SAND2018-13025R

FROSCH: A FAST AND ROBUST OVERLAPPING SCHWARZ DOMAIN DECOMPOSITION PRECONDITIONER BASED ON XPETRA IN TRILINOS[*]

ALEXANDER HEINLEIN[*], AXEL KLAWONN[*], SIVASANKARAN RAJAMANICKAM[†], AND OLIVER RHEINBACH[‡]

Abstract. A parallel two-level overlapping Schwarz domain decomposition preconditioner has been integrated into the Trilinos ShyLU-package. The preconditioner uses an energy-minimizing coarse space and can be constructed from an assembled sparse matrix. The software implements variants of the two-level overlapping Schwarz method from [Dohrmann, Klawonn, Widlund, SINUM 2008], where it was denoted Generalized Dryja, Smith, Widlund (GDSW). The implementation is based on [Heinlein, Klawonn, Rheinbach, SISC 2016] but has been improved significantly with respect to efficiency, generality, e.g., for the use of Tpetra instead of Epetra matrices, and its interface.

1. Introduction.

A parallel implementation of a two-level overlapping Schwarz preconditioner with GDSW (Generalized Dryja Smith Widlund) coarse space described in [7, 6, 8] has been integrated into the software library Trilinos; cf. [9]. The software is based on a previous implementation [7], which has been improved significantly; see also section 4 for the improved performance.

The software is now called `FROSch` (Fast and Robust Overlapping Schwarz). Efforts were made

1. for the seamless integration into the open-source Trilinos framework at Sandia National Laboratories
2. and to allow the efficient use of heterogeneous architectures making use of, e.g., 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 `ShyLU`. Currently, `ShyLU` contains also two other domain decomposition solvers, i.e., a Schur complement solver and an implementation of the BDDC method by Clark Dohrmann, and the node-level solvers `basker`, `fastilu`, `hts`, and `tacho`.
2. `FROSch` now supports the `Kokkos` programming model though the use of `Tpetra` matrices. The `FROSch` library can therefore profit from the efforts of the `Kokkos` package to obtain performance portability by template meta-programming, also 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. This will be described in this report.

2. The GDSW preconditioner.

We are concerned with finding the solution

[1]Mathematisches Institut, Universität zu Köln, Weyertal 86-90, 50931 Köln, Germany. e-mail: {`alexander.heinlein,axel.klawonn`}`@uni-koeln.de`

[2]Center for Computing Research, Scalable Algorithms Department, Sandia National Laboratories, Albuquerque, NM 87123. e-mail: `srajama@sandia.gov`

[3]Institut für Numerische Mathematik und Optimierung, Fakultät für Mathematik und Informatik, Technische Universität Bergakademie Freiberg, Akademiestr. 6, 09596 Freiberg, Germany. e-mail: `oliver.rheinbach@math.tu-freiberg.de`

[*]Submitted to the editors xx.xx.xx.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & 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-NA0003525.

of a sparse linear system

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

arising from a finite element discretization 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. [10]) using a coarse space constructed from energy-minimizing functions. It is meant to be used in combination with the Krylov methods from `Belos` [1]. The corresponding Schwarz operator can be written in the form

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

where $A_0 = \Phi^T A \Phi$ is the coarse space matrix, and the matrices $\tilde{A}_i = R_i K R_i^T$, $i = 1, ..., N$, 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 nullspace of the operator to the edges, vertices, and faces of the decomposition. Therefore, for a scalar elliptic problem, the coarse basis functions form a partition of unity on the whole domain Ω.

However, the dimension of the coarse space is in the order of

$$\dim(V_0) = \mathcal{O}(\dim(\text{null}(\hat{A}))(N_{\mathcal{V}} + N_{\mathcal{E}} + N_{\mathcal{F}})), \tag{3}$$

where $N_{\mathcal{V}}$, $N_{\mathcal{E}}$, $N_{\mathcal{F}}$ are the global numbers of vertices, edges, and faces, respectively, and $\hat{A}$ is the Neumann operator corresponding to the operator A in (1). The dimension of the coarse space is fairly high.

Therefore, GDSW coarse spaces of reduced dimension have very recently been introduced in [4]. For general problems, the dimension of the reduced GDSW coarse spaces is

$$\dim(V_0) = \mathcal{O}(\dim(\text{null}(\hat{A}))(N_{\mathcal{V}})), \tag{4}$$

which is, especially for unstructured decompositions, significantly lower than (3).

Both coarse types of GDSW coarse spaces are implemented in `FROSch` and in section 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, it was extended to a whole framework for Schwarz preconditioners, the code was transitioned from the package `Epetra` to `Xpetra`, and a new user interface was implemented.

In addition to that, some parts of the code have been improved and some functionality has been added to the code.

3.1. A Framework for Schwarz Preconditioners. The GDSW preconditioner is a two-level overlapping Schwarz method using a specific coarse space.

The standard two-level additive Schwarz operator reads

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

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

We will 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 construction 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 the abstract class `SchwarzOperator`, and the classes `SchwarzOperator` and `SchwarzPreconditioner` are both derived from the abstract `Xpetra::Operator`. As opposed to [7], `FROSch` is completely based on `Xpetra`.

3.2. 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 [5].

3.3. 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.

Two important features have been added. First, we have introduced the possibility to reconstruct a domain decomposition interface algebraically based on a unique

Previous implementation from [7]:

```
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>(K,ParameterList);

FROSchGDSW->initialize(Dimension,OL,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.*

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. Secondly, we have introduced a function that identifies Dirichlet boundary conditions based on the matrix entries. This is important because the nodes on the Dirichlet boundary are treated as interior nodes.

3.4. 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 up into the first and the 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. Figure 1.

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

4. Performance of the New `FROSch` Software. A performance comparison of the new software against the previous implementation is provided here. We consider

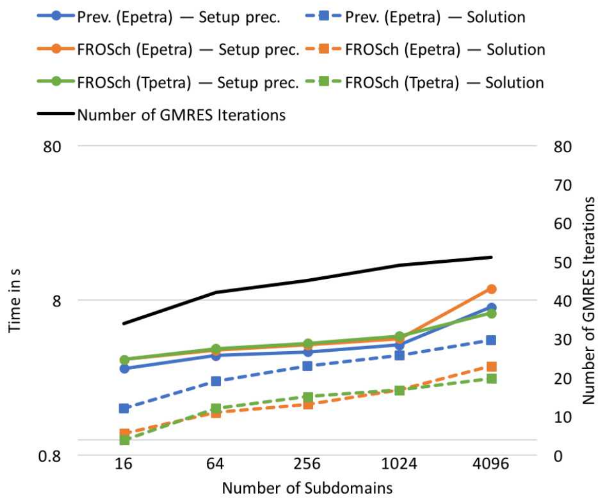

FIG. 2. *Weak scalability of the two-level Schwarz preconditioner with overlap $\delta = 5h$ and GDSW coarse space for model problem (5) in two dimensions with $H/h = 100$ (approximately 50k degrees of freedom per sudomain): comparison of the previous implementation (blue) and the current implementation in FROSch, i.e., the Epetra (orange) and the Tpetra (green) versions available through the Xpetra interface. The numbers of iterations (black) are exactly the same for all versions.*

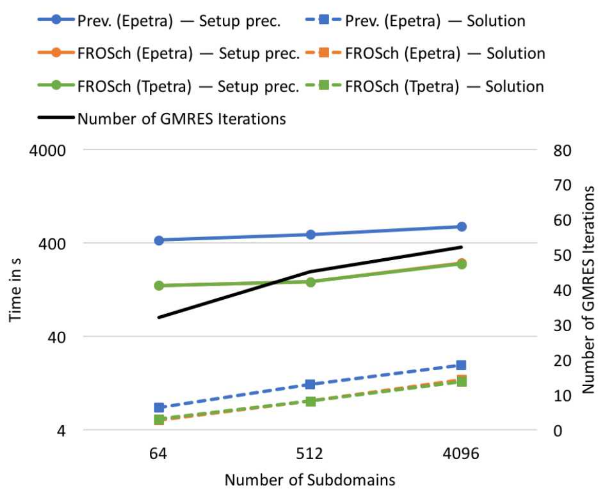

FIG. 3. *Weak scalability of the two-level Schwarz preconditioner with overlap $\delta = 2h$ and GDSW coarse space for model problem (5) in three dimensions with $H/h = 14$ (approximately 50k degrees of freedom per sudomain): comparison of the previous implementation (blue) and the current implementation in FROSch, i.e., the Epetra (orange) and the Tpetra (green) versions available through the Xpetra interface. The numbers of iterations (black) are exactly the same for all versions. The lines for the Epetra (orange) and the Tpetra (green) versions of FROSch overlap.*

a Laplace model problem on $\Omega \subset \mathbb{R}^d$, with $d = 2, 3$,

$$\begin{aligned} -\Delta u &= 1 \quad \text{in } \Omega, \\ u &= 0 \quad \text{on } \partial\Omega, \end{aligned} \tag{5}$$

discretized by piecewise quadratic finite elements.

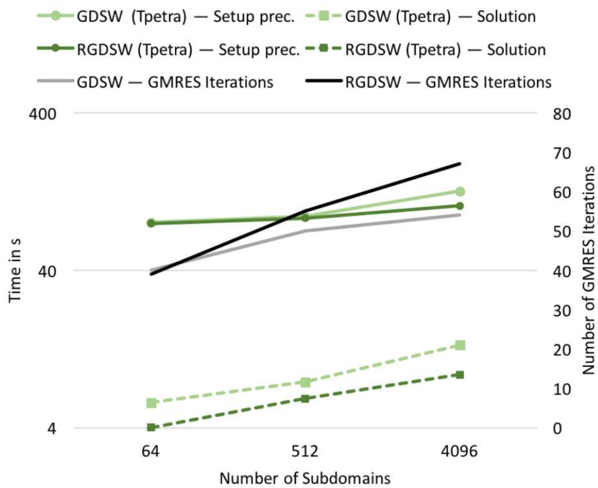

FIG. 4. *Weak scalability of the two-level Schwarz preconditioner with overlap $\delta = 1h$ for model problem (5) in three dimensions with $H/h = 14$ (approximately 35k degrees of freedom per sudomain): comparison of the GDSW and the RGDSW coarse space using the* `Tpetra` *version of the* `FROSch` *implementation.*

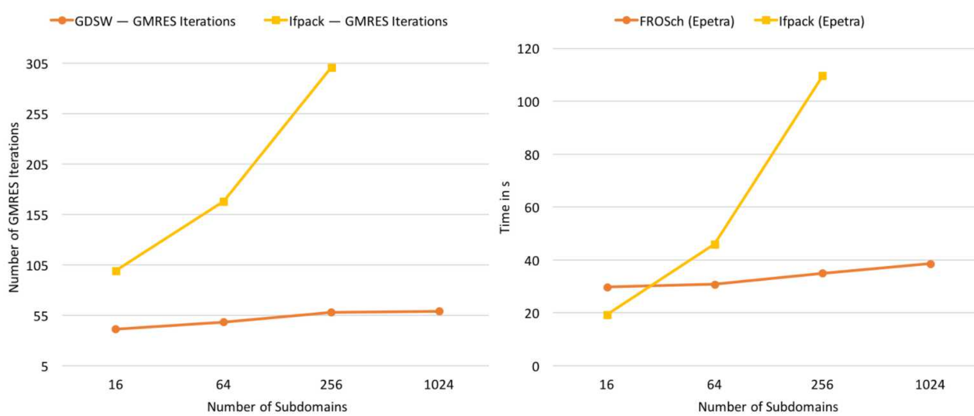

FIG. 5. *Weak scalability for model problem (5) in two dimensions with $H/h = 200$ (approximately 195k degrees of freedom per sudomain): comparison of* `FROSch` *using the GDSW coarse space and the one-level overlapping Schwarz preconditioner* `Ifpack`*; numbers of GMRES iterations (left) and total solver times (right). Using* `Mumps` *for all direct solves. For 1 024 subdomains,* `Ifpack` *did not converge within 500 GMRES iterations.*

In all tests, the performance of the previous implementation, which is based on `Epetra`, and the current implementation in `FROSch` is compared. In particular, two versions of the current implementation, the `Epetra` and the `Tpetra` version, are compared. Both are available through the `Xpetra` interface. As a Krylov-solver GMRES is used with a relative tolerance of 10^{-7} for the unpreconditioned residual. For the local and coarse problems, the direct solver KLU is used; only in Figure 5, `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.

We consider the setup phase and the solution phase. Note that we also include the identification of the interface components in the setup phase. This part does not scale well and can take a significant amount of time for a large number of processes; cf. [7].

In Figure 2, we present numerical results for the GDSW preconditioner and the RGDSW preconditioner (option 1 from [4, 8]), 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.

More interesting are the results in Figure 3, where we compare the preconditioners in three dimensions. 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 Figure 4, the GDSW and the reduced GDSW (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 coarse spaces. This effect becomes stronger when the number of subdomains is increased; cf. [8].

Finally, we present a comparison of `FROSch` using the GDSW coarse space and `Ifpack`, i.e., a one-level overlapping Schwarz preconditioner, in Figure 5. We observe that `Ifpack` does not scale. Already for 64 subdomains, the `FROSch` converges much faster, and for 1 024 subdomains, `Ifpack` does not converge within a maximum number of 500 GMRES iterations.

5. Conclusion. The solver package `FROSch`, which is a complete framework for Schwarz preconditioners, has been integrated into Trilinos. It is based on the package `Xpetra`, such that it is compatible with `Epetra` and `Tpetra`. Its performance has improved with respect to the previous implementation and the usability of the code has been significantly improved.

6. 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 (2012), pp. 241–255.

[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 (2008), pp. 2153–2168.

[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, vol. 60 of Lect. Notes Comput. Sci. Eng., Springer, Berlin, 2008, pp. 247–254.

[4] C. R. Dohrmann and O. B. Widlund, On the design of small coarse spaces for domain decomposition algorithms, Tech. Report 2017-987, Department of Computer Science, Courant Institute of Mathematical Sciences, New York University, January 2017.

[5] L. Formaggia, M. Fernandez, A. Gauthier, J. F. Gerbeau, C. Prud'homme, and A. Veneziani, The LifeV Project. Web, http://www.lifev.org.

[6] 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.

[7] 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 (2016), pp. C713–C747, https://doi.org/10.1137/16M1062843, http://dx.doi.org/10.1137/16M1062843.

[8] 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. Submitted to the Proceedings of the 24rd International Conference on Domain Decomposition Methods, Springer Lect. Notes Comput. Sci. Eng.

[9] 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 (2005), pp. 397–423, https://doi.org/http://doi.acm.org/10.1145/1089014.1089021.

[10] A. Toselli and O. Widlund, Domain decomposition methods—algorithms and theory, vol. 34 of Springer Series in Computational Mathematics, Springer-Verlag, Berlin, 2005.