

A Numerical Study of Some Parallel Algebraic Preconditioners

Xing Cai*

Simula Research Laboratory & University of Oslo
P.O. Box 1080, Blindern, 0316 Oslo, Norway
xingca@simula.no

Masha Sosonkina†

University of Minnesota
Duluth, MN 55812, USA
masha@d.umn.edu

Abstract

We present a numerical study of several parallel algebraic preconditioners, which speed up the convergence of Krylov iterative methods when solving large-scale linear systems. The studied algebraic preconditioners are of two types. The first type includes simple block preconditioners using incomplete factorizations or preconditioned Krylov solvers on the subdomains. The second type is an enhancement of the first type in that Schur complement techniques are added to treat subdomain-interface unknowns. The numerical experiments show that the scalability properties of the preconditioners are highly problem dependent, and a simple Schur complement technique is favorable for achieving good overall efficiency.

1. Introduction

Solving systems of linear equations is a topic of fundamental importance for the numerical solution of partial differential equations (PDEs). Almost all the numerical strategies rely on solving systems of linear equations on the form

$$\mathbf{A}\mathbf{x} = \mathbf{b}. \quad (1)$$

In (1), \mathbf{A} is often referred to as the system matrix, \mathbf{b} is called the right-hand side vector and vector \mathbf{x} contains the unknowns. In this paper, we only consider linear systems that arise from the standard discretization techniques, i.e., finite elements, finite differences and finite volumes. The system matrix \mathbf{A} is sparse in such cases, meaning that the number of nonzeros per row is small. Discretization on a very fine grid will result in a linear system that has a very large number of unknowns. Traditional direct solution methods

for (1), such as Gaussian elimination, will become inapplicable due to high operation counts, large storage demand and high sensitivity to computer round-off errors. Large-scale linear systems are therefore better handled by iterative methods, especially the so-called Krylov subspace solvers; see e.g. [7].

If the grid resolution keeps increasing, the dimension of \mathbf{A} will eventually become so large that one single processor has trouble handling the entire linear system. This makes the use of parallel computers mandatory. Of course, another perhaps more important motivation for adopting parallel computers in solving PDEs is to save time by using more processors. Krylov subspace methods are in fact easily parallelized, because they involve only three types of kernel operations: vector update/addition, inner-product between two vectors and matrix-vector product. Once the unknowns of (1) are partitioned among the processors, see Section 1.1 below, the three types of kernel operations can be parallelized by running local computations on each processor, plus inter-processor communication when necessary.

1.1. Distributed sparse linear systems

A natural way of partitioning the unknowns of (1) is to use the idea of *domain decomposition* [11], which partitions the global solution domain Ω into a set of subdomains. Such a partitioning implies a partitioning of the rows of \mathbf{A} . The grid points that lie on the boundary of a subdomain are called *interdomain interface points*, whereas the other grid points of a subdomain are called *internal points*; see Figure 1. We remark that the solid curves in Figure 1 represent “cutting lines” that divide the grid points disjointly among the subdomains. (Note that there are no grid points lying on the cutting lines.) We also observe that only the interdomain interface points have direct couplings with the external interface points that belong to neighboring subdomains. These interface points need to exchange values by communication during a parallel matrix-vector product.

Here we have an important observation. There is no need to first build up the global linear system (1) by a dis-

*This author has received support from The Research Council of Norway (Programme for Supercomputing) through a grant of computing time.

†The work of this author was supported in part by NSF under grants NSF/ACI-0000443 and NSF/INT-0003274, and in part by the Minnesota Supercomputing Institute.

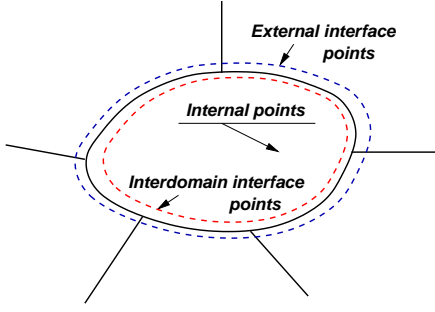


Figure 1. A subdomain that arises from domain decomposition.

cretization on the entire Ω , and then divide the rows of \mathbf{A} among the processors. A better approach is to decompose Ω first and assign one subdomain (and its associated subgrid) to each processor. Thereafter, each processor carries out discretization on its own subdomain to produce the designated rows of \mathbf{A} . These rows constitute a subdomain matrix. This “distributed discretization” approach avoids construction and storage of the global matrices/vectors. So the global linear system (1) only exists logically and we work with a *distributed sparse linear system* instead. Each processor thus handles only sub-matrices and sub-vectors associated with a subdomain.

An “economic” partitioning scheme should divide the grid points evenly and disjointly among the subdomains. Although an external interface point (see Figure 1) is the “responsibility” of a neighboring subdomain, it is beneficial for a subdomain to include all its external interface points into its local data structure. In this way, the rows of \mathbf{A} that correspond to the interdomain interface points can be built without need for communication. This is equivalent to allowing a minimum amount of overlap between neighboring subdomains. In other words, each subdomain includes its external interface points in its local data structure, but is not responsible for finding the solution on these points. More overlap between subdomains is strictly speaking not necessary for parallelizing Krylov subspace methods. However, an increased overlap may help to produce better parallel preconditioner, as is well known for the so-called overlapping domain decomposition methods; see e.g. [11].

1.2. Preconditioner and convergence

It is well known that the difficulty of solving a linear system, which arises from discretizing a PDE, grows as the grid resolution increases. For example, an elliptic PDE typically produces a system matrix \mathbf{A} with a condition number of order $\mathcal{O}(h^{-2})$, where $1/h$ characterizes the grid resolution. The number of iterations needed by a Krylov subspace

solver is typically of order $\mathcal{O}(h)$. To achieve faster convergence, we can use preconditioned iterations. That is, we may, e.g., solve an equivalent system

$$\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b}, \quad (2)$$

where \mathbf{M} is the preconditioner approximating \mathbf{A} in some way; see e.g. [7]. An extra difficulty in parallel solution of PDEs is that a good sequential preconditioner may not function equally well in the parallel setting. It is thus our purpose to study the behavior of the so-called parallel algebraic preconditioners.

2. Parallel algebraic preconditioners

We adopt the framework of a distributed sparse linear system, as has been described in Section 1.1. Each subdomain vector x_i of local unknowns is split into two parts: the sub-vector u_i of internal variables and the sub-vector y_i of interdomain interface variables. The right-hand side vector b_i is split accordingly into two sub-vectors f_i and g_i ,

$$x_i = \begin{pmatrix} u_i \\ y_i \end{pmatrix}; \quad b_i = \begin{pmatrix} f_i \\ g_i \end{pmatrix}. \quad (3)$$

The subdomain matrix A_i residing on processor i is block-partitioned according to this splitting, leading to

$$A_i = \begin{pmatrix} B_i & F_i \\ E_i & C_i \end{pmatrix}. \quad (4)$$

In this setup, the local linear system can be rewritten as:

$$\begin{pmatrix} B_i & F_i \\ E_i & C_i \end{pmatrix} \begin{pmatrix} u_i \\ y_i \end{pmatrix} + \left(\sum_{j \in \mathcal{N}_i}^0 E_{ij} y_j \right) = \begin{pmatrix} f_i \\ g_i \end{pmatrix}, \quad (5)$$

where the term $E_{ij} y_j$ represents the contribution to the local system from the neighboring subdomain j , and \mathcal{N}_i is the set of neighboring subdomains for subdomain i .

Parallel block preconditioners are the simplest algebraic preconditioning strategy, where each subdomain updates its local solution *independently* by solving a subdomain linear system formed by A_i and a given local residual. The subdomain systems can be solved in different ways, e.g., by a local (preconditioned) Krylov solver. Another simpler subdomain solver is to use the backward-forward procedure associated with an incomplete LU-factorization (ILU). We will use, in particular, a zero fill-in ILU (denoted ILU(0)) or a dual-threshold ILU (denoted ILUT); see [7].

Schur complement techniques refer to methods that iterate only on the interdomain unknowns, implicitly using internal unknowns as intermediate variables. Schur complement systems are derived by eliminating u_i from the local system (5). By noticing that $u_i = B_i^{-1}(f_i - F_i y_i)$, we get

$$S_i y_i + \sum_{j \in \mathcal{N}_i} E_{ij} y_j = g_i - E_i B_i^{-1} f_i \equiv g'_i, \quad (6)$$

where S_i is the *local* Schur complement,

$$S_i = C_i - E_i B_i^{-1} F_i. \quad (7)$$

By assembling all the local Schur complement systems (6), we can get a *global* Schur complement system for all the interface unknowns. This global Schur complement system has a natural block structure:

$$\begin{pmatrix} S_1 & E_{12} & \dots & E_{1p} \\ E_{21} & S_2 & \dots & E_{2p} \\ \vdots & & \ddots & \vdots \\ E_{p1} & E_{p-1,2} & \dots & S_p \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \end{pmatrix} = \begin{pmatrix} g'_1 \\ g'_2 \\ \vdots \\ g'_p \end{pmatrix}. \quad (8)$$

The diagonal blocks in this system, the matrices S_i , are dense in general. The off-diagonal blocks E_{ij} are sparse. More compactly, the global Schur complement system (8) can be written as

$$S y = g',$$

where all the subdomain interface variable vectors y_1, y_2, \dots, y_p are stacked into a long vector y . The matrix S is called the global Schur complement matrix. An idea proposed in [8] is to exploit methods that *approximately* solve (8) and use them as preconditioners for (1). Such a Schur complement based parallel algebraic preconditioner can be expressed by the following algorithm:

ALGORITHM 2.1 *Schur complement based preconditioner*

1. Compute local right-hand sides: $g'_i = g_i - E_i B_i^{-1} f_i$.
2. Solve $S y = g'$ approximately.
3. Solve $B_i u_i = f_i - F_i y_i$.

Let us consider Step 2 in more detail. First, we rewrite (8) as a preconditioned system with the diagonal blocks:

$$y_i + S_i^{-1} \sum_{j \in \mathcal{N}_i} E_{ij} y_j = S_i^{-1} [g_i - E_i B_i^{-1} f_i] \quad \text{for all } i. \quad (9)$$

This can be viewed as a block-Jacobi preconditioned version of (8). This global system can be solved by a GMRES-like accelerator, requiring a solve with S_i on subdomain i in each iteration. An ILU factorization of S_i can be easily extracted from an ILU factorization of A_i . Specifically, if A_i is of the form (4) and is factored as $A_i = L_i U_i$, where

$$L_i = \begin{pmatrix} L_{B_i} & 0 \\ E_i U_{B_i}^{-1} & L_{S_i} \end{pmatrix} \quad \text{and} \quad U_i = \begin{pmatrix} U_{B_i} & L_{B_i}^{-1} F_i \\ 0 & U_{S_i} \end{pmatrix},$$

then, $L_{S_i} U_{S_i}$ is equal to the local Schur complement S_i . So an ILU factorization of A_i provides an approximation of S_i . There are, of course, other techniques for approximating S_i , see e.g. [8].

In [5], an Algebraic Recursive Multilevel Solver (ARMS) [9], which acts on A_i , has been used to extract an

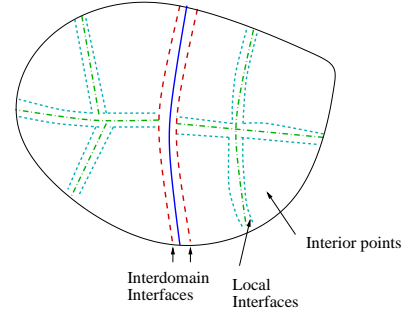


Figure 2. A two-level group-independent set.

approximation of the local Schur complement S_i . The construction of ARMS starts with defining “group-independent sets” in each subdomain. A group-independent set is a set of groups of unknowns such that there is no coupling between unknowns that belong to any two different groups [10]. An illustration is shown in Figure 2 for a simple case with two subdomains. The solid curve in the middle contains no vertices, it only illustrates the separation of the two subdomains. On each subdomain, an independent set reordering is applied locally. In Figure 2, local interfaces denote the unknowns that separate the group independent sets. The local system with matrix A_i can be reordered such that the rows corresponding to local and interdomain interface unknowns are placed after the rows corresponding to the group independent set unknowns. With such a permutation, the Schur complement include both types of the interface unknowns. We call this larger Schur complement the *expanded Schur complement*.

An advanced parallel algebraic preconditioner is thus to use a Krylov solver, preconditioned by a distributed ILU(0), for the global expanded Schur complement system. Such a Schur complement based preconditioner normally produces better convergence than a simple parallel block preconditioner. This Schur complement based preconditioner can, e.g., use ARMS as an approximate subdomain solver, denoted by $Schur_2$ in Section 4.4.

3. A suite of PDEs with test cases

In this section, we present a suite of four PDEs. We have deliberately chosen these PDEs, which are simple in the mathematical formulation, but of great importance for many complicated mathematical models. It is therefore hoped that the numerical results presented in Section 5 will be of interest for many. We have also taken care to select test cases involving 2D/3D computations on uniform, structured and unstructured computational grids. Although we use finite elements exclusively for the discretization, the resulting linear systems have similar properties as those arising from other discretization techniques, e.g., finite differences.

3.1. Poisson equation

Poisson equation is the simplest elliptic PDE and reads

$$-\nabla^2 u = f, \quad (10)$$

where f is prescribed.

Test Case 1. The Poisson equation (10) is considered in the 2D unit square. The right-hand side f is chosen as $f(x, y) = xe^y$, and $u(x, y) = -xe^y$ is given on the entire boundary $\partial\Omega$.

Test Case 2. The Poisson equation (10) is considered in the 3D unit cube. The right-hand side f is chosen as $f(x, y, z) = x(y^2 + z^2)e^{yz}$, and $u(x, y, z) = -xe^{yz}$ is given on the entire boundary $\partial\Omega$.

Test Case 3. The Poisson equation (10) is considered in a special 2D domain depicted in Figure 3. The right-hand side f and the boundary condition are the same as in Test Case 1. We use an unstructured 2D computational grid that has 521,185 points and 1,040,256 triangular elements.

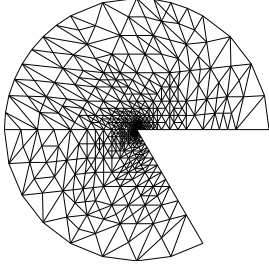


Figure 3. The solution domain and grid for Test Case 3; the Poisson equation.

3.2. Heat conduction

The following parabolic PDE is the simplest model of heat transfer in a homogeneous medium:

$$\frac{\partial u}{\partial t} = k\nabla^2 u, \quad (11)$$

where constant k denotes the heat conductivity.

Test Case 4. The 3D unit cube is used as the spatial solution domain Ω . We choose $k = 1$ and use an implicit time-stepping scheme for solving (11), i.e.,

$$\frac{u^l - u^{l-1}}{\Delta t} = \nabla^2 u^l \Rightarrow u^l - \Delta t \nabla^2 u^l = u^{l-1}, \quad (12)$$

where the superscript l denotes the time level number. We note that the resulting system matrix \mathbf{A} is on the form

$$\mathbf{A} = \mathbf{M} + \Delta t \mathbf{K}, \quad (13)$$

where matrix \mathbf{M} is often referred to as the *mass matrix* and \mathbf{K} arises from discretizing $-\nabla^2 u^l$. Moreover, we choose $\Delta t = 0.05$ and let $u^0(x, y) = \sin(\pi x) \sin(\pi y)$ be the initial condition. As boundary conditions, we use $u = 0$ on the side of $x = 1$ and $\frac{\partial u}{\partial n} = 0$ on the rest of $\partial\Omega$.

3.3. Convection-diffusion equation

For a stationary convective flow that is also affected by diffusion, its mathematical model reads as follows,

$$\vec{v} \cdot \nabla u = \nabla^2 u, \quad (14)$$

where the constant vector \vec{v} has magnitude v and an direction angle θ , i.e., $\vec{v} = (v \cos \theta, v \sin \theta)$. The magnitude v indicates the relation between convection and diffusion.

Test Case 5. We consider the convection-diffusion equation in the 2D unit square. The following boundary conditions are used: the normal derivative of u is zero on the right side ($x = 1$) and top side ($y = 1$). On the bottom side ($y = 0$) we have $u = 0$, whereas the left side ($x = 0$) is divided into two parts:

$$u = \begin{cases} 0 & x = 0, 0 \leq y \leq \frac{1}{4}, \\ 1 & x = 0, \frac{1}{4} < y \leq 1. \end{cases}$$

By choosing $v = 1000$, we make the test case convection-dominated. The angle of \vec{v} is chosen as $\theta = \pi/4$. Note that the boundary condition is discontinuous on $x = 0$, resulting in a rather sharp discontinuity of the solution that lies on a line starting from $(x = 0, y = 0.25)$ with an angle $\theta = \pi/4$; see Figure 4. Due to the dominating convection, we have to use one type of upwind weighting functions, see [4], resulting in an unsymmetric system matrix \mathbf{A} .

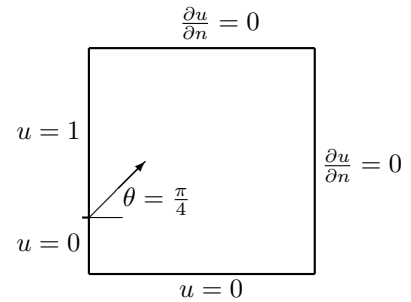


Figure 4. The solution domain of Test Case 5; the convection-diffusion equation.

3.4. Linear elasticity

For an elastic body subject to a volume load, the displacement field \vec{u} can be described by the following vector PDE that is of the elliptic type:

$$-\mu\Delta\vec{u} - (\mu + \lambda)\nabla(\nabla \cdot \vec{u}) = \vec{f}, \quad (15)$$

where μ and λ are elasticity constants, \vec{f} is a given vector function representing the volume load; see e.g. [11].

Test Case 6. We study the displacement field $\vec{u} = (u_1, u_2)$ inside one quarter of a ring, which has inner radius 1 and outer radius 2; see Figure 5. We use a curvilinear structured computational grid consisting of triangular shaped elements. For the boundary condition, the stress vector is specified on the entire boundary $\partial\Omega$ except on Γ_1 and Γ_2 , where $u_1 = 0$ on Γ_1 and $u_2 = 0$ on Γ_2 .

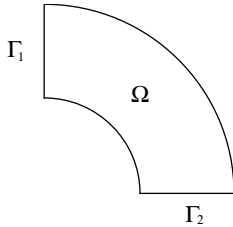


Figure 5. The solution domain for Test Case 6; the linear elasticity equation.

4. Experiment setup

4.1. Software

We will use a “composite” parallel code for running simulations of the test cases presented in Section 3. More specifically, two software packages—Diffpack and pARMS—are coupled together. Diffpack (see [2, 4]) is an object-oriented computational environment with a special emphasis on solving PDEs. A Diffpack simulator is typically set up using C++ objects of classes that are provided by hierarchically designed libraries. Diffpack has a rich collection of classes representing grids, fields, finite element discretization tools, matrices, vectors, Krylov solvers, preconditioners, convergence monitors, and many more. Parallelization of a sequential Diffpack simulator can be easily done by using a special add-on parallel toolbox, which contains functionalities for grid partitioning, communication pattern recognition, and inter-processor MPI communication routines that are associated with parallel Krylov iterations. We refer to [1] for more details.

pARMS (see [6]) is mainly a library of parallel linear algebra routines. Its main feature is a large collection of state-of-the-art parallel algebraic preconditioners based on various incomplete factorization techniques, recursive multilevel strategies, and Schur complement solvers. The reason for coupling together the two software packages is to equip Diffpack with modern parallel algebraic preconditioners. In the resulting “composite” code, Diffpack is responsible for the overall parallel computing framework, finite element discretization, control of Krylov iterations, whereas pARMS is responsible for performing the desired parallel algebraic preconditioning operations.

Due to Diffpack’s object-oriented design, it is quite easy to incorporate an “alien” code. In our case of using pARMS as new parallel preconditioners inside Diffpack, it suffices to write a small wrapper class, as a subclass of the generic preconditioner class in Diffpack. The function of this small wrapper class is to initiate the pARMS data structure, and shuffle data between Diffpack and pARMS every time a preconditioning operation needs to be carried out by the pARMS code.

4.2. Hardware

We have used two quite different parallel computing systems for running the simulations. One system is a low-end Linux cluster using Pentium III 1GHz processors and a fast Ethernet-based network. We have obtained exclusive access to run all the simulations on this relatively small parallel computer. The other system is a high-end SGI Origin 3800 machine using R14000 600MHz processors. Due to a restricted access to this parallel computer, we have only run a few selected simulations.

4.3. Measurements

In all the experiments to be presented in Section 5, we record the number of (F)GMRES iterations (with $m = 20$, see [7]), which is needed to achieve a *fixed* convergence criterion. More precisely, the 2-norm of the residual vector should be reduced by a factor of 10^6 , compared with its initial value. For Test Case 4, we only run one time step so the initial guess of the solution is the initial condition of the PDE. For all the other test cases, we use zero initial values for all the unknowns except those associated with Dirichlet boundary conditions. The obtained iteration counts reveal the convergence speed of different parallel algebraic preconditioners when applied to different test cases. We have used a *general* grid partitioning scheme (based on Metis [3]) for all the simulations. We remark that due to different random number generators on the two different parallel computing systems, the global computational grid is in fact partitioned differently on the Linux cluster than on the Origin

3800 machine.

In addition, we record the *wall-clock time* consumption of the preconditioned parallel (F)GMRES solver. For every test case, these measurements are used to compare the overall computational efficiency of the different algebraic preconditioners. However, we remark that wall-clock time measurements are highly sensitive to the work load level of a parallel computing system. (The Origin 3800 machine has often been heavily loaded during our experiments.) Certain care should therefore be taken when interpreting these timing results.

For each test case, we use a *fixed global problem size* and let the preconditioner type and the number of processors P vary. When the parallel efficiency is studied with respect to speed-ups, a fixed global problem size normally favors a smaller value of P , because the communication overhead is relatively low compared with the local computation. However, an opposite effect may occur if P exceeds a threshold such that the subdomain problems become small enough to be handled efficiently by the cache sitting on the an individual processor.

4.4. Notation

We use the following symbols to denote four different parallel algebraic preconditioners:

- $Schur_1$: A Schur complement enhanced preconditioner where the global Schur complement system is solved approximately using a few distributed global GMRES iterations preconditioned by block Jacobi. The associated subdomain solver is a few local GMRES iterations preconditioned by ILUT.
- $Schur_2$: A Schur complement enhanced preconditioner where the expanded global Schur complement system is solved approximately using a few distributed global GMRES iterations preconditioned by a global ILU(0). Two-level ARMS works as the approximate subdomain solver.
- $Block_1$: A simple block preconditioner using ILU(0) as the approximate subdomain solver.
- $Block_2$: A simple block preconditioner using ILUT as the approximate subdomain solver.

5. Numerical results

Results for test case 1. We use a distributed FGMRES solver and a fixed 2D global grid with $1001 \times 1001=1,002,001$ points. The following FGMRES iteration counts and wall-clock time measurements (in seconds) are obtained on the Linux cluster:

	$Schur_1$		$Schur_2$		$Block_1$		$Block_2$	
P	#itr	time	#itr	time	#itr	time	#itr	time
2	102	350.26	101	552.55	1188	1630.04	198	438.09
4	110	188.38	110	331.74	1371	1040.74	346	382.92
8	140	124.33	108	165.82	1008	425.18	352	200.26
16	146	76.70	104	100.59	977	223.31	315	102.65

We see that $Schur_1$ has the best overall computational efficiency for this test case, despite that $Schur_2$ has slightly faster and more stable convergence behavior. However, $Block_1$ has the best scalability with respect to wall-time per iteration. We have also tested $Schur_1$ and $Block_2$ on the Origin 3800 machine with the following results:

	$Schur_1$		$Block_2$	
P	#itr	time	#itr	time
8	121	303.34	270	118.34
16	145	190.14	386	83.89
32	167	153.31	511	122.70
64	184	88.78	840	98.96

We can see that the growth of iterations for $Schur_1$ is of moderate speed, whereas $Block_2$ requires many iterations for large values of P . We remark that the very poor wall-clock time measurements are due to the heavy load on the Origin 3800 machine, which ought to produce better measurements because of its more powerful processors and faster communication network. The different iteration counts for $P = 8$ and $P = 16$ are due to the different random number generators used in the grid partitioning scheme, as explained in Section 4.3.

Results for test case 2. We use a distributed FGMRES solver and a fixed 3D global grid with $101 \times 101 \times 101=1,030,301$ points. The following results are obtained on the Linux cluster:

	$Schur_1$		$Schur_2$		$Block_1$		$Block_2$	
P	#itr	time	#itr	time	#itr	time	#itr	time
2	60	307.25	70	790.69	93	273.15	74	254.11
4	60	161.77	70	441.38	97	154.90	81	137.82
8	60	88.89	70	275.94	103	89.68	87	81.45
16	61	53.60	69	196.27	110	50.77	93	45.21

We can see that all the four preconditioners produce quite fast convergence, where $Schur_1$ and $Schur_2$ show very stable iteration counts. Unlike the previous test case, the $Block_2$ preconditioner produces the best overall computational efficiency, while $Block_1$ also performs better than both $Schur_1$ and $Schur_2$ for this test case. We have also tested $Schur_2$ and $Block_2$ on the Origin 3800 machine with the following results:

	$Schur_2$		$Block_2$	
P	#itr	time	#itr	time
16	70	310.40	94	39.86
32	68	114.76	103	43.83
64	67	91.74	110	26.71

We can see that the iteration counts remain stable with $Schur_2$, whereas the iteration number growth for $Block_2$ is moderate.

Results for test case 3. We use a distributed FGMRES solver and a fixed global unstructured grid with 521,185 grid points. The following results are obtained on the Linux cluster:

P	$Schur_1$		$Schur_2$		$Block_1$		$Block_2$	
	#itr	time	#itr	time	#itr	time	#itr	time
2	148	223.20	74	256.54	1206	846.30	199	202.19
4	150	117.80	74	120.20	1451	530.48	171	88.87
8	152	66.07	74	70.60	1266	280.46	309	86.98
16	154	41.55	74	43.36	1163	155.06	337	51.83

The Schur complement enhanced preconditioners again show their advantage for this test case.

Results for test case 4. We use a distributed FGMRES solver and a fixed 3D global grid with $101 \times 101 \times 101=1,030,301$ points. The following results are obtained on the Linux cluster:

P	$Schur_1$		$Schur_2$		$Block_1$		$Block_2$	
	#itr	time	#itr	time	#itr	time	#itr	time
2	91	466.73	116	1332.34	165	475.84	129	436.84
4	93	251.02	115	794.15	166	257.60	132	224.83
8	93	139.57	114	473.67	164	139.88	137	131.49
16	95	85.91	113	322.01	168	80.30	148	73.45

For this 3D test case, all the preconditioners produce quite stable iteration counts. However, $Block_2$ seems to have the best performance with respect to the overall computational efficiency.

Results for test case 5. We use a distributed FGMRES solver and a fixed 2D global grid with $1001 \times 1001=1,002,001$ points. The following results are obtained on the Linux cluster:

P	$Schur_1$		$Schur_2$		$Block_1$		$Block_2$	
	#itr	time	#itr	time	#itr	time	#itr	time
2	11	36.84	27	150.88	285	428.94	29	64.36
4	17	29.16	29	94.71	291	227.30	54	61.31
8	14	12.22	29	53.77	297	131.66	38	21.87
16	16	8.15	30	30.69	298	75.53	40	14.46

We see that the $Schur_1$ preconditioner is a clear “winner” in the overall computational efficiency. Low iteration counts for $P = 32$ and $P = 64$ that are obtained on the Origin 3800 machine (see the table below) also seem to support the same observation.

P	$Schur_1$		$Schur_2$		$Block_1$		$Block_2$	
	#itr	time	#itr	time	#itr	time	#itr	time
8	15	13.96	27	35.72	296	75.30	32	55.10
16	17	25.13	not converged		297	49.25	39	28.35
32	20	21.97	30	20.11	318	38.63	78	19.57
64	23	13.93	30	13.94	309	13.13	92	18.19

Again, some of the wall-clock time measurements show traces of strong influence of heavy work load on the Origin 3800 machine. We remark that we “gave up” on $Schur_2$

when it did not help the FGMRES solver to converge for $P = 16$ on the Origin 3800 machine. This seems to be caused by an unfortunate grid partitioning result, since $Schur_2$ works fine on the Linux cluster for $P = 16$.

Results for test case 6. We use a distributed FGMRES solver and a fixed global curvilinear structured grid with 481×481 grid points, where each grid point is associated with two unknowns. The following results are obtained on the Linux cluster:

P	$Schur_1$		$Schur_2$	
	#itr	time	#itr	time
2	302	889.84	238	1224.90
4	556	840.66	193	544.26
8	724	572.28	280	404.59
16	620	280.70	299	255.06

This test case is clearly the “toughest” for the parallel algebraic preconditioners. The $Block_1$ and $Block_2$ preconditioners have trouble producing satisfactory convergence.

5.1. Effect of the subdomain shape

We have so far always used a general partitioning scheme that does not produce subdomains in the shape of rectangles or boxes, even if the global computational grid is uniform. Do we get more rapid convergence if the subdomains are of simple shapes? To answer this question, we try Test Case 2 again. This time we use a simple grid partitioning scheme that produces the subdomains as small boxes. The resulting FGMRES iteration counts and wall-clock time measurements, which are obtained on the Linux cluster for $P = 16$, are listed in the following table. We also compare them with the earlier measurements obtained by using a general grid partitioning scheme.

P	$Schur_1$		$Schur_2$		$Block_1$		$Block_2$	
	#itr	time	#itr	time	#itr	time	#itr	time
Test Case 2; general grid partitioning								
16	61	53.60	69	196.27	110	50.77	93	45.21
Test Case 2; simple grid partitioning								
16	63	50.91	70	193.81	109	47.79	93	43.01

We observe that the change in iteration counts is hardly noticeable. However, the simple grid partitioning scheme produces better balanced subdomains, such that the communication overhead is lower. This explains the slightly better wall-clock time measurements. However, we should remember that such a simple grid partitioning is *only* applicable to global computational grids in the shape of rectangles or boxes.

5.2. Comparison with additive Schwarz

It may be argued that an additive Schwarz preconditioner (see e.g. [11]) is an algebraic block preconditioner. But

we choose to view additive Schwarz preconditioners differently, because they normally use more than the minimum amount of overlap (see Section 1.1). In addition, their convergence depends greatly on the so-called *coarse grid corrections* (CGCs), which require an extra discretization on another global grid with very low grid resolution.

In the table below, we report the number of GMRES iterations and wall-clock time needed by an additive Schwarz preconditioner on the Linux cluster for Test Case 1. We remark that we use a simple partitioning scheme to produce subdomains as small rectangles. The amount of overlap in both x - and y -direction is about 5% of the side length of the subdomains. The global coarse grid has a fixed size: 5×17 , and the coarse grid level linear system is solved by Gaussian elimination. The subdomain solver in use is one Conjugate Gradient (see e.g. [7]) iteration accelerated by a special FFT-based preconditioner.

Test Case 1: global grid: 1001×1001				
	Additive Schwarz without CGCs		Additive Schwarz with CGCs	
P	#itr	time	#itr	time
2	26	88.86	13	47.84
4	33	61.80	18	37.50
8	65	64.04	26	29.97
16	115	56.24	39	23.36

We see that, when CGCs are not used, this additive Schwarz preconditioner has a dangerously rapid growth of iteration counts. However, with the help of CGCs, the additive Schwarz preconditioner clearly converges faster than all the four parallel algebraic preconditioners.

6. Concluding remarks

Our six test cases are of course insufficient for drawing any definitive conclusions on the numerical behavior of the parallel algebraic preconditioners that have been considered in this paper. However, the following observations seem to hold:

- The $Schur_1$ preconditioner seems to be the best choice, with respect to the overall computational efficiency. It has quite stable iteration counts, which are somewhat independent of P . Moreover, it has an acceptable scalability of the parallel efficiency, due to the relatively small overhead associated with solving the global Schur complement system.
- The $Schur_2$ preconditioner has the most stable iteration counts with respect to P . This preconditioner is good at achieving satisfactory convergence, but not overall computational efficiency.
- The $Block_1$ and $Block_2$ preconditioners have very good scalability of parallel efficiency, if we consider the computational cost per iteration with respect to P . However, $Block_1$ has often very slow convergence.

- All the four parallel algebraic preconditioners are quite insensitive to the shape of subdomains.

References

- [1] X. Cai and H. P. Langtangen. *Developing parallel object-oriented simulation codes in Diffpack*. In H.A. Mang, F.G. Rammerstorfer, J. Eberhardsteiner, editors, *Proceedings of Fifth World Congress on Computational Mechanics, Vienna, Austria, July 2002*.
- [2] Diffpack Home Page: <http://www.nobjects.com/>.
- [3] G. Karypis and V. Kumar. *Metis: Unstructured graph partitioning and sparse matrix ordering system*. Tech. Rep., Dept. of Computer Science, Univ. of Minnesota, 1995.
- [4] H. P. Langtangen. *Computational Partial Differential Equations – Numerical Methods and Diffpack Programming*. Springer-Verlag, 1999.
- [5] Z. Li, Y. Saad, and M. Sosonkina. *pARMS: A parallel version of the algebraic recursive multilevel solver*. Tech. Rep. UMSI-2001-100, Minnesota Supercomputer Institute, 2001.
- [6] pARMS Home Page: <http://www-users.cs.umn.edu/~saad/software/pARMS>.
- [7] Y. Saad. *Iterative Methods for Sparse Linear Systems*. PWS publishing, New York, 1996.
- [8] Y. Saad and M. Sosonkina. *Distributed Schur complement techniques for general sparse linear systems*. SIAM J. Sci. Comput. 21(4):1337–1356, 1999.
- [9] Y. Saad and B. Suchomel. *ARMS: An algebraic recursive multilevel solver for general sparse linear systems*. Tech. Rep. umsi-99-107-REVIS, Minnesota Supercomputer Institute, 2001.
- [10] Y. Saad and J. Zhang. *BILUM: Block versions of multi-elimination and multi-level ILU preconditioner for general sparse linear systems*. SIAM J. Sci. Comput. 20:2103–2121, 1999.
- [11] B. Smith, P. Bjørstad, and W. Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, New York, 1996.