Universität zu Köln

Center for Data and Simulation Science

Technical Report Series
Center for Data and Simulation Science

Alexander Heinlein, Axel Klawonn, Sivasankaran Rajamanickam, Oliver Rheinbach

FROSch: A Fast And Robust Overlapping Schwarz Domain Decomposition Preconditioner Based on Xpetra in Trilinos

Technical Report ID: CDS-2018-9
Available at http://kups.ub.uni-koeln.de/id/eprint/9018
Submitted on November 16, 2018

FROSch: A Fast And Robust Overlapping Schwarz Domain Decomposition Preconditioner Based on Xpetra in Trilinos

Alexander Heinlein[1,2], Axel Klawonn[1,2], Sivasankaran Rajamanickam[3], Oliver Rheinbach[4]

1 Introduction

This article describes a parallel implementation of a two-level overlapping Schwarz preconditioner with GDSW (Generalized Dryja–Smith–Widlund) coarse space described in previous work [9, 7, 12] into the Trilinos framework; cf. [13]. The software is a significant improvement of a previous implementation [9] ; see Sec. 4 for results on the improved performance.

In the software, now named `FROSch` (Fast and Robust Overlapping Schwarz), efforts were made for the seamless integration into the open-source Trilinos framework, and to allow the use of heterogeneous architectures, such as NVIDIA accelerators. These goals were achieved in the following way:

1. The GDSW preconditioner, i.e., the `FROSch` library, is now part of Trilinos as a subpackage of the package `ShyLU`. The `ShyLU` package provides distributed-memory parallel domain decomposition solvers, and node-level direct solvers for the subdomains. Currently, `ShyLU` has two other domain decomposition solvers, i.e., a Schur complement solver [14] and an implementation of the BDDC method by Clark Dohrmann, and node-level (in)complete `LU` factorizations (`basker`, `fastilu`) , Cholesky factorization (`tacho`) and triangular solves (`hts`).
2. `FROSch` now supports the `Kokkos` programming model through the use of `Tpetra` stack in Trilinos. The `FROSch` library can therefore profit from the

[1] Department of Mathematics and Computer Science, University of Cologne, Weyertal 86-90, 50931 Köln, Germany. E-mail: {alexander.heinlein,axel.klawonn}@uni-koeln.de.
[2] Center for Data and Simulation Science, University of Cologne, Germany, url: http://www.cds.uni-koeln.de
[3] Center for Computing Research, Scalable Algorithms Department, Sandia National Laboratories, Albuquerque, NM 87123. e-mail: srajama@sandia.gov
[4] Institut für Numerische Mathematik und Optimierung, Technische Universität Bergakademie Freiberg, Akademiestr. 6, 09599 Freiberg, Germany. E-mail: oliver.rheinbach@math.tu-freiberg.de.

efforts of the Kokkos package to obtain performance portability by template meta-programming, on modern hybrid architectures with accelerators.

During this process the GDSW code has been modified and improved significantly. The resulting FROSch library is now designed such that different types of Schwarz operators can be added and combined more easily. Consequently, various different Schwarz preconditioners can be constructed using the FROSch framework.

Recently, FROSch has been used in a three-level GDSW implementation [10, 11] and for the solution of incompressible fluid flow problems [8].

2 The GDSW Preconditioner

We are concerned with finding the solution of a sparse linear system

$$Ax = b, \tag{1}$$

arising from a finite element discretization with finite element space $V = V^h(\Omega)$ of an elliptic problem, such as, a Laplace problem, on a domain $\Omega \subset \mathbb{R}^d$, $d = 2,3$, with sufficient Dirichlet boundary conditions.

The GDSW preconditioner [2, 3] is a two-level additive overlapping Schwarz preconditioner with exact local solvers (cf. [16]) using a coarse space constructed from energy-minimizing functions. It is meant to be used in combination with the Krylov methods from the packages Belos [1] or AztecOO.

In particular, let Ω be decomposed into N nonoverlapping subdomains Ω_i, $i = 1,...,N$, and overlapping sudomains Ω_i', $i = 1,...,N$, respectively, and $V_i = V^h(\Omega_i')$, $i = 1,...,N$, be the corresponding local finite element spaces. Further, we define standard restriction operators $R_i : V \to V_i$, $i = 1,...,N$, from the global to the local finite element spaces. Then, the Schwarz operator of the GDSW method can be written in the form

$$P_{\text{GDSW}} = M_{\text{GDSW}}^{-1} A = \Phi A_0^{-1} \Phi^T A + \sum_{i=1}^{N} R_i^T A_i^{-1} R_i A, \tag{2}$$

where $A_0 = \Phi^T A \Phi$ is the coarse space matrix, and the matrices $A_i = R_i A R_i^T$, $i = 1,...,N$, represent the overlapping local problems; cf. [3]. The matrix Φ is the essential ingredient of the GDSW preconditioner. It is composed of coarse space functions which are discrete harmonic extensions from the interface to the interior degrees of freedom of nonoverlapping subdomains. The values on the interface are typically chosen as restrictions of the elements of the null space of the operator $\hat{A}$ to the edges, vertices, and faces of the decomposition, where $\hat{A}$ is the Neumann operator corresponding to the operator A in (1). Therefore, for a scalar elliptic problem, the coarse basis functions form a partition of unity on all subdomains that do not touch the Dirichlet boundary.

The condition number of the GDSW Schwarz operator is bounded as

$$\kappa(P_{\mathrm{GDSW}}) \leq C\left(1+\frac{H}{\delta}\right)\left(1+\log\left(\frac{H}{h}\right)\right)^2 \tag{3}$$

where h is the size of a finite element, H the size of a nonoverlapping subdomain, and δ the width of the overlap; see [2, 3, 4].

However, the dimension of the standard GDSW coarse space is in the order of $\dim(V_0) = \mathscr{O}(\dim(\mathrm{null}(\hat{A}))(N_{\mathscr{V}} + N_{\mathscr{E}} + N_{\mathscr{F}}))$, where $N_{\mathscr{V}}$, $N_{\mathscr{E}}$, and $N_{\mathscr{F}}$ are the global numbers of vertices, edges, and faces of the nonoverlapping domain decomposition, respectively. The dimension of the coarse space is fairly high. Therefore, GDSW coarse spaces of reduced dimension have very recently been introduced in [5]; see also [12] for parallel results. For general problems, the dimension of the reduced GDSW coarse spaces is $\dim(V_0) = \mathscr{O}(\dim(\mathrm{null}(\hat{A}))(N_{\mathscr{V}}))$, which is, especially for unstructured decompositions, significantly smaller. Both types of GDSW coarse spaces are implemented in `FROSch`, and in Sec. 4, we present performance results.

3 Software Design of the `FROSch` Library

During the integration of the `FROSch` library into Trilinos, the code was substantially restructured. In particular, in the transition from the Trilinos `Epetra` (used in [9]) to the newer `Xpetra` sparse matrix infrastructure, it was extended to a framework of Schwarz preconditioners. Additionally, parts of the code have been improved and functionality has been added. As opposed to [9], `FROSch` is completely based on `Xpetra`.

A Framework for Schwarz Preconditioners As described in Sec. 2, the GDSW preconditioner is a two-level overlapping Schwarz method using a specific coarse space. The GDSW Schwarz operator is of the form

$$P_{2-\mathrm{Lvl}} = \underbrace{\Phi A_0^{-1} \Phi^T A}_{P_0} + \sum_{i=1}^{N} \underbrace{R_i^T \tilde{A}_i^{-1} R_i A}_{P_i};$$

cf. (2); and therefore, it is the sum of local overlapping Schwarz operators P_i, $i = 1, ..., N$, and a global coarse Schwarz operator P_0. There are different ways to compose Schwarz operators P_i, $i = 0, ..., N$, e.g.:

Additive: $P_{\mathrm{ad}} = \sum_{i=0}^{N} P_i$

Multiplicative: $P_{\mathrm{mu}} = I - (I-P_N)(I-P_{N-1})\cdots(I-P_0)$

$P_{\mathrm{mu-sym}} = I - (I-P_0)\cdots(I-P_{N-1})(I-P_N)(I-P_{N-1})\cdots(I-P_0)$

Hybrid: $P_{\mathrm{hy-1}} = I - (I-P_0)\left(I - \sum_{i=0}^{N} P_i\right)(I-P_0)$

$P_{\mathrm{hy-2}} = \alpha P_0 + I - (I-P_N)\cdots(I-P_1);$

cf. [16]. Using the `FROSch` library, it is very simple to construct the different variants once the ingredients have been set up.

Let us explain this based on the example of the class `GDSWPreconditioner` in `FROSch`, which is derived from the abstract class `SchwarzPreconditioner` and contains an implementation of the GDSW preconditioner: in `FROSch`, the `SumOperator` is used to combine Schwarz operators in an additive way. The additive first level is implemented in the class `AlgebraicOverlappingOperator` and the coarse level of the GDSW preconditioner in the class `GDSWCoarseOperator`. Therefore, the `GDSWPreconditioner` is basically just the following composition of Schwarz operators:

```
GDSWPreconditioner = SumOperator( AlgebraicOverlappingOperator,
                                  GDSWCoarseOperator )
```

By replacing the `SumOperator` by a `ProductOperator`, the levels can be coupled in a multiplicative way. The different classes for Schwarz operators are all derived from an abstract `SchwarzOperator`, and the classes `SchwarzOperator` and `SchwarzPreconditioner` are both derived from the abstract `Xpetra::Operator`.

Transition from `Epetra` to `Xpetra` To facilitate the use of `FROSch` on novel architectures, the code was ported completely from `Epetra` data structures to `Xpetra`. As `Xpetra` provides a lightweight interface to `Epetra` as well as `Tpetra`, `FROSch` can now profit from the computational kernels from `Kokkos`, while maintaining compatibility to older `Epetra`-based software such as LifeV [6].

Improvement of the Code & Additional Functionality The efficiency of the code was improved and new functionality was added as part of this redesign. In particular, the routines for the computation of local-to-global mappings and the identification of the interface components have been rewritten and therefore improved with respect to their performance; see Sec. 4 for the numerical results.

Two important features have been added. First, we have introduced the possibility to reconstruct a domain decomposition interface algebraically based on a unique distribution of the degrees of freedom into subdomains and the nonzero pattern of the matrix. This works particularly well for scalar elliptic problems and piecewise linear elements. In general, the best performance is obtained when a `RepeatedMap` is provided by the user; cf. Fig. 1. This map corresponds to the nonoverlapping domain decomposition and is replicated in the interface degrees of freedom. Secondly, we have introduced a function that identifies Dirichlet boundary conditions based on the matrix entries. This is important because the coarse basis functions are zero on the Dirichlet boundary.

User Interface The user-interface of the `FROSch` library has been completely re-designed. Compared to the previous implementation, where the setup of the preconditioner was split into the first and second level, it is now split into the phases `initialize` and `compute`, also reducing the number of required lines of code to construct the GDSW preconditioner; cf. Fig. 1.

In the `initialize` phase, all data structure that corresponds to the structure of the problem is built, i.e., the index sets of overlapping subdomains and the interface are identified and the interface values of the GDSW coarse space are computed. In

Previous implementation from [9]:

```
Teuchos::RCP<SOS::SOS> M_SOS(new SOS::SOS(numVectors,numSubdomainsPerProcess,
     M_DomainMap,M_RangeMap));
Teuchos::RCP<SOS::SOSSetUp> M_SOSSetUp(new SOS::SOSSetUp(
     numSubdomainsPerProcess,dimension,dofs,M_rowMatrixTeuchos,M_DomainMap));

M_SOSSetUp->FirstLevel(M_ProcessMapOverlap);

M_SOSSetUp->SecondLevel(M_ProcessMapNodes,M_ProcessMap,SOS::LifeVOrdering,
     M_LocalDirichletBoundaryDofs,"Mumps",useRotations,M_LocalNodeList);

M_SOSSetUp->SetUpPreconditioner(M_SOS,"Mumps",secondLevelSolverParameterList,
     Type);
```

Current implementation `Shylu/FROSch`:

```
Teuchos::RCP<FROSch::GDSWPreconditioner<SC,LO,GO,NO> > FROSchGDSW(new FROSch::
     GDSWPreconditioner<SC,LO,GO,NO>(A,ParameterList);
FROSchGDSW->initialize(Dimension,Overlap,RepeatedMap);
FROSchGDSW->compute();
```

Fig. 1 Comparison of the user-interface for the previous implementation of the GDSW solver (top) and the current implementation in `FROSch` (bottom). The setup is split into the `initialize` and `compute` phases instead of the two levels.

the `compute` phase, all computations that are related to the values of the matrix A are performed, i.e., the overlapping problems are factorized, the interior values of the GDSW coarse basis functions are computed, and the coarse problem is assembled and factorized.

Therefore, the `initialize` and `compute` phases can be seen as the symbolic and the numerical factorization of a direct solver: if only the the values in the matrix A change, the preconditioner can be updated using `compute`, and if the structure of the problem is changed, `initialize` has to be called to update the preconditioner.

Also, `FROSch` provides a `Stratimikos` interface to be easily used from applications; `Stratimikos` provides a unified framework for linear solvers and preconditioners in Trilinos.

4 Performance of the New `FROSch` Software

A performance comparison of the new software against the previous implementation is provided here. We consider a Poisson model problem on $\Omega \subset \mathbb{R}^d$, $d = 2,3$, with full Dirichlet boundary condition, discretized by piecewise quadratic finite elements.

We compare the performance of the previous implementation, which is based on `Epetra`, and the current implementation in `FROSch`. In particular, the `Epetra` and the `Tpetra` version of the current implementation, which are both available through the `Xpetra` interface, are compared. As a Krylov-solver GMRES from `Belos` [1] is used with a relative tolerance of 10^{-7} for the unpreconditioned resid-

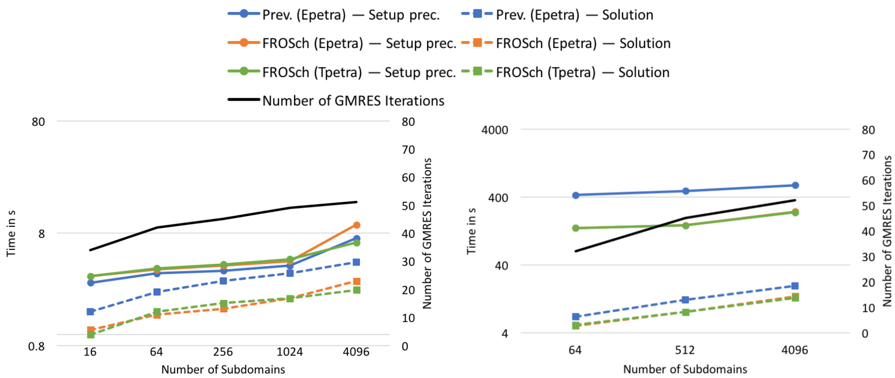

Fig. 2 Weak scalability of the two-level Schwarz preconditioner using the GDSW coarse space for the Poisson model problem: (Left) in two dimensions with overlap $\delta = 5h$ and $H/h = 100$ (approximately 50k degrees of freedom per sudomain); (Right) in two dimensions with overlap $\delta = 2h$ and $H/h = 14$ (approximately 50k degrees of freedom per sudomain). Comparison of the previous implementation (blue) and the current implementation in `FROSch`, i.e., the `Epetra` (orange) and the `Tpetra` (green) versions available through the `Xpetra` interface. The number of iterations (black) are identical for all versions.

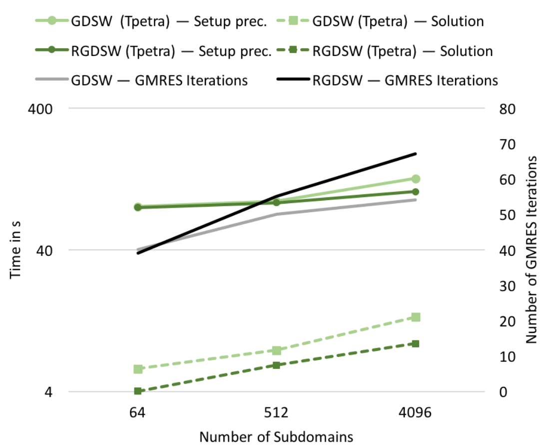

Fig. 3 Weak scalability of the two-level Schwarz preconditioner with overlap $\delta = 1h$ for the Poisson model problem in three dimensions with $H/h = 14$ (approximately 35k degrees of freedom per sudomain): comparison of the GDSW and the RGDSW coarse space using the `Tpetra` version of the `FROSch` implementation.

ual. For the local and coarse problems, the direct solver `KLU` is used; only in Fig. 4, `Mumps` is used as the direct solver. We always use one subdomain per processor core. The computations were performed on the magnitUDE supercomputer at Universität Duisburg-Essen, which has 15k cores (Intel Xeon E5-2650v4, 12C, 2.2GHz) and a total memory of 36 096 GB. Here, we do not exploit any node parallelism when using `Tpetra`.

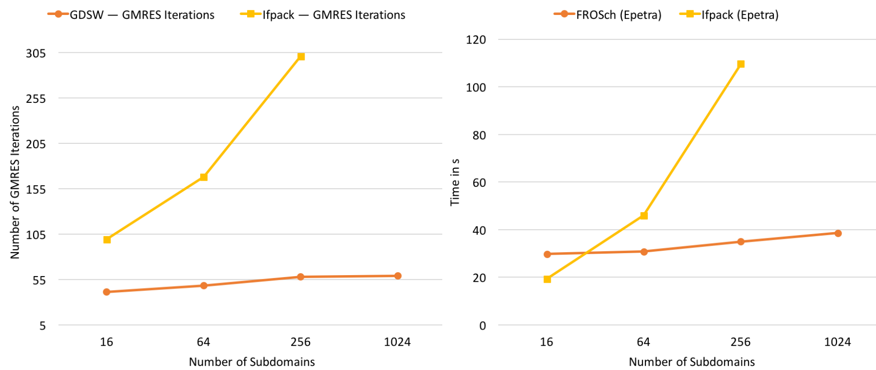

Fig. 4 Weak scalability for the Poisson model problem in two dimensions with $H/h = 200$ (approximately 195k degrees of freedom per sudomain): comparison of `FROSch` using the GDSW coarse space and the one-level overlapping Schwarz preconditioner `Ifpack`; numbers of GMRES iterations (left) and total solver times (right). Using `Mumps` for all direct solves. For 1 024 subdomains, `Ifpack` did not converge within 500 GMRES iterations.

We consider the setup phase and the solution phase and include the identification of the interface components in the setup phase. This part does not scale very well and can takes a significant amount of time for a large number of processes; cf. [9].

In Fig. 2 (left), we present numerical results for the GDSW and the RGDSW (option 1 from [5, 12]) preconditioner, respectively, in two dimensions. We observe that, in the solution phase, the new implementation is always faster than the previous implementation. The time for the setup phase is comparable. The results in Fig. 2 (right), where we compare the preconditioners in three dimensions, are more interesting. Again, we observe that the solution phase is faster by a similar factor. However, in three dimensions, the setup phase in the `FROSch` implementation is much faster compared to the previous implementation. We also observe that the `Tpetra` version is always slightly faster than the `Epetra` version of the new code.

In Fig. 3, the GDSW and the RGDSW coarse spaces are compared for the `Tpetra` version of the `FROSch` implementation. We observe that, due to the increasing dimension of the coarse space, the computation time can be improved when using reduced dimension coarse spaces. This effect becomes stronger when the number of subdomains is increased; cf. [12].

Finally, we present a comparison of `FROSch` using the GDSW coarse space and `Ifpack` [15], i.e., a one-level overlapping Schwarz preconditioner, in Fig. 4. We observe that `Ifpack` does not scale as it lacks a second level. Already for 64 subdomains, `FROSch` converges much faster, and for 1 024 subdomains, `Ifpack` does not converge within a maximum number of 500 GMRES iterations.

Conclusion We presented the new Trilinos library `FROSch` that allows the flexible construction of different overlapping Schwarz methods. The `FROSch` implementation of the GDSW preconditioner is significantly faster than the previous one.

Acknowledgements The authors gratefully acknowledge the computing time granted by the Center for Computational Sciences and Simulation (CCSS) at Universität Duisburg-Essen and provided on the supercomputer magnitUDE

(DFG grants INST 20876/209-1 FUGG, INST 20876/243-1 FUGG) at Zentrum für Informations- und Mediendienste (ZIM). Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.

References

1. E. Bavier, M. Hoemmen, S. Rajamanickam, and H. Thornquist. Amesos2 and belos: Direct and iterative solvers for large sparse linear systems. Scientific Programming, 20(3):241–255, 2012.
2. C. R. Dohrmann, A. Klawonn, and O. B. Widlund. Domain decomposition for less regular subdomains: overlapping Schwarz in two dimensions. SIAM J. Numer. Anal., 46(4):2153–2168, 2008.
3. C. R. Dohrmann, A. Klawonn, and O. B. Widlund. A family of energy minimizing coarse spaces for overlapping Schwarz preconditioners. In Domain decomposition methods in science and engineering XVII, volume 60 of LNCSE, pages 247–254. Springer, Berlin, 2008.
4. C. R. Dohrmann and O. B. Widlund. Hybrid domain decomposition algorithms for compressible and almost incompressible elasticity. IJNME, 82(2):157–183, 2010.
5. C. R. Dohrmann and O. B. Widlund. On the design of small coarse spaces for domain decomposition algorithms. SIAM J. Sci. Comput., 39(4):A1466–A1488, 2017.
6. L. Formaggia, M. Fernandez, A. Gauthier, J. F. Gerbeau, C. Prud'homme, and A. Veneziani. The LifeV Project. Web.
7. A. Heinlein. Parallel Overlapping Schwarz Preconditioners and Multiscale Discretizations with Applications to Fluid-Structure Interaction and Highly Heterogeneous Problems. PhD thesis, Universität zu Köln, Germany, 2016.
8. A. Heinlein, C. Hochmuth, and A. Klawonn. Monolithic overlapping schwarz domain decomposition methods with GDSW coarse spaces for incompressible fluid flow problems. Technical report, 2018. Submitted for publication to SIAM J. Sci. Comput.
9. A. Heinlein, A. Klawonn, and O. Rheinbach. A parallel implementation of a two-level overlapping Schwarz method with energy-minimizing coarse space based on Trilinos. SIAM J. Sci. Comput., 38(6):C713–C747, 2016.
10. A. Heinlein, A. Klawonn, O. Rheinbach, and F. Röver. A three-level extension of the GDSW overlapping schwarz preconditioner in two dimensions. Technical report, March 2018. Submitted for publication to LNCSE.
11. A. Heinlein, A. Klawonn, O. Rheinbach, and F. Röver. A three-level extension of the GDSW overlapping schwarz preconditioner in three dimensions. Technical report, November 2018. Submitted for publication to LNCSE.
12. A. Heinlein, A. Klawonn, O. Rheinbach, and O. Widlund. Improving the parallel performance of overlapping Schwarz methods by using a smaller energy minimizing coarse space. 2017. Accepted for publication to the Proceedings of the 24rd International Conference on Domain Decomposition Methods, Springer LNCSE.
13. M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley. An overview of the Trilinos project. ACM Trans. Math. Softw., 31(3):397–423, 2005.
14. S. Rajamanickam, E. G. Boman, and M. A. Heroux. ShyLU: A hybrid-hybrid solver for multicore platforms. In 2012 IEEE 26th International Parallel and Distributed Processing Symposium, pages 631–643, May 2012.
15. M. Sala and M. Heroux. Robust algebraic preconditioners with IFPACK 3.0. Technical Report SAND-0662, Sandia National Laboratories, 2005.
16. A. Toselli and O. Widlund. Domain decomposition methods—algorithms and theory, volume 34 of Springer Series in Computational Mathematics. Springer-Verlag, Berlin, 2005.