excellent convergence properties in practice.

Next let us consider the convergence behavior of our ω_i-weighted balanced methods for a higher dimensional example. Here we focus on $d = 6$ and again set $P = 100$ and $S = 8$. Thus the size N of the overall problem is fixed to $100 \cdot 2^8$ with a size of 2^8 for each subproblem. Now we study the error reduction property of our fault-tolerant algorithms and consider again, besides the case of no fault at all, the cases where the faults are randomly generated in each cycle for each processor according to a Bernoulli distribution. We again employ the different fault-probabilities $p_{\text{fault}} = 0.01, 0.02, 0.05, 0.1, 0.2$ where, on average, one, two, five, ten and twenty processors fail in each cycle. For the overlap parameter $\gamma = 0.5, 1, 1.5, 2,$

the results for the ω_i-weighted balanced two-level Richardson/Schwarz iteration and the correspondingly preconditioned conjugate gradient method are shown in Figure 22.

Fault Tolerance, Balanced ω_i, $S = 8$, $P = 100$, $q = 16$, $d = 6$
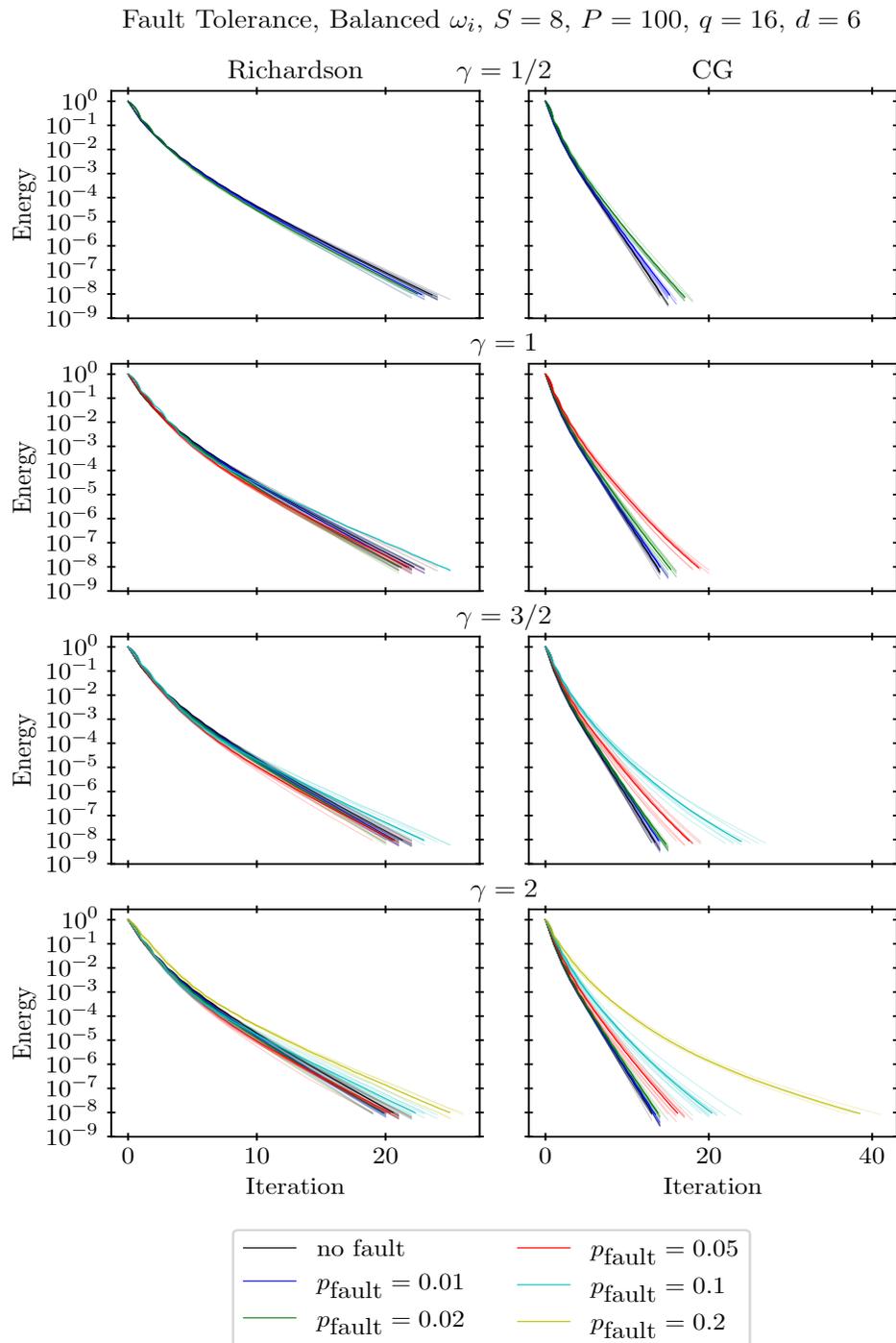
FIGURE 22. Number of iterations (single runs and mean) of the fault-tolerant version of the ω_i-weighted balanced two-level additive Schwarz/Richardson iteration (left) and the associated preconditioned conjugate gradient method (right) for different fault probabilities and different overlaps $\gamma = 0.5, 1, 1.5, 2, q = 16, d = 6$.

We now observe that, for the Richardson case, the preasymptotic error reduction is substantially longer but smoothly changes to the asymptotic convergence, which is in contrast to the one-dimensional case. Furthermore we see that we obtain nearly the same convergence curves for all the different values of p_{fault} for a fixed value of γ. Moreover we now find that the number of iterations stays the same or slightly worsens for larger values of p_{fault} if γ increases. Again we observe that there are no results for large values of p_{fault} but small overlap γ, which stems from the failure of the recovery process due to insufficient redundancy. Consequently, γ should be chosen small but nevertheless still sufficiently large to guarantee the necessary redundancy with respect to the fault rate p_{fault}. This holds for the associated conjugate gradient method as well. Moreover, for $p_{\text{fault}} = 0$, the number of iterations is not very sensitive to γ, it even marginally improves. For larger values of p_{fault} it monotonously improves with rising values of γ. The corresponding number of iterations is still smaller compared to the Richardson case, possibly due to the $\sqrt{\kappa}$-versus-κ effect, but the advantage of the conjugate gradient method now tends to be not as profound anymore. Especially for larger fault rates p_{fault} and larger values of γ the conjugate gradient methods tends to lose its superiority over the Richardson iteration and can even be inferior, as the example $\gamma = 2, p_{\text{fault}} = 0.2$ shows: Now the Richardson approach only needs 25 iterations, whereas the corresponding conjugate gradient method needs 40 iterations. In any case, depending on the choice of the value of p_{fault}, the value of γ must be large enough to allow for a proper recovery after faults.

Finally, we consider the weak scaleup behavior of our ω_i-weighted balanced algorithms under faults. We select a substantial overlap of $\gamma = 3$ and set $d = 1, q = 16, S = 8$. Moreover we allow the values $p_{\text{fault}} = 0, 0.01, 0.02, 0.05, 0.1, 0.2$. This corresponds to $p_{\text{fault}} \cdot P$ faults in each cycle on average for varying numbers of P. The resulting number of iterations are shown in Figure 23. The results for the average convergence rate ρ^{ave} and the asymptotic convergence rate ρ^{asy} are shown in Figure 24.

We see that the weak scaling constant associated to the iterations, i.e. the respective number of iterations for rising numbers of P, first (for $p_{\text{fault}} = 0.01, 0.02, 0.05, 0.1$) marginally improves in the Richardson case and then slightly deteriorates again ($p_{\text{fault}} = 0.2$), which can be explained by the preasymptotic behavior. In the conjugate gradient case it monotonously deteriorates for rising values of p_{fault} as intuitively expected, but the corresponding iteration numbers are much better than those for the Richardson iteration. An analogous behavior can be observed for the average convergence rate ρ^{ave}. In addition the corresponding weak scaling constant for the asymptotic convergence rate ρ^{asy} behaves similarly and is only worse by a value of about 0.02 and about 0.1 than for ρ^{ave} in the Richardson and conjugate gradient case, respectively.

At last, for the six-dimensional case, the weak scaleup behavior of our ω_i-weighted balanced algorithms under faults, i.e. the number of iterations and the average and the asymptotic convergence rate, are shown in the Figures 25 and 26.
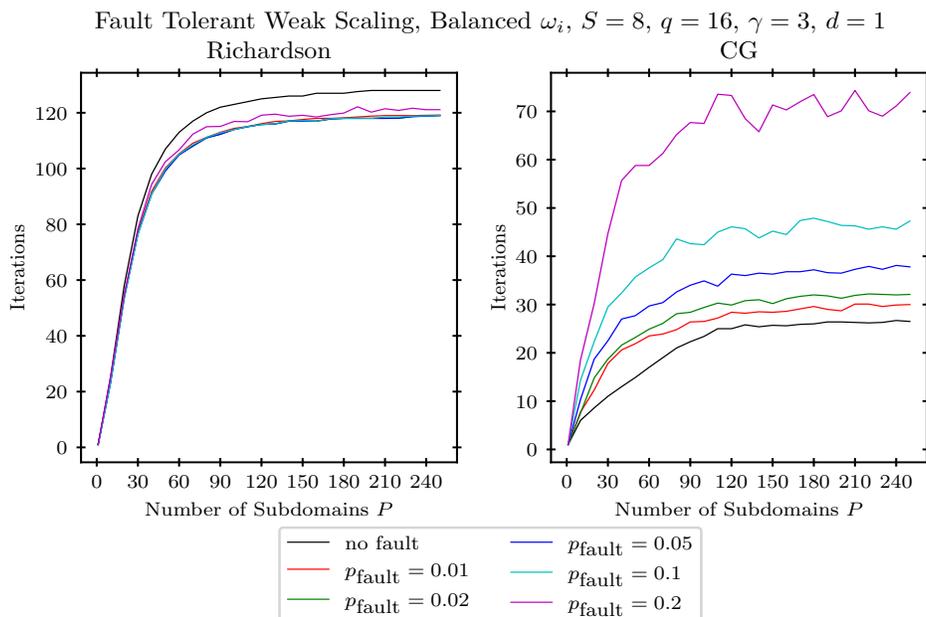
FIGURE 23. Number of iterations (mean) of the fault-tolerant version of the ω_i-weighted balanced two-level additive Schwarz/Richardson iteration (left) and the associated conjugate gradient method (right) for different fault probabilities, $\gamma = 3, S = 8, q = 16, d = 1$.
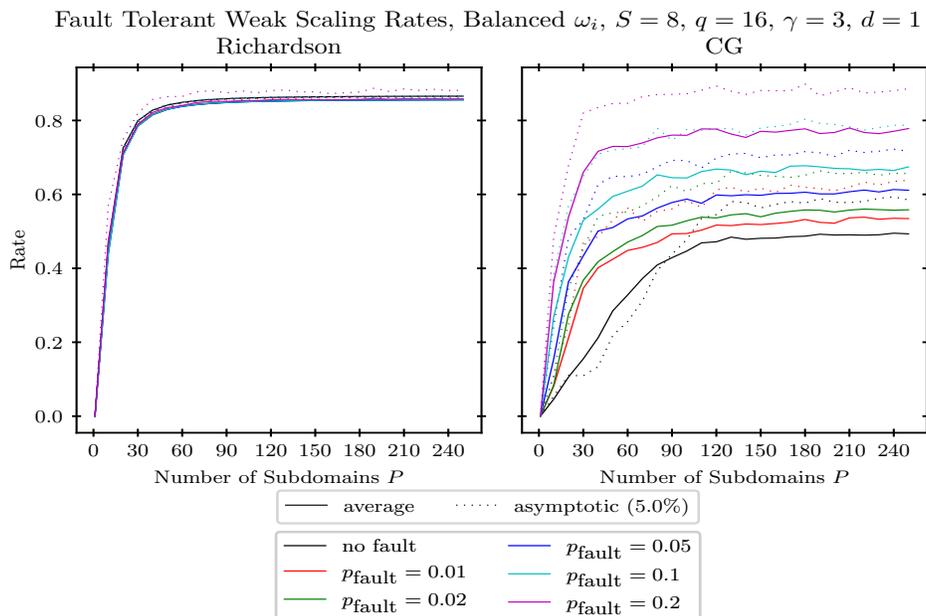
FIGURE 24. Average and asymptotic rates (mean) of the fault-tolerant version of the ω_i-weighted balanced two-level additive Schwarz/Richardson iteration (left) and the associated conjugate gradient method (right) for different fault probabilities, $\gamma = 3, S = 8, q = 16, d = 1$.

Fault Tolerant Weak Scaling, Balanced ω_i, $S = 8$, $q = 16$, $\gamma = 3$, $d = 6$
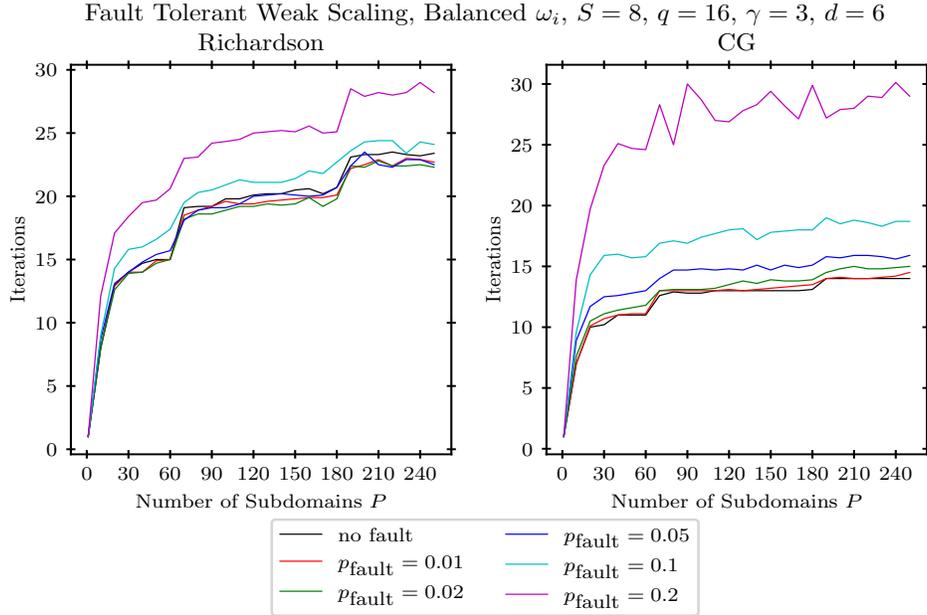
FIGURE 25. Number of iterations (mean) of the fault-tolerant version of the ω_i-weighted balanced two-level additive Schwarz/Richardson iteration (left) and the associated conjugate gradient method (right) for different fault probabilities, $\gamma = 3$, $S = 8, q = 16, d = 6$.

Fault Tolerant Weak Scaling Rates, Balanced ω_i, $S = 8$, $q = 16$, $\gamma = 3$, $d = 6$
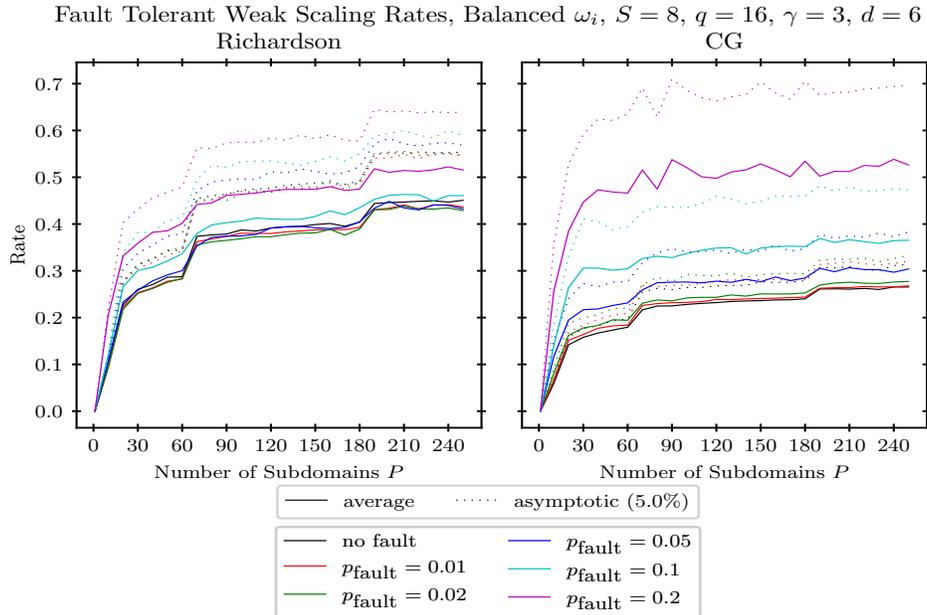
FIGURE 26. Average rate and asymptotic rate (mean) of the fault-tolerant version of the ω_i-weighted balanced two-level additive Schwarz/Richardson iteration (left) and the associated conjugate gradient method (right) for different fault probabilities, $\gamma = 3, S = 8, q = 16, d = 6$.

We see that the algorithm exhibits good weak scaling properties also for the six-dimensional case. For rising values of P, the number of iterations stays at around 23 for the balanced Richardson iteration and is nearly independent of the fault rate (except of the case $p_{\text{fault}} = 0.2$ which needs 28 iterations). Weak scaling can also be observed for the corresponding conjugate gradient method. Now however the weak scaling constant is dependent on the fault rate p_{fault} and increases for rising numbers of faults, albeit with still less iterations than in the Richardson case (except for the case $p_{\text{fault}} = 0.2$). The situation is the same for the average rate and the asymptotic rate is basically the same.

Altogether we conclude that, for a moderate number of faults and a sufficiently large choice of γ, our fault-tolerant algorithms work and scale well with rising numbers of P and associated rising number $p_{\text{fault}} \cdot P$ of faults. Thus we obtain fault-tolerance also in the weak scaling regime.

5. Discussion and conclusion

In the present article we proposed and studied a simple additive two-level overlapping domain decomposition method for d-dimensional elliptic PDEs, which relies on a space-filling curve approach to create equally sized subproblems and to determine an overlap based on the one-dimensional ordering of the space-filling curve. Moreover we employed agglomeration and a purely algebraic Galerkin discretization in the construction of the coarse problem to avoid d-dimensional geometric information. This resulted in a parallel subspace correction iteration and an associated parallel preconditioner for the conjugate gradient method which can be employed for any value of P and in any dimension d. It exhibits the same asymptotic convergence rate as a conventional geometric domain decomposition method, i.e. $\kappa = O(1 + \frac{H}{\delta})$ with $\delta = ch$, compare [TW04, Theorem 3.13] and (4.30), but allows for more flexibility and simplicity with respect to N, P and d due to the space-filling curve construction. Moreover it turned out that its balanced version according to (2.7), which, compared to the plain version, involves two additional matrix-vector-multiplications, three additional vector-vector-additions and one additional solve of the coarse scale problem, was clearly superior and is therefore the recommended approach. To gain fault-tolerance we stored on each processor, besides the data of the associated subproblem, a copy of the coarse problem and also the data of a fixed number of (partial) neighboring subproblems with respect to the one-dimensional ordering of the subproblems induced by the space-filling curve. This redundancy then allows to restore necessary data if processors fail during the computation and to therefore restore convergence in the case of faults. We showed theoretically and practically that the convergence rate of such a linear iteration method for a moderate number of faults stays approximately the same in expectation and only its order constant deteriorates slightly due to the faults. This was also observed in practice for the associated preconditioned conjugate gradient method.

Moreover weak scaling could be achieved in any situation. Altogether, we obtained a fault-tolerant, parallel and efficient domain decomposition method based on space-filling curves, which is especially suited for higher-dimensional elliptic problems.

Note here that we employed an exact subdomain solver in our domain decomposition method. It can be replaced by a faster but iterative and thus somewhat inexact method, like for example algebraic multigrid. This needs to be explored in more detail in the future.

Moreover our algorithmic approach possesses a range of parameters, which influence each other and need to be chosen properly. The determination of an optimal set of parameters is a difficult task. Nevertheless some general guidelines could be derived: One key parameter is the number of degrees of freedom for the coarse grid. Here it became clear that the coarse grid size should be chosen relative to the fine grid size. The distance between the two levels controls the balance between the required number of iterations on the one hand and the cost to solve the coarse system in each iteration on the other hand. Fortunately, as the dimension increases, the size of the coarse grid can be decreased, which reduces the cost for each iteration. Furthermore, as seen from Lemma 2.1, it is sensible to choose another key parameter of our scheme, namely the overlap γ, as an integer multiple of $\frac{1}{2}$, i.e. $\gamma = \frac{1}{2}n, n \in \mathbb{N}_+$. This way, the different possible scalings via D_i and ω_i coincide and boil down to a constant scaling with $1/(2\gamma + 1)$ and symmetry of the associated operator is guaranteed, at least for the Laplacian and diffusion operators with constant coefficients. Furthermore the choice of γ as an integer multiple of $\frac{1}{2}$ is also natural for a uniform redundancy of stored data, which is advantageous for the recovery and reconstruction in the case of faults. Moreover we learned from our convergence studies and from our fault tolerance experiments that γ can be chosen to be quite small for any real world setting. For example, Figures 21 and 26 showed that $\gamma = 0.5$ or $\gamma = 1$ still produce excellent convergence for both considered types of solvers for fault rates of up to 2% and 5% per iteration/cycle, respectively. These fault rates are indeed way beyond those of any existing high performance compute system, where merely one or very few faults currently occur on average each day. They are also beyond the expected fault rates for future exascale compute systems. For practical purposes this means that we will see no deterioration of the convergence rates at all, as long as the data recovery process is executed properly. This is guaranteed due to the overlap and the redundant parallel storage of current data and local parts of the iterate which allows the algorithm to recover after a fault occurs. In this respect, iterative methods are fault-forgiving and the iteration can continue dynamically. Moreover our aim was to derive a fault-tolerant parallel domain decomposition solver, which can be later employed for the subproblems in the combination method. There we encounter short run times, since the numbers of degrees of freedom of all the arising subproblems are similar to the one-dimensional case, i.e. of size $\approx 2^L$ only, compare (2.9). But this is good news: The fault-repair mechanism is rarely needed for real-life fault probabilities within the short run times of our solver for the small subproblems arising in the combination method.

We next intend to employ our domain decomposition method as the inner, parallel and fault-tolerant solver for the subproblems arising in the sparse grid combination method. Let us recall the sparse grid combination formula from (2.9), i.e.

$$(5.33) \qquad u(x) \approx u_L^{(c)}(x) := \sum_{i=0}^{d-1} (-1)^i \begin{pmatrix} d-1 \\ i \end{pmatrix} \sum_{|l|_1 = L+(d-1)-i} u_l(x).$$

Here the different subproblems can be computed in parallel independently of each other. Moreover, for layer i, we have

$$\begin{pmatrix} L+d-2-i \\ d-1 \end{pmatrix}$$

different subproblems, where each subproblem has approximately the same number $N(d,i) = \prod_{j=1}^{d}(2^{l_j}+1) = O(2^{L+d-1-i})$ of degrees of freedom. We now employ an additional step of parallelization by means of the domain decomposition treatment of each of these subproblems. To this end, we use

$$(5.34) \qquad\qquad\qquad P := \hat{P} \cdot 2^{d-1-i}$$

subdomains and thus processors for each subproblem on layer i. This choice of a d- and i-dependent P via a universal $\hat{P}$ in (5.34) results in a subdomain size of roughly $N(d,i)/P = 2^{L+d-1-i}/P = 2^{L+d-1-i}/(\hat{P} \cdot 2^{d-1-i}) = 2^L/\hat{P}$, which is independent of d and i. Then, in our elliptic situation and except for the coarse scale problems, only small subdomain problems appear. But, depending on d, L and $\hat{P}$, there can be millions of these subdomains for the combination method. To be precise, the amount of fine level subdomains for the overall set of subproblems in the combination method (2.9) is

$$\frac{\hat{P}}{(d-1)!} \sum_{k=0}^{d-1} \left(2^{d-1-k} \prod_{i=1}^{d-1} (L+d-1-k-i) \right),$$

where each occurring subdomain problem has approximately equal size $2^L/\hat{P}$.

As an example, consider the goal of an overall discretization error of 10^{-12}. Then $h \approx 10^{-6}$ and $L = 20$. Moreover we choose $\hat{P} = 2^{10}$. For this case, Table 1 gives the number of subdomain problems arising in the combination method if our domain decomposition approach is employed with P given by (5.34) for each subproblem of layer i.

$d =$	1	2	3	4	5	6
	$1 \cdot \hat{P}$	$59 \cdot \hat{P}$	$1.391 \cdot \hat{P}$	$20.889 \cdot \hat{P}$	$237.706 \cdot \hat{P}$	$1.754.744 \cdot \hat{P}$

TABLE 1. Overall number of subdomain problems, each one of size $2^L/\hat{P}$, in the combination method with domain decomposition of each subproblem.

We see that we obtain a large number of subdomain problems to be solved in a doubly parallel way (one level of parallelism stems from the combination formula itself, the other stems from the domain decomposition of each subproblem of the combination method

into subdomain problems). This will allow the use of extremely large parallel compute systems, i.e. the larger d is the larger the parallel system is that can be employed in a meaningful way. Furthermore the use of the fault-tolerant domain decomposition method as the inner solver for the subproblems in the combination method results in a fault-tolerant and parallel solver for the combination method in a natural way. There, the fault-repair mechanism is provided on the fine domain decomposition level and not just on the coarse subproblem level of the combination method, as it was previously done in [HHLS14, PBG14, ASHH16, OHHBP17, LOP20].

For our simple elliptic Poisson problem we encounter very short computing times in the 10 second range for each of the subproblems arising in the combination method, and therefore a fault occurs extremely rarely during its run time. The true value of our fault-tolerant approach will rather become apparent in time-dependent parabolic problems with long time horizons and long computation times, where our elliptic DDM-solver is employed in each time step of an implicit time-discretization method within the combination method. This however is future work.

References

[A20] E. Agullo et al. Resiliency in numerical algorithm design for extreme scale simulations. Report Dagstuhl Seminar 20101, March 1–6, 2020. arXiv:2010.13342v1, submitted to the International Journal of High Performance Computing Applications.

[AAD11] H. Alcin, O. Allain, and A. Dervieux. Volume-agglomeration coarse grid in Schwarz algorithm. In: J. Fort, J. Fürst, J. Halama, R. Herbin and F. Hubert (eds). Finite Volumes for Complex Applications VI, Problems and Perspectives. Springer Proceedings in Mathematics, Volume 4, Springer, Berlin, Heidelberg, 3–11, 2011.

[ASHH16] M. Ali, P. Strazdins, B. Harding, and M. Hegland. Complex scientific applications made fault-tolerant with the sparse grid combination technique. *The International Journal of High Performance Computing Applications* 30:335–359, 2016.

[P20] G. Altai. //pypi.org/project/hilbertcurve/, retrieved 06.10.2020.

[B13] M. Bader. Space-Filling Curves: An Introduction with Applications in Scientific Computing, Texts in Computations Science and Engineering, Volume 9, Springer-Verlag, 2013.

[BH03] R. Bank and M. Holst. A new paradigm for parallel adaptive meshing algorithms. *SIAM Review* 45(2):291–323, 2003.

[BDK15] H. Bouwmeester, A. Dougherty, and A. Knyazev. Nonsymmetric preconditioning for conjugate gradient and steepest descent methods. *Procedia Computer Science* 51:276–285, 2015.

[B86] J. Bramble, J. Pasciak and A. Schatz. The construction of preconditioners for elliptic problems by substructuring, I. *Math. Comp.* 47(175):103–134, 1986.

[BPS89] J. Bramble, J. Pasciak, and A. Schatz. The construction of preconditioners for elliptic problems by substructuring, IV. *Math. Comp.* 53(187):1–24, 1989.

[B00] S. Brenner. Lower bounds for two-level additive Schwarz preconditioners with small overlap. *SIAM J. Sci. Comput.* 21(5):1657–1669, 2000.

[BG04] H.-J. Bungartz and M. Griebel. Sparse grids. *Acta Numerica* 13:147–269, 2004

[BGRZ94] H.-J. Bungartz, M. Griebel, D. Röschke, and C. Zenger. Two proofs of convergence for the combination technique for the efficient solution of sparse grid problems. In: D. Keyes and J. Xu, editors, *Domain Decomposition Methods in Scientific and Engineering Computing, DDM7*, Contemp. Math. 180, 15–20. American Mathematical Society, Providence, 1994

[B71] A. Butz. Alternative algorithm for Hilbert's space-filling curve, *IEEE Transactions on Computers* 20(4):424-426, 1971.

[C78] G. Cantor. Ein Beitrag zur Mannigfaltigkeitslehre, *J. Reine Angew. Math. (Crelle's Journal)* 84:242-258, 1878.

[C17] P. Chernoch. Mapping N-dimensional value to a point on Hilbert curve, //stackoverflow.com/questions/499166/mapping-n-dimensional-value-to-a-point-on-hilbert-curve/10384110#10384110, retrieved 01.04.2017.

[C20] P. Chernoch. //github.com/paulchernoch/HilbertTransformation, retrieved 06.10.2020.

[CS99] X. Cai and M. Sarkis. A restricted additive Schwarz preconditioner for general sparse linear systems. *SIAM J. Sci. Comput.* 21:239–247, 1999.

[DW87] M. Dryja and O. Widlund. An additive variant of the Schwarz alternating method for the case of many subregions. Ultracomputer Note 131, Department of Computer Science, Courant Institute, New York, 1987.

[DW94] M. Dryja and O. Widlund. Domain decomposition algorithms with small overlap. *SIAM J. Sci. Comput.* 15(3):604–620, 1994.

[DJN15] V. Dolean, P. Jolivet, and F. Nataf. An Introduction to Domain Decomposition Methods: Algorithms, Theory, and Parallel Implementation, SIAM, 2015.

[G12] J. Garcke. Sparse Grids in a Nutshell. In: J. Garcke and M. Griebel, editors, *Sparse Grids and Applications,* Volume 88 of *Lecture Notes in Computational Science and Engineering*, Springer, 57–80, 2012.

[Gri92] M. Griebel. The combination technique for the sparse grid solution of PDEs on multiprocessor machines. *Parallel Processing Letters* 2(1):61–70, 1992.

[GSZ92] M. Griebel, M. Schneider, and C. Zenger. A combination technique for the solution of sparse grid problems. In: P. de Groen and R. Beauwens, editors, *Iterative Methods in Linear Algebra*, IMACS, Elsevier, North Holland, 263–281. 1992.

[GH14] M. Griebel and H. Harbrecht. On the convergence of the combination technique. In: *Sparse grids and Applications.* Volume 97 of *Lecture Notes in Computational Science and Engineering*, Springer, 55–74. 2014.

[GKZ07] M. Griebel, S. Knapek, and G. Zumbusch. Numerical Simulation in Molecular Dynamics. Springer, Berlin, Heidelberg, 2007.

[GO95] M. Griebel and P. Oswald. On the abstract theory of additive and multiplicative Schwarz algorithms. *Numer. Math.* 70:163–180, 1995.

[GO12] M. Griebel and P. Oswald. Greedy and randomized versions of the multiplicative Schwarz method. *Linear Algebra Appl.* 437:1596–1610, 2012.

[GO16] M. Griebel and P. Oswald. Schwarz iterative methods: Infinite space splittings. *Constructive Approximation* 33(1):121–139, 2016.

[GO18] M. Griebel and P. Oswald. Stochastic subspace correction in Hilbert space. *Constructive Approximation* 48(3):501–521, 2018.

[GO20] M. Griebel and P. Oswald. Stochastic subspace correction methods and fault tolerance. *Math. Comp.* 89(321):279–312, 2020.

[GZ00] M. Griebel and G. Zumbusch. Parallel adaptive subspace correction schemes with applications to elasticity. *Computer Methods in Applied Mechanics and Engineering* 184:303–332, 2000.

[GZ01] M. Griebel and G. Zumbusch. Hash based adaptive parallel multilevel methods with space-filling curves. In: H. Rollnik and D. Wolf, editors, *NIC Symposium 2001*, Volume 9 of *NIC Series*, Forschungszentrum Jülich, Germany, 479–492. 2002.

[HHLS14] B. Harding, M. Hegland, J. Larson, and J. Southern. Fault tolerant computation with the sparse grid combination technique. *SIAM J. Sci. Comput.* 37(03):C331–C353, 2014.

[HGRW16] M. Huber, B. Gmeiner, U. Rüde, and B. Wohlmuth. Resilience for massively parallel multigrid solvers. *SIAM J. Sci. Comput.* 38(5):S217–S239, 2016.

[H91] D. Hilbert. Über die stetige Abbildung einer Linie auf ein Flächenstück, *Mathematische Annalen* 38:459–460, 1891.

[JYS19] D. Jauk, D. Yang, and M. Schulz. Predicting faults in high performance computing systems: An in-depth survey of the state-of-the-practice. in *SC '19: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Article No. 30, 1–13, 2019.

[KRS13] S. Kavila, P. Raju, S. Satapathy, A. Machiraju, G. Kinnera, and K. Rasly. A survey on fault management techniques in distributed computing. In: S. Satapathy et al., editor, *Proc. of Int. Conf. on Front. of Intell. Comput.*, AISC 199, 593–602, 2013.

[LOP20] R. Lago, M. Obersteiner, T. Pollinger, J. Rentrop, H.-J. Bungartz, T. Dannert, M. Griebel, F. Jenko, and D. Pflüger. EXAHD – A massively parallel fault-tolerant sparse grid approach for high-dimensional turbulent plasma simulations. In: H.-J. Bungartz, S. Reiz, B. Uekermann, P. Neumann, and W. Nagel, editors, *Software for Exascale Computing - SPPEXA 2016-2019*, Volume 136 of *Lecture Notes in Computational Science and Engineering*, Springer, 301-329, 2020.

[L00] J. Lawder. Calculation of mappings between one and n-dimensional values using the Hilbert space-filling curve, research report BBKCS-00-01, School of Computer Science and Information Systems, Birkbeck College, University of London, 2000.

[L04] H. Lebesgue. Leçons sur l'Intégration et la Recherche des Fonctions Primitives, Gauthier-Villars, 1904.

[M93] J. Mandel. Balancing domain decomposition. *Comm. Numer. Methods Engrg.* 9:233-241, 1993.

[M88] L. Mansfield. On the use of deflation to improve the convergence of conjugate gradient iteration. *Comm. Applied Numer. Meth.* 4:151-156, 1988.

[M90] L. Mansfield. On the conjugate gradient solution of the Schur complement system obtained from domain decomposition. *SIAM J. Numer. Anal.* 27(6):1612–1620, 1990.

[M00] D. Moore. Fast Hilbert curve generation, sorting, and range queries. http://www.tiac.net/-sw/2008/10/Hilbert/moore/, retrieved 01.04.2017.

[N79] E. Netto. Beitrag zur Mannigfaltigkeitslehre, *J. Reine Angew. Math. (Crelle's Journal)* 86:263-268, (1879).

[N87] R. Nicolaides. Deflation of conjugate gradients with applications to boundary value problems. *SIAM J. Numer. Anal.* 24:355-365, 1987.

[OHHBP17] M. Obersteiner, A. Parra Hinojosa, M. Heene, H. Bungartz, and D. Pflüger. A highly scalable, algorithm-based fault-tolerant solver for gyrokinetic plasma simulations. In: *ScalA '17: Proceedings of the 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*, November 2017, Article No 2, 1–8, 2017.

[PAS14] S. Pauli, P. Arbenz, and C. Schwab, Intrinsic fault tolerance of multi-level Monte Carlo methods, In: *Parallel Computing: Accelerating Computational Science and Engineering (CSE)*, M. Bader et al., editors, IOS Press, 2014, 471–480.

[P90] G. Peano. Sur une courbe qui remplit toute une aire plane. *Math. Ann.* 36:157-160, (1890).

[PBG14] D. Pflüger, H.-J. Bungartz, M. Griebel, F. Jenko, T. Dannert, M. Heene, C. Kowitz, A. Hinojosa, and P. Zaspel. EXAHD: An exa-scalable two-level sparse grid approach for higher-dimensional problems in plasma physics and beyond. In: L. Lopes et al, editors, *Euro-Par 2014: Parallel Processing Workshops*, Volume 8806 of *Lecture Notes in Computer Science*, Springer International Publishing, 565–576. 2014.

[QV99] A. Quarteroni and A. Valli. Domain Decomposition Methods for Partial Differential Equations, Oxford Science Publications, 1999.

[S94] H. Sagan. Space-Filling Curves, Universitext, Springer-Verlag, New York, 1994.

[S70] H. Schwarz. Über einen Grenzübergang durch alternierendes Verfahren. *Vierteljahrsschrift der Naturforschenden Gesellschaft in Zürich* 15:272–286, 1870.

[SG10] B. Schroeder and G. Gibson. A large-scale study of failures in high-performance computing Systems. *IEEE Transactions on Dependable and Secure Computing*, 7(4):337-350, 2010.

[Sh20] A. Shavit. //github.com/adishavit/hilbert/blob/master/hilbert.c#L529, retrieved 06.10.2020.

[S04] J. Skilling. Programming the Hilbert curve, *AIP Conference Proceedings* 707:381-387, 2004.

[S90] B. Smith. Domain Decomposition Algorithms for the Partial Differential Equations of Linear Elasticity. Ph.D. Thesis, Courant Institute, New York, 1990.

[SBWG96] B. Smith, P. Bjorstad, and W. Gropp. Domain Decomposition, Parallel Multilevel Methods for Elliptic Partial Differential Equations, Cambridge University Press, 1996.

[SW14] M. Snir, R. Wisniewski, et al. Addressing failures in exascale computing. *Int. J. High Perf. Comput. Appl.* 28(2):129–173, 2014.

[Sp20] A. Spencer. //www.math.utah.edu/cgi-bin/man2html.cgi?/usr/local/man/man3/hilbert.3, retrieved 06.10.2020.

[Sta19] L. Stals. Algorithm-based fault recovery of adaptively refined parallel multilevel grids. *The International Journal of High Performance Computing Applications* 33:189 − 211, 2019.

[T92] S. Thomas. hilbert.c, in fossies.org, URT/urt-3.1a/lib/hilbert.c, The Utah Raster Toolkit, fossies.org/linux/misc/old/URT3.1a.tar.gz/URT/urt-3.1a/lib/hilbert.c, retrieved 01.04.2017.

[Tre05] M. Treaster. A survey of fault-tolerance and fault-recovery techniques in parallel systems. *ACM Computing Research Repository* CoRR 501002:1–11, 2005.

[TW04] A. Toselli and O. Widlund. Domain Decomposition Methods - Algorithms and Theory. *Springer Series in Computational Mathematics*, Volume 34, Springer, Berlin, Heidelberg, 2004.

[VN06] C. Vuik and R. Nabben. A comparison of deflation and the balancing preconditioner. *SIAM J. Sci. Comput.* 27(5):1742–1759, 2006.

[W09] O. Widlund. The development of coarse spaces for domain decomposition algorithms. In: M. Bercovier, M. Gander, R. Kornhuber and O. Widlund, editors. *Domain Decomposition Methods in Science and Engineering XVIII.* Volume 70 of *Lecture Notes in Computational Science and Engineering*, Springer, Berlin, Heidelberg, 241–248, 2009.

[YWWYZ12] Y. Yuan, Y. Wu, Q. Wang, G. Yang, W. Zheng, Job failures in high performance computing systems: A large-scale empirical study. *Computers and Mathematics with Applications* 63(2):365-377, 2012.

[Z91] C. Zenger. Sparse grids. In: *Parallel algorithms for partial differential equations*, Proceedings of the 6th GAMM-Seminar, Kiel/Germany, 1990, Volume 31 of *Notes Numer. Fluid Mech.*, Vieweg, Braunschweig, 241–251, 1991.

[Z03] G. Zumbusch. Parallel Multilevel Methods. Adaptive Mesh Refinement and Loadbalancing, Teubner, 2003.

MICHAEL GRIEBEL, INSTITUT FÜR NUMERISCHE SIMULATION, UNIVERSITÄT BONN, FRIEDRICH-HIRZEBRUCH-ALLEE 7, 53115 BONN, AND FRAUNHOFER INSTITUTE FOR ALGORITHMS AND SCIENTIFIC COMPUTING (SCAI), SCHLOSS BIRLINGHOVEN, 53754 SANKT AUGUSTIN, GERMANY

Email address: griebel@ins.uni-bonn.de

MARC-ALEXANDER SCHWEITZER, INSTITUT FÜR NUMERISCHE SIMULATION, UNIVERSITÄT BONN, FRIEDRICH-HIRZEBRUCH-ALLEE 7, 53115 BONN, AND FRAUNHOFER INSTITUTE FOR ALGORITHMS AND SCIENTIFIC COMPUTING (SCAI), SCHLOSS BIRLINGHOVEN, 53754 SANKT AUGUSTIN, GERMANY

Email address: schweitzer@ins.uni-bonn.de

LUKAS TROSKA, INSTITUT FÜR NUMERISCHE SIMULATION, UNIVERSITÄT BONN, FRIEDRICH-HIRZEBRUCH-ALLEE 7, 53115 BONN, AND FRAUNHOFER INSTITUTE FOR ALGORITHMS AND SCIENTIFIC COMPUTING (SCAI), SCHLOSS BIRLINGHOVEN, 53754 SANKT AUGUSTIN, GERMANY

Email address: lukas.troska@scai.fraunhofer.de