77

Domain Decomposition and Multilevel Methods in Diffpack

Are Magnus Bruaset, Hans Petter Langtangen, and

Gerhard W. Zumbusch

1 Introduction

Looking back a decade or two, the computing power commonly available to scientists and engineers has grown at an amazing rate. Following this race for megaflops, the scientific community has developed a taste for solving mathematical problems of increasing levels of complexity. Naturally, this trend calls for sophisticated numerical methods that are capable of solving the problems in question in an efficient, yet reliable, way.

Since the bottleneck of many scientific applications turns out to be the numerical solution of linear or nonlinear systems of equations, this field has been subject to intensive research over the years. In particular, much attention has been paid to domain decomposition and multigrid methods, which have proven to be highly efficient strategies for many types of applications. Although the performance of such methods has been theoretically analyzed, extensive numerical experimentation is usually required in more complicated applications in order to obtain the best possible results. From this point of view, there is a need for software environments with genuine support for these types of numerical experiments, giving the user access to different algorithmic scenarios at the press of a button.

Domain decomposition and multilevel methods contain a variety of more standard numerical building blocks (linear solvers, matrix assembly, interpolation of fields, etc.). Successful software for complicated applications must offer the user a flexible run-time combination of all these different components. The purpose of the present paper is to describe how one can achieve such flexible software. In particular, we present a unified framework for domain decomposition and multilevel methods, and show how this framework can be efficiently implemented in existing software packages for PDEs[1].

The unified framework about to be presented is in part well known from the analysis

1 The software design discussed in this paper has been implemented and verified using the object-oriented PDE library Diffpack [Dif, BL97].

Ninth International Conference on Domain Decomposition Methods
Editor Petter E. Bjørstad, Magne S. Espedal and David E. Keyes

of overlapping and non-overlapping methods [DW90], as well as from theory for overlapping and multilevel schemes [Xu92]. In this context, the goal of this paper is to extend the known framework to cover even more methods in common use, especially some Schur complement and nonlinear schemes. We will formulate the framework in a novel way that encourages systematic implementation of a wide class of domain decomposition and multilevel methods. Finally, we report on the experiences gathered from a particular implementation in the Diffpack software.

2 The Unified Framework

Abstract Schwarz Method

We consider linear systems, $Ax = f$, where A arises from the discretization of a partial differential operator. The prototype solution algorithm, the additive Schwarz method, can be written as

$$B \ = \ \sum_j R_j^* \, B_j \, R_j \tag{2.1}$$

with approximate solvers B_j operating on a subspace V_j and restrictions R_j and adjoint interpolations R_j^*. In particular we consider the following methods and its variants (see also [SBG96]), which all can be written in the framework:

multilevel iteration	additive, multiplicative, nonlinear
overlapping Schwarz	additive, multiplicative, nonlinear with, without coarse grid
Schur complement iteration	exact, inexact local solves
Schur complement preconditioner	Neumann-Neumann, wirebasket with, without coarse grid

The interface basically consists of

- transfer or restriction operators such as R_j and R_j^*,
- (approximate) subspace solvers B_j,
- evaluation of residuals $(f - A_j x)$ on a subdomain (optional).

Here, A_j is the discrete operator on V_j. The appropriate solver B_j is normally chosen as a traditional linear or nonlinear method, whereas the "transfer" R_j is usually implemented via sparse matrices or difference stencils. However, also algorithmic implementations of R_j and message passing in a parallel environment are possible. Software components for B_j, R_j, R_j^* and $f - A_j x$ are normally found in a package for PDEs.

Multilevel Methods

Standard multigrid algorithms fit into the frame outlined in 2. In the linear multigrid context the abstraction of the (not necessarily adjoint) grid transfer R_j, R_j^* and the local pre- and post-smoothers B_j, B_j^* is sometimes referred to as abstract multigrid.

The implementation follows equation (4.1.1) in Hackbusch [Hac85]. Using multigrid as a preconditioner implies the initial guess $x = 0$.

However, it is also possible to include nonlinear multigrid with nonlinear smoothers B_j, B_j^* [Zum96a]. Using equation (9.3.3) in [Hac85], we can implement the nonlinear FAS scheme and the nested nonlinear multigrid version by Hackbusch. The pre- and post-smoothers are now nonlinear iterative solvers.

This abstract multilevel approach can be used for different variants of multigrid. The particular algorithm depends on the initialization and implementation of the smoothers B_j, the transfer operators R_j, and the operators A_j. There may be non-nested, non-matching, or adaptively refined grids, operators defined by Galerkin products or operator dependent transfers, or algebraic multigrid transfers and operators.

Schur Complement Methods

We decompose the stiffness matrix into one part related to the coupling interface c and several independent parts related to the interior nodes of the subdomains $j = 1, \dots, m$.

$$A = \left(\begin{array}{cccc|c} A_{11} & & & & A_{1c} \\ & A_{22} & & & A_{2c} \\ & & \ddots & & \vdots \\ \hline A_{c1} & A_{c2} & \dots & & A_{cc} \end{array}\right)$$

The Schur complement is defined by $S = A_{cc} - \sum_j A_{cj} A_{jj}^{-1} A_{jc}$.

In the case we solve subdomain problems exactly or accurate enough, or we are able to compute S itself rather than the action of S, the equation system reduces to a system $S\, x_c = f_c$ for the unknowns on the interface x_c.

Neumann-Neumann Preconditioner for the Schur Complement. Preconditioning of the interface system with a Neumann-Neumann algorithm [BGLV89] looks like equation 2.1 in the framework of additive Schwarz methods. The operators B_j now solve homogeneous Neumann problems obtained by sub-assembly on a subdomain. The transfer operators R_j and R_j^* copy (scaled) nodal data from the interface x_c to the nodes on the boundary of the subdomain and vice versa. Hence the ordinary additive Schwarz implementation can be used.

A coarse grid can be added to the Neumann-Neumann preconditioner without changes of the algorithm [DW95] just like in the overlapping Schwarz case adding one subspace $j = 0$. Coupling a coarse grid equation in a multiplicative symmetric way instead, called balancing [Man93], requires some extensions resulting in a mixture of the additive and the multiplicative Schwarz algorithm. One can use the standard global-to-local communication pattern and additional residual evaluations to create the multiplicative coupling.

Wire Basket Preconditioner for the Schur Complement. A preconditioner of wirebasket type [Smi91] for the Schur complement may also be written as an additive Schwarz method now for subspaces of the interface consisting of vertices v, single faces f, and single edges e,

$$B_c = R^{v*} B^v R^v + \sum_j R_j^{e*} B_j^e R_j^e + \sum_k R_k^{f*} B_k^f R_k^f$$

The transfer operators R_j^e and R_k^f perform a hierarchical basis transform. The vertex solver B^v can be implemented by a standard coarse grid solver and the edge solvers B_j^e can be substituted by diagonal scaling. However, the face solvers B_k^f have to be implemented as preconditioners for an interface problem covering both adjacent subdomains.

The two-dimensional analog BPS has a similar structure. It can be implemented by leaving out the faces and choosing some local interface preconditioner for the edge solvers B_j^e [BPS86].

Inexact Schur Complement. Introducing approximate local Dirichlet solvers B_j in the computation of the Schur complement $S \approx A_{cc} - \sum_j A_{cj} B_j A_{jc}$, one cannot restrict the computations to $Sx_c = f_c$, but the full system $Ax = f$ must be considered. We seek a preconditioner for the matrix A, which is constructed using a preconditioner B_c for S and preconditioners $\bar{B}_j$ for the local Dirichlet type problems A_{jj} that may differ from B_j used for the Schur complement. However, the inexact solvers B_j in the Schur complement method are still not fully understood theoretically [HLM91].

We write the Schur decomposition in the additive Schwarz framework

$$B \; = \; R_c^* \, B_c \, R_c \; + \; \sum_j R_j^* \, \bar{B}_j \, R_j$$

with restrictions $R_j x = x_j$ and the transformation to the Schur system $R_c x = x_c - \sum_j A_{cj} B_j x_j$.

One can use multilevel and domain decomposition methods or standard iterative solvers for the implementation of the local solvers B_j, B_j^*, $\bar{B}_j$, and the local solvers used in B_c. This approach to the Schur complement can also be used for nonlinear problems in a Picard iteration manner (see [Zum96b]) since the residual for the full system has to be evaluated every outer iteration step anyway. An exact Dirichlet solver B_j leads to zero residuals at interior nodes and is more efficiently implemented computing on the interface directly as in the previous section.

3 Realizing the Framework

As mentioned initially, the abstract view of the different domain decomposition and multilevel methods taken in the description above has been realized in terms of software. More precisely, the described framework is the basis for the implementation of domain decomposition and multilevel algorithms in the Diffpack simulation environment. Diffpack [Dif] consists of a set of object-oriented libraries written in C++, intended to simplify the implementation of PDE solvers [BL97]. The involved libraries contain many useful abstractions that come quite natural to developers of PDE software. For instance, the application programmer has immediate access to high-level abstractions for linear systems, various matrix formats, different solvers and preconditioners, as well as building blocks for finite element and finite difference discretizations.

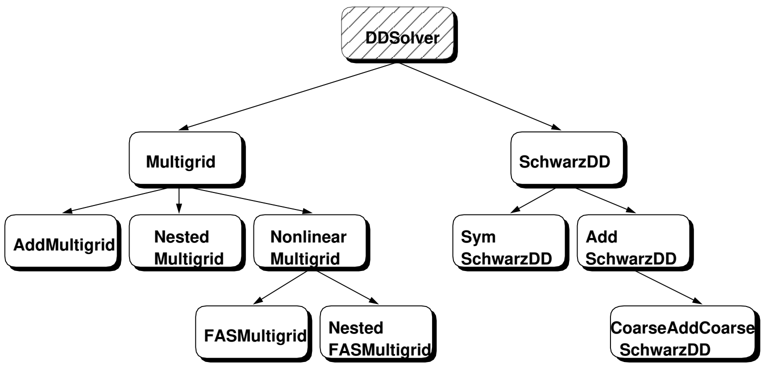
Originally, Diffpack was designed without attention to domain decomposition or multilevel algorithms. Nevertheless, by adopting a unified approach to such methods, one can in principle implement this functionality as an add-on module to existing PDE packages. However, it is then crucial that the framework organizing the different methods can take advantage of tools already present in the available software platform. During the course of the multilevel extension of Diffpack, we have experienced that a clean design of such an add-on module depends heavily on a clean design and modular structure of the underlying libraries. We believe that abstract data types and object-oriented programming are important mechanisms for achieving the necessary degree of modularity. In fact, the Diffpack code for the described framework was realized as a high-level, compact combination of existing C++ classes. We refer to [ABL97, BL97, ABC$^+$97, BL96] and the references therein for information about object-oriented numerics, the efficiency of C++ for scientific computing, the design of Diffpack and examples on Diffpack applications. It should be mentioned that object-oriented implementations of domain decomposition and multilevel strategies have also been addressed by other authors, e.g. as part of the PETSc system [GS94, PET].

Equation solvers in Diffpack utilize abstractions for linear systems, as well as linear and nonlinear solvers. In this context, a linear system consists of a coefficient matrix, a solution vector, a right-hand side and possibly a preconditioner, which will be reused in our framework. The preconditioner can either be a matrix or an *action*. In case of an action, the preconditioner can, e.g., call a linear solver for the same or a related PDE problem. Linear solvers can utilize convergence monitors in order to control the degree of solution accuracy [BL96]. At its present stage of development, Diffpack offers relatively simple nonlinear solvers, like Newton's method or the Picard iteration. This type of algorithm requires the programmer to define and solve a linear (sub)system. Operators can be defined in terms of coefficient matrices arising from finite element assembly. Since a finite element grid is just a C++ object in Diffpack, it is easy to create a hierarchy of grids, and apply toolboxes for finite element schemes and the PDE's definition to create the associated operators and right-hand sides.

The layered, modular design of Diffpack building blocks can be immediately applied to create linear and nonlinear operators, smoothers, transfer operators, residuals and other basic components needed in domain decomposition and multilevel methods, reusing existing code. For instance, the subspace solver B_j in a domain decomposition method may be implemented as a linear solver call or a nonlinear solution procedure, both present in the original package. To allow maximum flexibility, the programmer of the PDE application is responsible for defining B_j, and contrary to the PETSc approach [GS94, PET], we do not *require* a reference to a linear solver object. The main ingredients of the domain decomposition interface are the procedures for grid transfer and for local solvers. Data is only stored for some auxiliary vectors and parameters like the multigrid cycle type (V, W, ...).

The technical details of taking an existing Diffpack application and equipping it with domain decomposition and multilevel methods are described elsewhere [Zum96a, Zum96b]. However, due to the flexibility of the original software components, it turns out to be trivial to run a multigrid solver and experiment with various pre- and post-smoothers (choice of algorithm, number of sweeps, order of unknowns), coarse grid solvers (iterative and direct, grid size), cycle-types, nested iterations, non-matching grids, semi-coarsening, multigrid used as a preconditioner or as a stand-alone solver, different nonlinear versions, grid types and special procedures to initialize operators. For domain decomposition, the type, precision and termination of subdomain solvers, the decomposition of the domain, the type of a coarse grid and coarse-grid solver, and the scaling of transfer operators are of main

Figure 1 Multilevel and Domain Decomposition algorithms. Abstract `DDSolver`, the multiplicative and additive multigrid algorithms including nonlinear versions and additive, multiplicative, symmetric multiplicative and mixed additive/ multiplicative Schwarz algorithms.



interest. Consequently, the resulting software environment satisfies the most important requirement stated in the introduction of this paper; to offer the user genuine support for systematic numerical experiments with sophisticated multilevel strategies.

As previously mentioned, Schur complement methods do not immediately fit into a unified framework. This is also reflected in the pilot implementation. We use an implicit representation of the Schur complement, implemented as an algorithmically defined matrix. This is basically a new type of matrix in Diffpack, realized as a subclass in the existing matrix hierarchy [BL96], which implements the matrix product Sx only. New and old application software can of course work with this matrix type through an abstract (base class) interface. The action Sx is implemented by calling (reusing) subdomain Dirichlet solvers for A_{jj}^{-1} and several matrix multiplications, since the concept of a Schur complement was not present in Diffpack. Direct access to S is then not available. However, standard Schwarz preconditioners can be used for the Schur complement system, reusing even the domain decomposition algorithms.

The domain decomposition and multilevel methods introduced in Diffpack are organized in a class hierarchy with `DDSolver` as base class, see Figure 1. Various specific solution strategies are organized as subclasses of either (multiplicative) `Multigrid` or (alternating) Schwarz domain decomposition (`SchwarzDD`). `Multigrid` uses level to level data transfer ($j \to j+1$), while the `SchwarzDD` algorithms uses local to global data transfer. `SymSchwarzDD` is a symmetric version of the multiplicative `SchwarzDD`, cycling back and forth like the multiplicative `Multigrid`. The `CoarseAddCoarseSchwarzDD` version is used for special treatment of the coarse grid, while the subdomains are treated as in the additive `AddSchwarzDD`. The hierarchy itself is designed to optimize code reuse and reflects software issues, while one can certainly think of modifications in the categorization. The subclasses make use of existing solvers and preconditioners in Diffpack, while still offering the duality of being accessible as new solvers and new preconditioners in the original libraries. For the authors, this experience of playing around with abstractions and extending libraries in ways that were not initially planned for, has been a strong indication that modern programming techniques, such as object-oriented programming, are vital for an accelerated development of scientific computing.

4 Efficiency

We have outlined a flexible software framework for domain decomposition and multilevel methods, where the particular Diffpack implementation is in C++. Many will expect the computational efficiency of such flexible implementations and the use of C++ to be significantly worse than special-purpose Fortran codes tailored at a specific PDE and solution algorithm. To shed some light on this problem we have performed some simple numerical experiments with multigrid methods for a Poisson equation, with smooth variable coefficients, on the unit square. A general Diffpack implementation, also applicable to unstructured grids, was compared to (a) the adaptive PLTMG code [Ban94], (b) a finite difference based example Fortran code with MPI [Dou95a], and (c) a constant 5-point stencil sparse matrix Fortran Poisson solver Madpack5 [Dou95b]. The table below shows CPU times for various problem sizes.

level j size n	4 289	5 1089	6 4225	7 16641	8 66049	9 263169
PLTMG	.43	1.7	5.7	30	140	–
MPI example	.15	.17	.23	.49	1.8	7.6
madpack5	.01	.04	.12	.50	2.8	13
Diffpack	.07	.16	.39	1.5	7.1	24

As we see, there is no indication that the object-oriented implementation style in C++ implies a significant loss of computational efficiency. The reasons for this are simple; object-orientation is only used for high-level administration in Diffpack, whereas CPU-time consuming operations usually take place in low level C/Fortran-style routines that can be highly optimized by today's compiler technology. Moreover, the general finite element software in Diffpack makes use of simplified, optimized algorithms when it is known that the grid is a uniform lattice.

5 Conclusion

We have outlined a unified framework for the whole set of domain decomposition and multilevel methods. The framework has been realized in Diffpack using object-oriented programming techniques. We have indicated that the implementation has the same level of efficiency as tailored, traditional implementations in Fortran or C, but with much more flexibility and extensibility.

REFERENCES

[ABC+97] Arge E., Bruaset A. M., Calvin P. B., Kanney J. F., Langtangen H. P., and Miller C. T. (1997) On the efficiency of C++ for scientific computing. In Dæhlen M. and Tveito A. (eds) *Mathematical Models and Software Tools in Industrial Mathematics*. Birkhäuser.

[ABL97] Arge E., Bruaset A. M., and Langtangen H. P. (1997) Object-oriented numerics. In Dæhlen M. and Tveito A. (eds) *Mathematical Models and Software Tools in Industrial Mathematics*. Birkhäuser.

[Ban94] Bank R. E. (1994) *PLTMG: A Software Package for Solving Elliptic Partial Differential Equations – Users' Guide 7.0*. SIAM Books, Philadelphia.

[BGLV89] Bourgat J. F., Glowinski R., LeTallec P., and Vidrascu M. (1989) Variational formulation and algorithm for trace operator in domain decomposition calculations. In Chan T. F., Glowinski R., Périaux J., and Widlund O. B. (eds) *Proc. Second Int. Conf. on Domain Decomposition Meths.*, pages 3–16. SIAM, Philadelphia.

[BL96] Bruaset A. M. and Langtangen H. P. (1996) Object-oriented design of preconditioned iterative methods. *To appear in ACM Trans. Math. Software* .

[BL97] Bruaset A. M. and Langtangen H. P. (1997) A comprehensive set of tools for solving partial differential equations; Diffpack. In Dæhlen M. and Tveito A. (eds) *Numerical Methods and Software Tools in Industrial Mathematics*. Birkhäuser.

[BPS86] Bramble J. H., Pasciak J. E., and Schatz A. H. (1986) The construction of preconditioners for elliptic problems by substructuring, I. *Math. Comp.* 47: 103–134.

[Dif] Diffpack world wide web home page. http://www.oslo.sintef.no/diffpack/.

[Dou95a] Douglas C. C. (1995) Example multigrid code using MPI. ftp://na.cs.yale.edu/pub/mgnet/www/mgnet/Codes/douglas/.

[Dou95b] Douglas C. C. (1995) Madpack: A family of abstract multigrid or multilevel solvers. *Comput. Appl. Math.* 14: 3–20.

[DW90] Dryja M. and Widlund O. B. (1990) Towards a unified theory of domain decomposition algorithms for elliptic problems. In Chan T. F., Glowinski R., Périaux J., and Widlund O. B. (eds) *Proc. Third Int. Conf. on Domain Decomposition Meths.*, pages 3–21. SIAM, Philadelphia.

[DW95] Dryja M. and Widlund O. B. (1995) Schwarz methods of Neumann-Neumann type for three-dimensional elliptic finite element problems. *Comm. Pure Appl. Math.* 48: 121–155.

[GS94] Gropp W. and Smith B. (1994) Scalable, extensible, and portable numerical libraries. In *Proceedings of Scalable Parallel Libraries Conference*. IEEE, Los Alamitos, CA.

[Hac85] Hackbusch W. (1985) *Multi-Grid Methods and Applications*. Springer, Berlin.

[HLM91] Haase G., Langer U., and Meyer A. (1991) Domain decomposition methods with inexact subdomain solvers. *J. Numer. Lin. Alg. Appl.* 1: 27–41.

[Man93] Mandel J. (1993) Balancing domain decomposition. *Comm. Numer. Meth. Engrg.* 9: 233–241.

[PET] Petsc world wide web home page. http://www.mcs.anl.gov/petsc/petsc.html.

[SBG96] Smith B., Bjørstad P., and Gropp W. (1996) *Domain Decomposition. Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, New York.

[Smi91] Smith B. F. (1991) A domain decomposition algorithm for elliptic problems in three dimensions. *Numer. Math.* 60(2): 210–234.

[Xu92] Xu J. (1992) Iterative methods by space decomposition and subspace correction: A unifying approach. *SIAM Review* 34: 581–613.

[Zum96a] Zumbusch G. W. (1996) Multigrid methods in Diffpack. Technical Report STF42 F96016, SINTEF Applied Mathematics, Oslo.

[Zum96b] Zumbusch G. W. (1996) Overlapping domain decomposition methods in Diffpack/ Schur complement domain decomposition methods in Diffpack. Technical report, SINTEF Applied Mathematics, Oslo.