Domain Overlap for Iterative Sparse Triangular Solves on GPUs

Hartwig Anzt, Edmond Chow,
Daniel B. Szyld, and Jack Dongarra

Report 15-11-24
Novermber 2015. Revised February 2016

Department of Mathematics
Temple University
Philadelphia, PA 19122

This report is available in the World Wide Web at
`http://www.math.temple.edu/~szyld`

Domain Overlap for Iterative Sparse Triangular Solves on GPUs

Hartwig Anzt, Edmond Chow, Daniel B. Szyld, and Jack Dongarra

Abstract Iterative methods for solving sparse triangular systems are an attractive alternative to exact forward and backward substitution if an approximation of the solution is acceptable. On modern hardware, performance benefits are available as iterative methods allow for better parallelization. In this paper, we investigate how block-iterative triangular solves can benefit from using overlap. Because the matrices are triangular, we use "directed" overlap, depending on whether the matrix is upper or lower triangular. We enhance a GPU implementation of the block-asynchronous Jacobi method with directed overlap. For GPUs and other cases where the problem must be overdecomposed, i.e., more subdomains and threads than cores, there is a preference in processing or scheduling the subdomains in a specific order, following the dependencies specified by the sparse triangular matrix. For sparse triangular factors from incomplete factorizations, we demonstrate that moderate directed overlap with subdomain scheduling can improve convergence and time-to-solution.

1 Introduction

Sparse triangular solves are an important building block when enhancing Krylov solvers with an incomplete LU (ILU) preconditioner [28]. Each iteration of the solver requires the solution of sparse triangular systems involving the incomplete

Hartwig Anzt
University of Tennessee, e-mail: hanzt@icl.utk.edu

Edmond Chow
Georgia Institute of Technology, e-mail: echow@cc.gatech.edu

Daniel B. Szyld
Temple University, e-mail: szyld@temple.edu

Jack Dongarra
University of Tennessee, e-mail: dongarra@icl.utk.edu

factors. Exact solves with sparse triangular matrices are difficult to parallelize due to the inherently sequential nature of forward and backward substitution. Level scheduling strategies [28] aim at identifying sets of unknowns that can be computed in parallel (called "levels"), but these sets are often much smaller than the parallelism provided by the hardware. Particularly on manycore architectures like graphics processing units (GPUs), level-scheduling techniques generally fail to exploit the concurrency provided.

At the same time, the incomplete factorizations are typically only a rough approximation, and exact solutions with these factors may not be required for improving the convergence of the Krylov solver. Given this situation, interest has developed in using "approximate triangular solves" [3]. The concept is to replace the exact forward and backward substitutions with an iterative method that is easy to parallelize. Relaxation methods like the Jacobi method provide parallelism across vector components, and can be an attractive alternative when running ILU-preconditioned Krylov methods on parallel hardware. For problems where only a few steps of the iterative method applied to the sparse triangular systems are sufficient to provide the same preconditioning quality to the outer Krylov method, the approximate approach can be much faster [15, 14]. A potential drawback of this strategy, however, is that disregarding the dependencies between the vector components can result in slow information propagation. This can, in particular, become detrimental when using multiple local updates for better cache reuse [5]. In this paper, we investigate improving the convergence of approximate sparse triangular solves by using overlap strategies traditionally applied in domain decomposition methods. Precisely, we enhance a block-iterative method with restricted Schwarz overlap, and analyze the effect of non-uniform overlap that reflects the information propagation dependencies. The findings gained are then used to realize overlap in a GPU implementation of block-asynchronous Jacobi.

The remainder of the paper is structured as follows. Section 2 provides some background on sparse triangular solves, block-asynchronous Jacobi, and different types of Schwarz overlap with the goal of setting the larger context for this work. In Sect. 3, the benefits of restricted additive Schwarz and directed overlap are investigated for different synchronization strategies, and with specific focus on sparse triangular systems arising from incomplete factorization preconditioners. Section 4 gives details about how we realized overlap in the GPU implementation. In Sect. 5, we analyze the convergence and performance improvements we achieved by enhancing block-asynchronous Jacobi with restricted overlap. We conclude in Sect. 6.

2 Background and Related Work

2.1 Sparse Triangular Solves

Due to their performance-critical impact when used in preconditioned Krylov methods, much attention has been paid to the acceleration of sparse triangular solves. The traditional approach tries to improve the exact solves. The most common strategies are based on level scheduling or multi-color ordering [2, 30, 21, 22, 23]. A more disruptive approach is to use partitioned inverses [1, 27], where the triangular matrix is written as a product of sparse triangular factors, and each triangular solve becomes a sequence of sparse matrix vector multiplications. Also, the use of sparse approximate inverses for the triangular matrices were considered [17, 11]. The solution of the triangular systems is then replaced by the multiplication with two sparse matrices that are approximating the respective inverses of the triangular factors. With the increase in parallelism that is available in hardware, *iterative* approaches to solving triangular systems become tantalizing, as they provide much finer grained parallelism. In situations where an approximate solution is acceptable, which often is the case for incomplete factorization preconditioning, iterative triangular solves can accelerate the overall solution process significantly, even if convergence is slightly degraded [15, 3]. With regard to the increased parallelism expected for future hardware systems, iterative triangular solves are also attractive from the standpoint of fault-tolerance [8].

2.2 Jacobi Method and Block-Asynchronous Iteration

Classical relaxation methods like Jacobi and Gauss-Seidel use a specific update order of the vector components, which implies synchronization between the distinct iterations. The number of components that can be computed in parallel in an iteration depends on whether the update of a component uses only information from the previous iteration (Jacobi type) or also information from the current iteration (Gauss-Seidel type). Using newer information generally results in faster convergence, which however reduces the parallelism: Gauss-Seidel is inherently sequential and requires a strict update order; for Jacobi, all components are updated simultaneously within one iteration [3]. If no specific update order is enforced, the iteration becomes "chaotic" or "asynchronous" [13, 19]. In this case, each component update takes the newest available values for the other components. The asymptotic convergence of asynchronous iterations is guaranteed if the spectral radius of the positive iteration matrix, $\rho(|M|)$, is smaller than unity [19]. This is a much stronger requirement than needed for Jacobi, however it is always fulfilled for sparse triangular systems [15, 3]. The fine-grained parallelism and the lack of synchronization make asynchronous methods attractive for graphics processing units (GPUs), which themselves operate in an asynchronous-like fashion within one kernel operation [24]. In

particular, the special case where subsets of the iteration vector are iterated in synchronous Jacobi fashion and asynchronous updates are used in-between the subsets can efficiently be realized on GPUs [7]. The potential of this "block-asynchronous Jacobi" on GPU hardware was investigated in [5]. Block asynchronous Jacobi was also considered as smoother for geometric multigrid methods [6], and evaluated in a mixed-precision iterative refinement framework [4]. In [3], block-asynchronous Jacobi was employed as an iterative solver for sparse triangular systems arising from incomplete factorization preconditioning. Precisely, the benefits of replacing exact sparse triangular solves with approximate triangular solves were demonstrated for an ILU-preconditioned Krylov solver running on GPUs. This work ties on the findings presented therein by enhancing the block-asynchronous Jacobi method with overlap strategies.

2.3 Overlapping Domains and Restricted Additive Schwarz

The idea of improving information propagation by overlapping blocks originated with domain decompositions methods. In these methods, a large domain is split into subdomains, where local solution approximations are computed. A global solution is generated by iteratively updating the local parts and communicating the components of the solutions in the domain intersections. Increasing the size of these intersections usually accelerates the information propagation [32, 34]. In the alternating Schwarz method, the subdomains are processed sequentially in a fixed order. The additive Schwarz method performs subdomains solves in parallel. To avoid write conflicts in the overlap region, the update of the global solution must be implemented carefully. One approach that has proven to be very effective is "Restricted additive Schwarz" (RAS) proposed in [12], where each processor restricts the writing of the solution to the local subdomain, and discards the part of the solution in the region that overlaps other subdomains. The convergence properties of RAS are analyzed in [20]. An initial asynchronous approach for an additive Schwarz method is presented in [18]. The underlying idea is very similar to the work presented in this paper, however it starts with a physical domain decomposition problem and then allows for asynchronism in the update order. In contrast, the approach we present starts from an arbitrary linear system that is interpreted as a domain decomposition problem with domain sizes induced by the GPU thread block size used in the block-asynchronous Jacobi method. Some theoretical convergence results for asynchronous iterations with overlapping blocks can also be found in [33].

3 Random-Order Alternating Schwarz

3.1 Domain overlap based on matrix partitioning

In domain decomposition methods for solving partial differential equations (PDEs), the subdomains are usually contiguous, physical subdomains of the region over which a solution is sought. In this work, however, we adopt a black-box-solver setting, where no details about the physical problem or domain is available. This is a reasonable premise as many software packages must accommodate this situation. We note that in this case, incomplete factorization preconditioners are often employed, as they work well for a large range of problems.

For a matrix problem, we call a subdomain the set of unknowns corresponding to a partition of unknowns (e.g., partitioning of the unknown vector) which may not correspond to a physical subdomain even if the problem comes from a PDE. Here, the overlap for the distinct subdomains can not be derived from the physical domain, but has to be generated from the dependency graph of the matrix. This strategy was first proposed in [9]. We use the terminology introduced therein by calling this kind of overlap "algebraic Schwarz overlap". Algebraic Schwarz overlap of level 1 is generated by including all unknowns that are distant by one edge to the subdomain when solving the local problem. Recursively applying this strategy results in overlap of higher levels: for level o overlap, all unknowns distant by at most o edges are considered. See Fig. 1 for an illustration of algebraic Schwarz overlap.

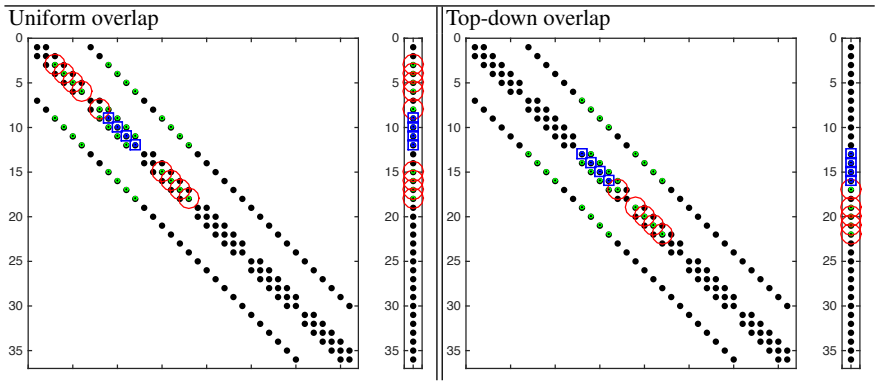

Fig. 1: Algebraic Schwarz overlap for a 5-point stencil discretization of the Laplace problem in 2D. The unknowns of the original subdomain are marked as blue squares, the overlap of level one is derived from the matrix structure and indicated by red circles. The left figure shows uniform overlap, the right figure shows top-down overlap

In a first experiment, we analyze the effect of overlap for a block-iterative solver. The target problem is a finite difference discretization of the Laplace problem in

3D. The discretization uses a 27-point stencil on a $8 \times 8 \times 8$ grid, resulting in a symmetric test matrix where 10,648 edges connect 512 unknowns. The block-iterative method splits the iteration vector into 47 blocks that we call "subdomains" to be consistent with domain decomposition terminology. On each subdomain, the local problem is solved via 2 Jacobi sweeps. Subdomain overlap is generated as algebraic Schwarz overlap. Restricted Schwarz overlap only updates the components part of the original subdomain. The motivation for restricting the results also in sequential subdomain updates is the GPU architecture we target in the experimental part of the paper. There, multiple subdomains are updated in parallel. All experiments in this section are based on a MATLAB implementation (release R2014a) running in double precision.

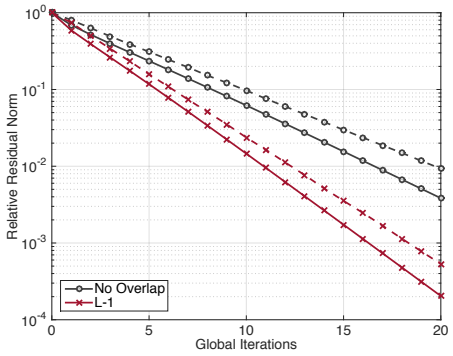

Fig. 2: Convergence of the restricted alternating Schwarz using 2 Jacobi sweeps as local solver on the subdomains applied to the Laplace test problem. The subdomain updates are scheduled in Gauss-Seidel order *(solid lines)*, or in a random order *(dashed lines)*

Figure 2 shows how restricted alternating Schwarz using level 1 overlap improves the convergence rate when solving the above Laplace problem. Each subdomain is updated once in a global iteration, and the results are averaged over 100 runs. The solid lines are for sequential top-down subdomain scheduling (Gauss-Seidel), the dashed lines are for a random update order. In the remainder, we call the latter "random-order restricted alternating Schwarz". Random subdomain scheduling results in slower average convergence, but the improvement obtained from restricted Schwarz overlap is of similar quality. We note that extending the original subdomains with overlap increases the computational cost, as on each subdomain a larger local problem has to be solved. For this test problem, level 1 overlap increases the total number of floating point operations for one global sweep by a factor of 6.68. This issue is not reflected in Fig. 2 showing the convergence with respect to global iterations. The increase in the computational cost does however typically not reflect the increase in execution time, as in a parallel setting also the communication plays an important role. In the end, the interplay between hardware characteristics, the

linear system, and the used decomposition into subdomains determines whether the time-to-solution benefits from using overlapping subdomains [31].

3.2 Directed Overlap

The purpose of using overlap is to propagate information faster across the local problems. More overlap usually results in faster convergence. However, it can be expected that not all the information propagated provide the same convergence benefits:

1. It can be expected that propagating information in the dependency direction of the unknowns may provide larger benefit.
2. For non-parallel subdomain updates, propagating information from subdomains already updated in the current iteration may provide larger benefit than information from the previous iterate.

A non-directed or bidirected dependency graph (structurally symmetric matrix) makes it impossible to benefit from scheduling the subdomains in dependency order. For each dependency that is obeyed, the opposite dependency is violated. In this case, the optimization of the information propagation boils down to propagating primarily information from subdomains that have already been updated in the current iteration. For the sequential subdomain scheduling in top-down Gauss-Seidel order, domain overlap pointing opposite the subdomain scheduling order propagates information from already updated subdomains. Overlap pointing in the scheduling direction propagates information of the previous iteration. Hence, bottom-up overlap may carry "more valuable" information than overlap pointing top-down. For the remainder of the paper we use the term "directed overlap" if the original subdomain is extended only in a certain direction:

- "Top-down overlap" means that the original subdomain is extended by unknowns adjacent in the graph representation of the matrix that have larger indexes, i.e., are located closer to the end of the iteration vector;
- "Bottom-up overlap" means that the original subdomain is extended by unknowns adjacent in the graph representation of the matrix that have smaller indexes, i.e., are located closer to the top of the iteration vector;

We note, that in case the problem originates from a discretization of a partial differential equation, the concept of directed overlap does in general not correspond to a direction in the physical domain. An example where the directed overlap has a physical representation is a 1-dimensional physical domain in combination with consecutive numbering of the unknowns. More generally, if a domain in an n-dimensional space is divided into (possibly overlapping) subdomains with boundaries which do not intersect each other, consecutive numbering allows for a physical interpretation. For a visualization of directed overlap, see the right side of Fig. 1.

The advantage of extending the subdomains only in one direction compared to uniform overlap is that the computational cost of solving the local problems grows slower with the overlap levels.

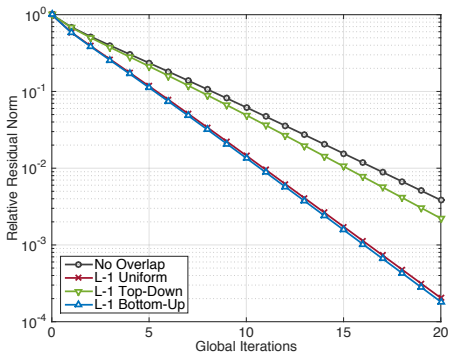

Fig. 3: Convergence of the sequential restricted alternating Schwarz using top-down subdomain scheduling and different overlap strategies. The test matrix is the Laplace problem introduced in the beginning of the section

Figure 3 compares the convergence of block-Jacobi using different restricted Schwarz overlap strategies for a sequential top-down subdomain scheduling. All overlap strategies result in faster convergence than the overlap-free block-Jacobi. However, significant difference in the convergence rate can be observed: the top-down overlap fails to propagate new information, and propagating information from subdomains not yet updated in the global iteration provides only small convergence improvement. The uniform overlap treats information from adjacent unknowns equally, independent of whether the respective subdomain has already been updated in the current iteration or not. The resulting convergence improvement comes at a 6.68 times higher computational cost, as elaborated previously. Using directed overlap pointing bottom-up increases the computational cost only by a factor 3.34. For this test case, the bottom-up overlap provides the same convergence improvement like the uniform overlap. The lower computational cost makes this strategy superior. Although disregarding overlap in direction of "old" neighbors may in general result in a lower convergence rate than uniform overlap, this test validates the expectation that propagating new information provides higher benefits when solving a symmetric problem.

For a random update order, it is impossible to define an overlap direction that propagates information only from already updated subdomains. On average, the effects of using overlap pointing bottom-up and overlap pointing top-down equalize, and the resulting convergence rate is lower than when using uniform overlap, see Fig. 4.

The situation changes as soon as we look into structurally non-symmetric matrices with a directed dependency graph. While propagating information from freshly

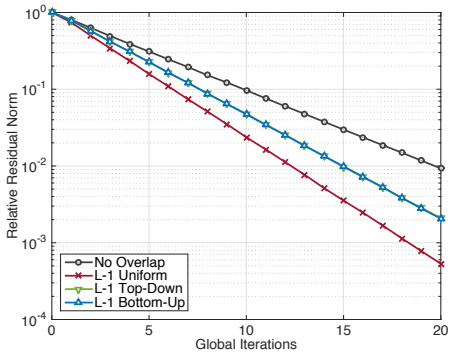

Fig. 4: Convergence of the sequential random-order restricted alternating Schwarz for different overlap strategies. The test matrix is the Laplace problem introduced in the beginning of the section

updated subdomains may still be preferred, the dependencies have a much more important role for convergence. Obviously, these dependencies should also be considered in the subdomain scheduling. Then, it is possible to choose directed overlap that benefits twofold: information gets propagated in dependency directions; and this information comes from subdomains that have already been updated in the current iteration.

For a non-symmetric matrix, it is impossible to always find a subdomain update order that obeys all dependencies. A scenario where this is possible, however, is the solution of sparse triangular systems and sequential component updates. The resulting algorithm is nothing other than forward and backward substitution. Subdomains containing multiple unknowns are available if components are independent and only depend on previously updated components. The strategy of identifying these subdomains and updating the components in parallel is known as level scheduling [29]. Unfortunately, the subdomains are often too small to allow for efficient parallel execution. Also, the components are usually not adjacent in the iteration vector, which results in expensive memory access patterns. But also when using subdomains coming from a decomposition of the iteration vector, it is often possible to identify an overall dependency direction. Some dependencies might be violated, but aligning the subdomain scheduling to the triangular system's dependency direction usually results in faster convergence [3].

In the scenario of random subdomain scheduling, the idea of orienting overlap opposite the update order fails, but overlap may still be adjusted to the dependency graph. Bottom-up overlap propagates information in dependency direction for a lower triangular system, top-down overlap obeys the dependencies for an upper triangular system.

In Fig. 5, the convergence of the different overlap strategies is compared for a lower and an upper sparse triangular system. The systems arise as incomplete LU factorization without fill-in (ILU(0)) for the previously considered Laplace problem. On each subdomain, two Jacobi sweeps are used to solve the local problem.

The sequential subdomain updates are scheduled in a random order, and the results are averaged over 100 runs. Despite ignoring the benefits of orienting the overlap towards already updated subdomains, the convergence of the restricted alternating Schwarz using directed overlap opposite the dependency direction matches the convergence of uniform overlap. Propagating information opposite the dependency direction does not provide noticeable benefits compared to the non-overlapping block-iterative solver.

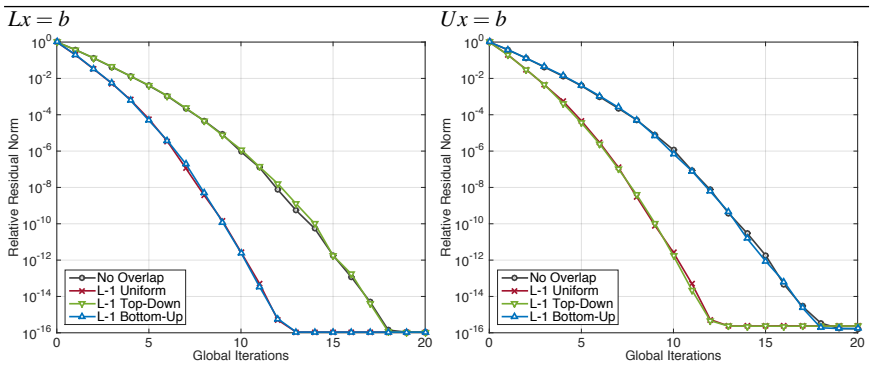

Fig. 5: Convergence of the sequential random-order restricted alternating Schwarz using 2 Jacobi sweeps as local solver on the subdomains. The test cases are the sparse triangular systems arising as incomplete LU factorization without fill-in (ILU(0)) of the Laplacian problem

4 Restricted Overlap on GPUs

We now want to evaluate whether using restricted Schwarz overlap can improve iterative sparse triangular solves on GPUs. The hardware characteristics however require some modification of the approach suggested in Sect. 3. On GPUs, operation execution is realized via a grid of thread blocks, where the distinct thread blocks apply the same kernel to different data [24]. The distinct thread blocks all have the same "thread block size", which is the number of compute threads in one thread block. Multiple thread blocks can be executed in concurrent fashion. The number of thread blocks handled in parallel depends on the thread block size, requested shared memory, and the characteristics of the used GPU hardware. This setting suggests to assign each thread block to one subdomain. If the subdomain size matches the thread block size, all the unknowns in a subdomain can be handled in parallel by the distinct compute threads, which is the desired setting for using Jacobi sweeps as local solver. For non-overlapping subdomains of equal size, this mapping works fine. Extending the subdomains with algebraic Schwarz overlap however becomes difficult, as the overlap for distinct subdomains may differ in size. Fixing the thread block

size to the largest subdomain results in idle threads assigned to smaller subdomains; smaller thread block sizes fail to realize the Jacobi sweeps in parallel fashion. An additional challenge comes from the solution of the local problem being based on local memory. On GPUs, this is the shared memory of the distinct multiprocessors. Due to the limited size, it is impossible to keep the complete iteration vector in shared memory, but the unknowns of the local problem (original subdomain and overlap) have to be stored in consecutive fashion. As these components are in general not adjacent in the global iteration vector, an additional mapping is required. This increases pressure on the memory bandwidth, the typically performance-limiting instance in this algorithm. Also, the scattered access to the components of the algebraic Schwarz overlap in the global iteration vector results in expensive memory reads [24].

Given this background, we relax the mathematical validity of the overlap in favor of higher execution efficiency. Precisely, replace the algebraic Schwarz overlap with "block-overlap". For a given decomposition of the iteration vector, block overlap is generated by extending the subdomains in size such that the subdomains adjacent in the iteration vector overlap. Similar to the algebraic Schwarz overlap, unknowns can be part of multiple subdomains. However, it also is possible that not all components in a subdomain are connected in the dependency graph. The restricted Schwarz setting avoids not only write conflicts, but also ensures that structurally disconnected overlap is not written back to the global iteration vector. Compared to the algebraic Schwarz overlap, two drawbacks can be identified:

- Block overlap can miss important dependencies if the respective unknowns are not included in the block-extension of the subdomain.
- Components part of the block overlap but not structurally connected to the original subdomain increase the cost of the local solver, but do not contribute to the global approximation.

These handicaps become relevant for matrices with significant entries distant to the main diagonal. For matrices where most entries are reasonably close to the diagonal, the higher execution efficiency on GPUs may outweigh the drawbacks. Sparse triangular systems as they arise in the context of approximate incomplete factorization preconditioning often have an Reverse Cuthill-McKee (RCM) ordering, as this ordering helps in producing accurate incomplete factorization preconditioners [16, 10]. At the same time, RCM ordering reduces the matrix bandwidth, which makes block overlap more attractive (more matrix entries captured in the overlap regions).

To match the thread block size of the non-overlapping block-iterative solver, we shrink the original subdomains, and fill up the thread block with overlap. We note that shrinking the original subdomain size and restricting the writes requires a higher number of subdomains for covering the iteration vector. The corresponding higher number of thread blocks reflects the increased computational cost when using block overlap. If subdomains adjacent in the iteration vector are scheduled to the same multiprocessor, overlapping subdomains allows for temporal and spacial cache reuse [14]. The data is loaded into the fast multiprocessor memory only once, and can then be reused for the overlap of an adjacent subdomain. Figure 6 visualizes

the strategy of mapping thread blocks to subdomains for the case of non-overlapping subdomains *(left)*, and the case of directed bottom-up overlap *(right)* , respectively. Note that in the latter, the overlap threads of each thread block only read the data for the local problem (**r**), but do not write back the local solution to the global iteration vector.

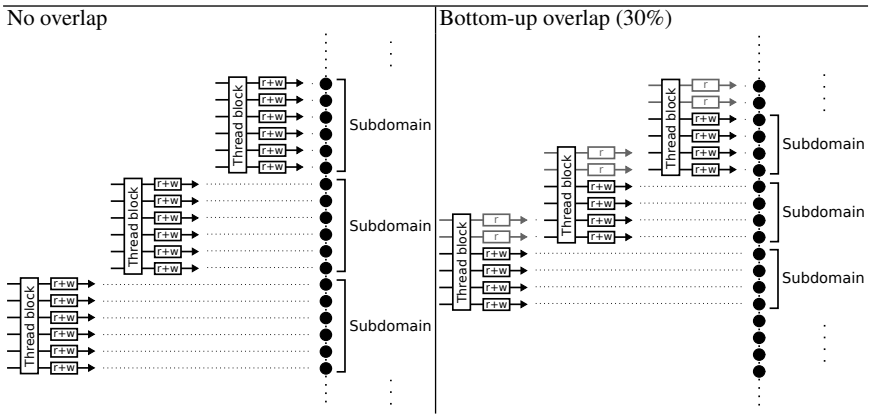

Fig. 6: Mapping thread blocks to the subdomains in case of non-overlapping subdomains *(left)* and bottom-up overlap *(right)* . In the latter case, only the threads at the lower end of each thread block write back the solution update (**r+w**), the threads at the upper end only read the data in for the local problem (**r**)

On GPUs, multiple thread blocks are executed in concurrent fashion. For large problems there however exist more subdomains than can be handled in parallel. We call this case an "overdecomposition": Not all subdomains are updated simultaneously, and the update order impacts the convergence of the block-iterative method. Unfortunately, GPUs generally do not allow insight or modifications to the thread block execution order. However, backward-engineering experiments reveal that for the used GPU architecture, the thread blocks are usually scheduled in consecutive increasing order [14]. For sparse triangular solves, this property can be exploited to improve the information propagation by numbering the thread blocks in dependency direction [3]. The fact that this scheduling order can not be guaranteed gives the solver a block-asynchronous flavor, and requires to report all experimental results as average over multiple runs. As the block overlap is mathematically inconsistent with algebraic Schwarz overlap, we avoid the term "restricted additive Schwarz", but refer to the implementation as "block-asynchronous Jacobi with restricted overlap".

5 Experimental Results

5.1 Test environment

The experimental results were obtained using a Tesla K40 GPU (Kepler microarchitecture) with a theoretical peak performance of 1,682 GFlop/s (double precision). The 12 GB of GPU main memory, accessed at a theoretical bandwidth of 288 GB/s, was sufficiently large to hold all the matrices and all the vectors needed in the iteration process. Although all operations are handled by the accelerator, we mention for completeness that the host was being an Intel Xeon E5 processor (Sandy Bridge). The implementation of all GPU kernels is realized in CUDA [25], version 7.0 [26], using a thread block size of 256. For non-overlapping subdomains, this thread block size corresponds to the size of the subdomains; for overlapping subdomains the size is split into subdomain and overlap. Double precision computations were used.

For the experimental evaluation, we consider solving with the incomplete factorizations of different test matrices, including all problems tested in [3] to show the potential of iterative sparse triangular solves. The test matrices are general sparse matrices from the University of Florida matrix collection (UFMC), and a finite difference discretization of the 3D Laplace problem with Dirichlet boundary conditions. For the discretization, a 27pt stencil on a $64 \times 64 \times 64$ mesh is used, resulting in structurally and numerically symmetric matrix. We consider all matrices in RCM ordering to reduce the matrix profile, as this increases the effectiveness of overlapping matrix rows that are nearby, as well as the effectiveness of incomplete factorizations. Table 1 lists the characteristics of the lower triangular matrices from the incomplete factors. The sparsity plots of these triangular matrices are shown in Fig. 7. We report all experimental results as the average over 100 runs to account for nondeterministic scheduling effects on the GPU in the block-asynchronous Jacobi solver.

Table 1: Characteristics of the sparse lower triangular ILU(0) factors employed in the experimental tests

	Matrix	Description	Size n	Nonzeros n_z	n_z/n	Condition number
UFMC	BCSSTK38	Stiffness matrix, airplane engine component	8,032	116,774	14.54	6.87e+08
	CHP	Convective thermal flow (FEM)	20,082	150,616	7.50	7.90e+05
	CONSPH	Concentric spheres (FEM)	83,334	1,032,267	12.39	6.81e+06
	DC	Circuit simulation matrix	116,835	441,781	3.78	6.54e+10
	M_T1	Structural problem	97,578	4,269,276	43.74	4.78e+10
	STO	3D electro-physical duodenum model	213,360	1,660,005	7.78	1.38e+07
	VEN	Unstructured 2D Euler solver (FEM)	62,424	890,108	14.26	1.85e+07
	LAP	3D Laplace problem (27-pt stencil)	262,144	3,560,572	13.58	9.23e+06

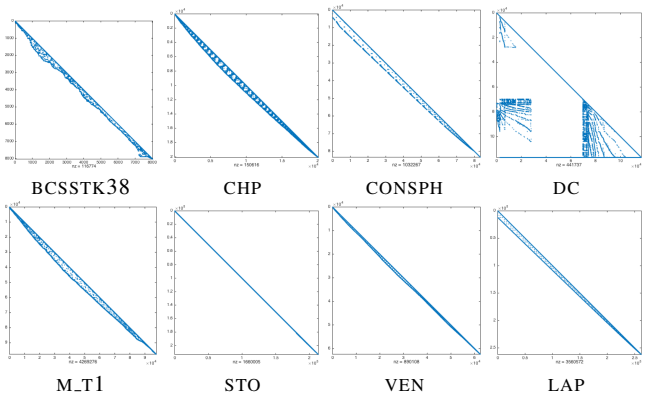

Fig. 7: Sparsity plots of the sparse lower triangular factors listed in Table 1.

5.2 Sparse triangular solves

Figures 8 and 9 show convergence and timing results for the lower sparse triangular factors coming from incomplete factorizations of the selected UFMC matrices and the Laplace matrix. The figures on the left side show convergence of the residual norm with respect to iterations, while the figures on the right side relate the residual norm to the execution time. The GPU thread block scheduling was set to promote top-down subdomain scheduling for information propagation in the dependency direction. In addition, bottom-up overlaps of different sizes were used, which also accounts for the dependency direction of lower triangular matrices. The notation used in the figures relates the size of the overlap to the thread block size, i.e., 25% overlap means that each thread block of size 256 contains a subdomain of size 192 and 64 overlap components. For 25% overlap, the number of thread blocks necessary to cover the complete iteration vector increases by one third compared to a non-overlapping decomposition using a subdomains size of 256. The computational cost, i.e., the number of thread blocks being scheduled, increases by the same factor.

We first make some overall observations. From Fig. 8 and Fig. 9, we observe a range of behaviors for the different problems. In most of the cases, the use of overlap improves the convergence rate. The exceptions are the STO and VEN problems, for which there is very little effect due to overlap, and M_T1 where overlap can actually make convergence worse. For the problems where overlap improves convergence rate, there is still the question of whether or not computation time is improved, since overlap increases the amount of work. The best timings may come from a small or a moderate amount of overlap (rather than a large amount of overlap), balancing the extra computational effort with improved convergence rate.

For the CHP problem, a small convergence improvement can be achieved by using overlap, and this improvement grows, as expected, with the size of the overlap. However, when considering the GPU execution time, overlap is always worse than

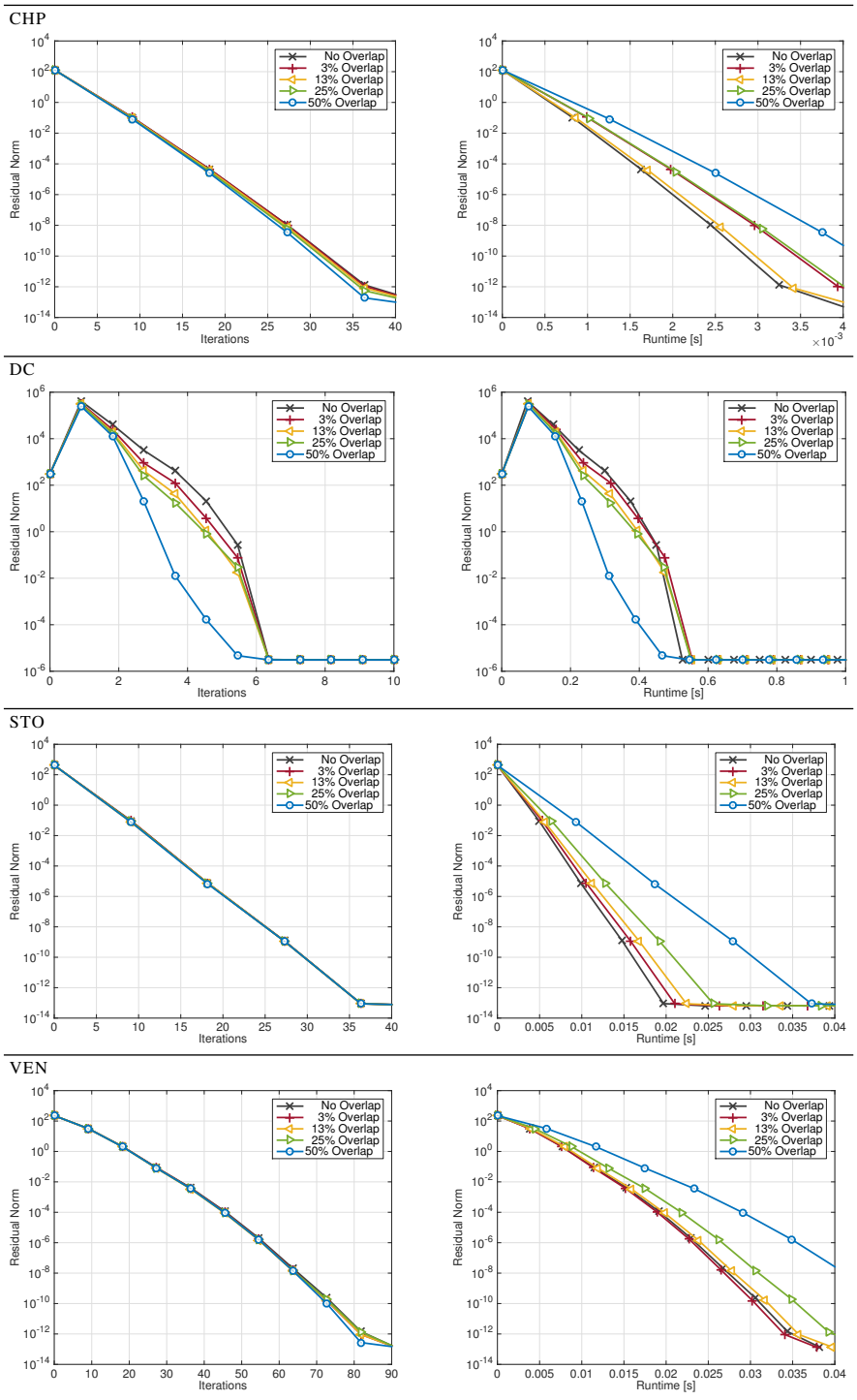

Fig. 8: Convergence *(left)* and residual-performance *(right)* of the block-asynchronous Jacobi using 2 Jacobi sweeps as local solver on the subdomains. The test cases are the sparse lower triangular systems arising as incomplete LU factorization without fill-in (ILU(0)) for the UFMC problems

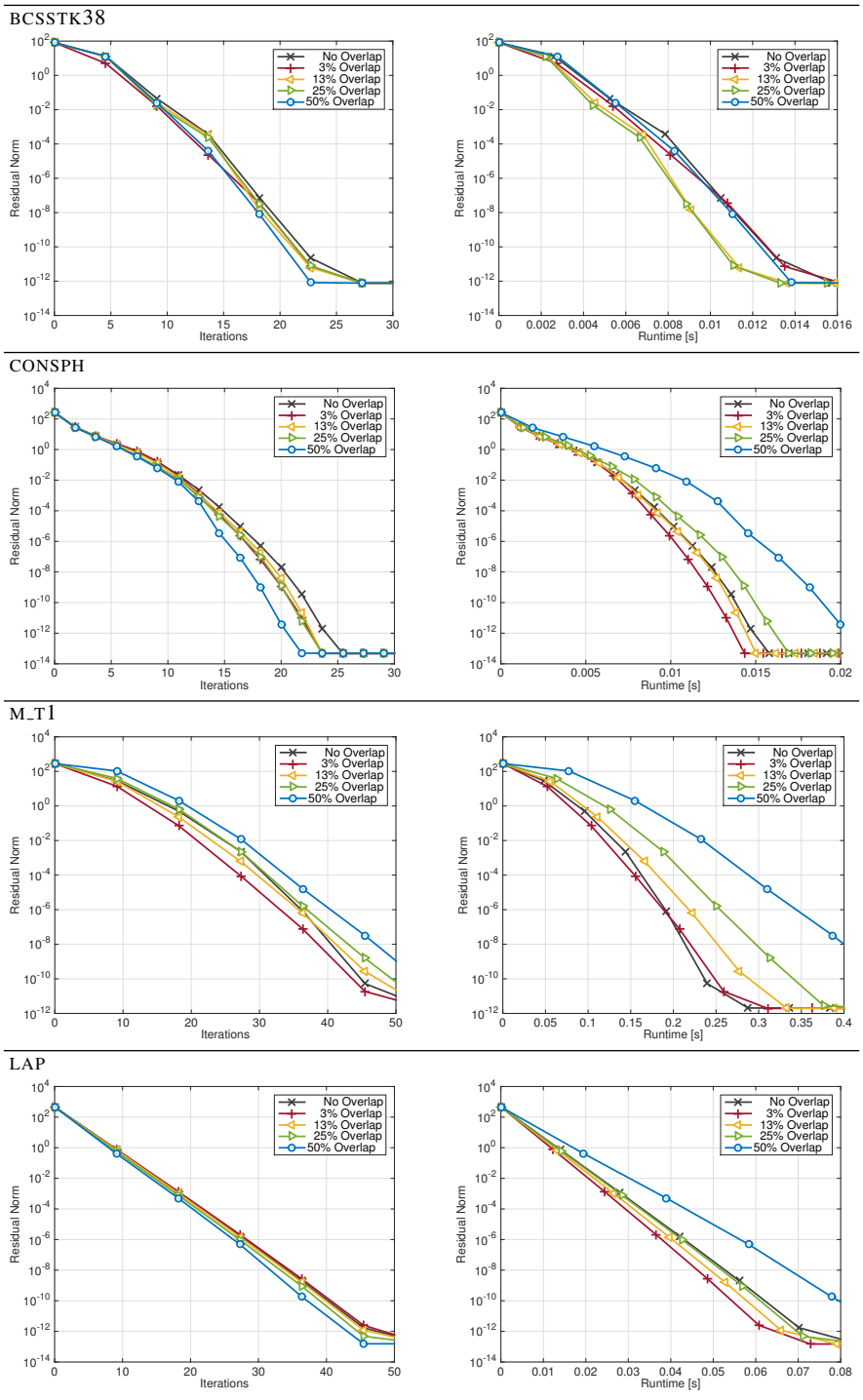

Fig. 9: Convergence *(left)* and residual-performance *(right)* of the block-asynchronous Jacobi using 2 Jacobi sweeps as local solver on the subdomains. The test cases are the sparse lower triangular systems arising as incomplete LU factorization without fill-in (ILU(0)) for the UFMC problems and the LAP problem

non-overlap for this problem. On the other hand, for the DC problem, overlap can improve convergence rate significantly. In addition, overlap does not significantly increase computational cost, as this matrix is very sparse compared to the other test matrices.

For the STO and VEN problems, overlap makes little or no improvement to convergence, as already mentioned. The STO matrix is a large matrix compared to the others, and overlap of adjacent rows in the matrix may introduce few additional couplings in the dependency direction, i.e., many off-diagonal entries are too far from the main diagonal for block overlap to include them. For these large matrices, a decomposition into physical subdomains would be better.

For the BCSSTK38 problem, overlap accelerates the overall solution process. The best results are achieved for 13% and 25% overlap. Overlap is beneficial also for the CONSPH problem, but the convergence improvement (left side) is not reflected in the time-to-solution metric. Only moderate overlap (see results for 3% and 13%, respectively) accelerate the solution process. From the figures, we see that for most problems, 50% overlap has the worst execution time, but the best execution time is given by 3–25% overlap.

The M_T1 problem is an example for which overlap degrades the convergence rate. For this problem, the matrix has a block structure that is easily captured by non-overlapping subdomains, but this structure is not well-matched by overlapping subdomains. Again for this problem, a decomposition into physical subdomains would be better.

Finally, for the LAP problem, convergence can be improved by using overlap. Again, the best solver is not the one using the most overlap, but the one using 3% overlap. This is a typical pattern for block overlap: moderate overlap helps in faster information propagation, but large overlap includes too many structurally disconnected components that increase the computational cost, but do not aid in accelerating convergence.

6 Summary and Future Work

We investigate the potential of enhancing block-iterative methods with restricted Schwarz overlap. For systems carrying a nonsymmetric dependency structure, pointing the overlap opposite the dependencies propagates the information in dependency direction. This improves the convergence rate. We propose a GPU implementation where we relax the consistency to algebraic Schwarz overlap in favor of higher execution efficiency. For sparse triangular factors arising as incomplete LU factors we analyze the convergence and performance benefits achieved by enhancing block-asynchronous Jacobi with directed overlap of different size. Depending on the matrix structure, restricted overlap can improve time-to-solution performance.

In the future, we will look into optimizing the block overlap used in the GPU implementation to the matrix structure. Adapting the overlap size to the location of the most significant off-diagonal entries improves convergence at moderate cost

increase. This optimization makes the block overlap more similar to algebraic Schwarz overlap, and will in particular work for problems with a finite element origin.

Acknowledgments

This material is based upon work supported by the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research, Applied Mathematics program under Award Numbers DE-SC-0012538 and DE-SC-0010042. Daniel B. Szyld was supported in part by the U.S. National Science Foundation under grant DMS-1418882. Support from NVIDIA is also gratefully acknowledged.

References

1. Alvarado, F.L., Schreiber, R.: Optimal parallel solution of sparse triangular systems. SIAM Journal on Scientific Computing 14, 446–460 (1993)
2. Anderson, E.C., Saad, Y.: Solving sparse triangular systems on parallel computers. Internat. Journal High Speed Computing 1, 73–96 (1989)
3. Anzt, H., Chow, E., Dongarra, J.: Iterative sparse triangular solves for preconditioning. In: Träff, J.L., Hunold, S., Versaci, F. (eds.) Euro-Par 2015: Parallel Processing, Lecture Notes in Computer Science, vol. 9233, pp. 650–661. Springer Berlin Heidelberg (2015)
4. Anzt, H., Luszczek, P., Dongarra, J., Heuveline, V.: GPU-Accelerated Asynchronous Error Correction for Mixed Precision Iterative Refinement. In: Euro-Par. pp. 908–919 (2012)
5. Anzt, H., Tomov, S., Dongarra, J., Heuveline, V.: A block-asynchronous relaxation method for graphics processing units. Journal on Parallel Distributed Computing 73(12), 1613–1626 (2013)
6. Anzt, H., Tomov, S., Gates, M., Dongarra, J., Heuveline, V.: Block-asynchronous Multigrid Smoothers for GPU-accelerated Systems. In: Ali, H.H., Shi, Y., Khazanchi, D., Lees, M., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS. Procedia Computer Science, vol. 9, pp. 7–16. Elsevier (2012)
7. Anzt, H.: Asynchronous and Multiprecision Linear Solvers - Scalable and Fault-Tolerant Numerics for Energy Efficient High Performance Computing. Ph.D. thesis, Karlsruhe Institute of Technology, Institute for Applied and Numerical Mathematics (2012)
8. Anzt, H., Dongarra, J., Quintana-Ortí, E.S.: Tuning stationary iterative solvers for fault resilience. In: Proceedings of the 6th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems. pp. 1:1–1:8. ScalA '15, ACM, New York, NY, USA (2015)
9. Benzi, M., Frommer, A., Nabben, R., Szyld, D.B.: Algebraic theory of multiplicative Schwarz methods. Numerische Mathematik 89, 605–639 (2001)
10. Benzi, M., Szyld, D.B., van Duin, A.: Orderings for incomplete factorization preconditionings of nonsymmetric problems. SIAM Journal on Scientific Computing 20, 1652–1670 (1999)
11. Benzi, M., Tůma, M.: A comparative study of sparse approximate inverse preconditioners. Applied Numerical Mathematics 30, 305–340 (1999)
12. Cai, X.C., Sarkis, M.: A restricted additive Schwarz preconditioner for general sparse linear systems. SIAM Journal on Scientific Computing 21, 792–797 (1999)
13. Chazan, D., Miranker, W.: Chaotic Relaxation. Linear Algebra and Its Applications 2(7), 199–222 (1969)

14. Chow, E., Anzt, H., Dongarra, J.: Asynchronous iterative algorithm for computing incomplete factorizations on GPUs. In: Kunkel, J., Ludwig, T. (eds.) Proceedings of 30th International Conference, ISC High Performance 2015, Lecture Notes in Computer Science, Vol. 9137. pp. 1–16 (2015)
15. Chow, E., Patel, A.: Fine-grained parallel incomplete LU factorization. SIAM Journal on Scientific Computing 37, C169–C193 (2015)
16. Duff, I.S., Meurant, G.A.: The effect of ordering on preconditioned conjugate gradients. BIT 29(4), 635–657 (1989)
17. Duin, A.C.N.V.: Scalable parallel preconditioning with the sparse approximate inverse of triangular matrices. SIAM Journal on Matrix Analysis and Applications 20, 987–1006 (1996)
18. Frommer, A., Schwandt, H., Szyld, D.B.: Asynchronous weighted additive Schwarz methods. Electronic Transactions on Numerical Analysis 5, 48–61 (1997)
19. Frommer, A., Szyld, D.B.: On asynchronous iterations. Journal of Computational and Applied Mathematics 123, 201–216 (2000)
20. Frommer, A., Szyld, D.B.: An algebraic convergence theory for restricted additive Schwarz methods using weighted max norms. SIAM Journal on Numerical Analysis 39, 463–479 (2001)
21. Hammond, S.W., Schreiber, R.: Efficient ICCG on a shared memory multiprocessor. Inter. Journal of High Speed Computing 4, 1–21 (1992)
22. Mayer, J.: Parallel algorithms for solving linear systems with sparse triangular matrices. Computing 86(4), 291–312 (2009)
23. Naumov, M.: Parallel solution of sparse triangular linear systems in the preconditioned iterative methods on the GPU. Tech. Rep. NVR-2011-001, NVIDIA (2011)
24. NVIDIA Corporation: CUDA C best practices guide. http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/
25. NVIDIA Corporation: NVIDIA CUDA Compute Unified Device Architecture Programming Guide, 2.3.1 edn. (2009)
26. NVIDIA Corporation: NVIDIA CUDA TOOLKIT V7.0 (2015)
27. Pothen, A., Alvarado, F.: A fast reordering algorithm for parallel sparse triangular solution. SIAM Journal on Scientific and Statistical Computing 13, 645–653 (1992)
28. Saad, Y.: Iterative Methods for Sparse Linear Systems. SIAM, Philadelphia, PA, USA (2003)
29. Saad, Y., Zhang, J.: Bilum: Block versions of multi-elimination and multi-level ilu preconditioner for general sparse linear systems. SIAM Journal on Scientific Computing 20, 2103–2121 (1997)
30. Saltz, J.H.: Aggregation methods for solving sparse triangular systems on multiprocessors. SIAM Journal on Scientific and Statistical Computing 11, 123–144. (1990)
31. Shang, Y.: A parallel finite element variational multiscale method based on fully overlapping domain decomposition for incompressible flows. Numerical Methods for Partial Differential Equations 31, 856–875 (2015)
32. Smith, B.F., Bjørstad, P.E., Gropp, W.D.: Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations. Cambridge University Press, New York, NY, USA (1996)
33. Szyld, D.B.: Different models of parallel asynchronous iterations with overlapping blocks. Computational and Applied Mathematics 17, 101–115 (1998)
34. Toselli, A., Widlund, O.B.: Domain Decomposition Methods - Algorithms and Theory, Springer Series in Computational Mathematics, vol. 34. Springer Berlin Heidelberg (2005)