The efficient parallel solution of PDEs

Gundolf Haase*, Ulrich Langer†

Johannes Kepler University Linz
Institute of Mathematics
Altenbergerstr. 69, A-4040 Linz
Austria

Computers and Mathematics with Applications:31 (3/4),1996, pp.151-159

Abstract

The report presents some results in solving finite element equations via a parallel version of the preconditioned cg-method (ParPCG). We use a nonoverlapping domain decomposition and construct preconditioners based on Additive and Multiplicative Schwarz Methods (ASM/MSM). As components in the preconditioner, multigrid methods, hierarchical bases, new extension techniques and modified BPS- and BPX-preconditioners for handling the unknowns at the coupling nodes on the boundaries between subdomains are used. The scale up efficiency (e.g. an increasing number of processors causes an increasing problem size) of the algorithm by doubling the number of processors is larger than 95%. Even the practical not relevant speed up efficiency (e.g. an increasing number of processors and a constant problem size) reaches 80% by doubling the number of processors.

Keywords : Boundary value problems, finite element method, domain decomposition, preconditioners, parallel iterative solvers.

1991 Mathematics subject classifications : 65N55, 65N50, 65N30

1 Introduction

During the last decade a lot of attention was paid to the development of parallel algorithms on massively parallel machines. Together with A. Meyer the authors proposed the parallelization and the preconditioning of the Conjugate Gradient (CG) method on the basis of a non-overlapping Domain Decomposition (DD) approach. The DD preconditioner proposed contains three components which can be chosen in order to adapt the preconditioner to the problem under consideration as well as possible. One component is a (modified) Schur-complement preconditioner that has been studied by the DD communitiy very intensively [1, 2, 3]. Another component is a preconditioner for the local problems with homogeneous Dirichlet boundary conditions arising in each subdomain. The most sensitive part is the basis transformation operator transforming the nodal f.e. basis on the interfaces into the approximate discrete harmonic basis [4, 5, 6, 7, 8]. In order to construct the two last components, we can use local multigrid methods. Local multigrid methods with zero initial guess have been already used for constructing the Dirichlet problem preconditioner as well as for the basis transformation. In the first case this is certainly sufficiently efficient. However, in the basis transformation case, the analysis shows that in general one has to carry out at least $O(\ln h^{-1})$ multigrid iterations in order to bound the term caused by the basis transformation in the condition number estimate uniformely in h [9]. In [10], A.M. Matsokin and

*ghaase@miraculix.numa.uni-linz.ac.at; phone/fax: austria–732–2468–9164 / –10
†ulanger@miraculix.numa.uni-linz.ac.at; phone/fax: austria–732–2468–9167 / –10

S.V. Nepomnyaschikh proposed a norm preserving extension procedure which immediately provides a uniform bound. In [11], the authors together with A. Meyer and S.V. Nepomnyaschikh derived a cheap hierarchical extension procedure with nearly the same behavior as the norm preserving extension procedure proposed in [10]. We combined these ideas, i.e. the grid functions obtained by hierarchical extension from the coupling boundaries (interfaces) are used as initial guesses for the local multigrid methods in the subdomains. So, we can improve the extented grid function very efficiently in direction of the harmonic extension without paying to much for it.

For measuring the efficiency of the various parallel algorithms the speed up (constant global problem size and increasing number of processors) and the scale up (increasing global problem size and increasing number of processors) are often used. For practical reasons the scale up is the more serious one because of the better use (costs !) of the hardware. We show that the scale up criterion is fulfilled in a good way by our algorithms under the restriction of low costs (time) per unknown.

2 The nonoverlapping DD-preconditioned parallelized CG-method

2.1 The algebraic approach (ASM)

Using a proper data distribution [8] and some version of a parallelized CG-method proposed by K.H. Law [12], we have developed various DD-preconditioners which do not (or very low) increase the amount of communication in the CG-method [4, 5].

Let us consider the abstract symmetric, $\mathbf{V}_0$–elliptic and $\mathbf{V}_0$–bounded variational problem

$$\text{Find } \; u \in \mathbf{V}_0 \, : \, a(u,v) \; = < F, v > \; \forall v \in \mathbf{V}_0 \; , \tag{1}$$

arising from the weak formulation of a scalar second–order, symmetric and uniformly elliptic boundery value problem (b.v.p.) given in a plane bounded domain $\Omega = \mathbf{R}^2$.

As in the finite element substructuring technique, we decompose Ω into p non-overlapping subdomains $\Omega_i \; (i = 1, 2, \ldots, p)$ such that $\overline{\Omega} = \bigcup\limits_{i=1}^{p} \overline{\Omega}_i$, and each subdomain Ω_i into Courant's linear triangular finite elements δ_r such that this discretization process results in a conform triangulation of Ω . In the following, the indices "C" and "I" correspond to the nodes belonging to the coupling boundaries (interfaces) $\Gamma_C = \bigcup\limits_{i=1}^{p} \partial\Omega_i \setminus \Gamma_D$ and to the interior $\Omega_I = \bigcup\limits_{i=1}^{p} \Omega_i$ of the subdomains, respectively, where Γ_D is that part of $\partial\Omega$ where Dirichlet–type boundary conditions are given.

So, we arrive at the large, sparse, symmetric and positive definte system of f.e. equations

$$\underbrace{\begin{pmatrix} K_C & K_{CI} \\ K_{IC} & K_I \end{pmatrix}}_{K} \begin{pmatrix} \underline{u}_C \\ \underline{u}_I \end{pmatrix} \; = \; \begin{pmatrix} \underline{f}_C \\ \underline{f}_I \end{pmatrix} \tag{2}$$

that we are going to solve on some parallel computer. Following the papers [6, 7], we construct a preconditioner for equation (2) via the Additive Schwarz Method (ASM). The preconditioner in the cg-preconditioning step $C\underline{w} \; = \; \underline{r}$ has the follwing structure :

$$C = \begin{pmatrix} I_C & K_{CI} B_I^{-T} \\ O & I_I \end{pmatrix} \begin{pmatrix} C_C & O \\ O & C_I \end{pmatrix} \begin{pmatrix} I_C & O \\ B_I^{-1} K_{IC} & I_I \end{pmatrix} . \tag{3}$$

This preconditioner contains the three components C_C, $C_I = \operatorname{diag}\left(C_{I,i}\right)_{i=1,2,\ldots,p}$ and $B_I = \operatorname{diag}\left(B_{I,i}\right)_{i=1,2,\ldots,p}$, which one has to choose in order to adapt the preconditioner to the specialities of the problem under consideration. In [6, 7] the authors showed that the spectral equivalence constants of the preconditioner C with respect to the matrix K are h-independent if and only if C_I and C_C are spectrally equivalent to K_I and to the modified Schur complement $S_C + T_C$ (with $S_C = K_C - K_{CI} K_I^{-1} K_{IC}$ and

$T_C = K_{CI}(K_I^{-1} - B_I^{-T})K_I(K_I^{-1} - B_I^{-1})K_{IC}$) , respectively, and if the spectral radius $\varrho(S_C^{-1})T_C$ is independent of h. The basis transformation operator B_I has to be non-singular. The preconditioner (3) leads us directly to the notation of the preconditioning step used in the Algorithm 1

Algorithm 1 : The ASM-DD Preconditioner [6, 7]
$\underline{\mathbf{w}}_C = C_C^{-1} \sum\limits_{i=1}^{p} A_{C,i}^T \left(\underline{r}_{C,i} - K_{CI,i}B_{I,i}^{-T}\underline{r}_{I,i}\right)$ $\underline{\mathbf{w}}_{I,i} = C_{I,i}^{-1}\underline{r}_{I,i} - B_{I,i}^{-1}K_{IC,i}\underline{\mathbf{w}}_{C,i} \quad ; i = 1, 2, \ldots, p$
Determined by the user : $C_C = ?$, $C_I = ?$, $B_I = ?$

where $A_i = \begin{pmatrix} A_{C,i} & A_{CI,i} \\ A_{IC,i} & A_{I,i} \end{pmatrix}$ denotes the subdomain connectivity matrix which is used for a convenient notation only. The subdomain f.e. assembling process which is connected with nearest neighbour communication stands behind this notation [5, 6, 7]. The special case $C_I = B_I$ leads to a simple DD-preconditioner (Algorithm 2).

2.2 Defining the preconditioner via iterative techniques

When using the operators C_C, C_I, B_I in the preconditioner (3), it is obvious that the invertation of those operators should be very cheap in the sense of arithmetical costs. For defining C_C we use the BPS- [2] and the BPX-technique [13, 3]. The other two components are constructed by iterative techniques with the selfadjoint (with respect to the K_I-inner energy product) iteration operator $M_I = \text{diag}\,[M_{I,i}]_{i=1,2,\ldots,p}$ and the iteration operator $\overline{M}_I = \text{diag}\left[\overline{M}_{I,i}\right]_{i=1,2,\ldots,p}$ so that we can write

$$C_I := K_I \left(I_I - M_I^k\right)^{-1} \tag{4}$$

$$B_I := K_I \left(I_I - \overline{M}_I^s\right)^{-1} . \tag{5}$$

The definitions above mean that the iteration process starts with an zero initual guess $\underline{w}^0 = 0$. In the following M_I and $\overline{M}_I$ are defined via local multigrid cycles as iteration technique. The use of (4) and (5) to Algorithm 1 is quite easy to implement.

2.3 The symmetric MSM-DD-preconditioner

In analogy to the construction of the preconditioner via ASM it is possible to construct a DD-preconditioner using a symmetric version of the MSM, for details see [14]. The remarkable result is the fact that this MSM-preconditioner can be interpreted as an ASM-preconditioner of the form (3). But the structure of the new (resulting from the MSM) ASM-components is of such a form that preferably the use of iterative techniques (M_I, $\overline{M}_I$ defined earlier) takes advantage of this new preconditioner. The new ASM(MSM)-components have the following form :

$$C_{I,theo} = \widehat{C}_I = K_I \left(I_I - M_I^{2k}\right)^{-1} \tag{6}$$

$$B_I = \widehat{B}_I = K_I \left(I_I - M_I^k \overline{M}_I^s\right)^{-1} . \tag{7}$$

The premise of a selfadjoint (with respect to the K_I-inner product) iteration operator M_I in the above theoretical definition for the inner preconditioner C_I can be weakened by using an arbitrary contraction operator ($\| M_I \| = \| \overset{*}{M}_I \| < 1$) and by replacing M_I^{2k} by $M_I^k \, \overset{*}{M}{}_I^k$, i.e.

$$C_I = \widehat{C}_I = K_I \left(I_I - M_I^k \, \overset{*}{M}{}_I^k\right)^{-1} . \tag{8}$$

The preconditioner is realized in such a way that instead of using $4k$ applications of M_I just $2k$ applications of M_I are needed. The algorithm looks a little bit more simple than the Algorithm 3 described in the following section.

3 The non-zero initial guess in the basis transformation

The estimation of the condition number of the preconditioned system $C^{-1}K$ as well as numerical experiments showed us that the crucial point in the preconditioning operator C (3) is the local basis transformation operato $B_{I,i}$. Looking carefully on the right-hand side of the basis transformation equation

$$B_{I,i}\,\widetilde{\underline{w}}_{I,i} \;=\; -K_{IC,i}\,\underline{w}_{C,i} \qquad i=1,2,\ldots,p\,, \tag{9}$$

we observe that the right-hand side belongs to the subspace $K_{IC}\mathbf{R}^{N_C}$. So, we must adapt the basis transformation technique to this subspace.

The most promising technique for constructing $B_{I,i}$ consists certainly in the use of a suitable non-zero initial guess in the iterative method $\overline{M}_I$. The first idea of using a full multigrid cycle instead of a multigrid cycle with a zero initial guess gives us a better iteration count but almost no gain in computation costs [15, 9]. A second idea consists of the hierarchical approximation of a norm-preserving extension technique by S.V. Nepomnyaschikh which can be found in [11]. Because of the fact that solving equation (9) is equivalent to the problem of finding the harmonic extension of the boundary data into the interior of the domain, the used extension technique is a arithmetically cheap and a good method for achieving the initial guess.
Denoting the extension technique by E_{IC} we can write our new basis transformation operator B_{IC} in the form

$$B_{IC} \;=\; -B_I^{-1}K_{IC} \;=\; \overline{M}_I^s\,E_{IC} + \left(I_I - \overline{M}_I^s\right)\,K_I^{-1}\,(-K_{IC}) \quad . \tag{10}$$

The resulting algorithm for the preconditioning step $C\underline{w} = \underline{r}$ in the ParPCG yields the notation :

Algorithm 3 : The MSM-DD Preconditioner [14]				
1)	$\underline{y}_{I,i}$	$=$	$(I_{I,i} - \overset{*}{M}{}^k_{I,i})\,K_{I,i}^{-1}\underline{r}_{I,i}$	$i=1,2,\ldots,p$
2)	$\hat{\underline{w}}_{I,i}$	$=$	$\overset{*}{\overline{M}}{}^s_{I,i}\underline{y}_{I,i} + (I_{I,i} - \overset{*}{\overline{M}}{}^s_{I,i})\,K_{I,i}^{-1}\underline{r}_{I,i}$	$i=1,2,\ldots,p$
3)	$\underline{w}_C$	$=$	$C_C^{-1}\sum\limits_{i=1}^{p} A_{C,i}^T\,(\underline{r}_{C,i} - K_{CI,i}\hat{\underline{w}}_{I,i} + E_{CI,i}(\underline{r}_{I,i} - K_{I,i}\hat{\underline{w}}_{I,i}))$	$i=1,2,\ldots,p$
4)	$\underline{z}_{I,i}$	$=$	$\overline{M}^s_{I,i}E_{IC,i}\underline{w}_{C,i} + (I_{I,i} - \overline{M}^s_{I,i})\,K_{I,i}^{-1}\,(-K_{IC,i}\underline{w}_{C,i})$	$i=1,2,\ldots,p$
5)	$\widetilde{\underline{r}}_{I,i}$	$=$	$\underline{r}_{I,i} - K_{IC,i}\underline{w}_{C,i}$	$i=1,2,\ldots,p$
6)	$\widetilde{\underline{w}}_{I,i}$	$=$	$\underline{y}_{I,i} + \underline{z}_{I,i}$	$i=1,2,\ldots,p$
7)	$\underline{w}_{I,i}$	$=$	$M^k_{I,i}\widetilde{\underline{w}}_{I,i} + (I_{I,i} - M^k_{I,i})\,K_{I,i}^{-1}\widetilde{\underline{r}}_{I,i}$	$i=1,2,\ldots,p$
Determined by the user :	$C_C = ?,\ M_I = ?,\ \overline{M}_I = ?,\ k = ?,\ s = ?$			

Together with the MSM-method we have got a cheap and, at the same time, good preconditioner C. If we use the hierarchical extension combined with multigrid and a BPX-like Schur-Complement preconditioner we attain a count of arithmetical operations of the order $O(h^{-2}\log\log h^{-1}\log\varepsilon^{-1})$ when solving equation (2) via ParPCG [11]. The numerical examples in the following section demonstrate this.

4 Numerical Experiences with the MSM-DD-preconditioner on Massively Parallel Computers

In the numerical examples we use Algorithm 1 (ASM), Algorithm 3 (MSM + hier. extension) and the follwing abbreviations (just examples) :

V01	:	one V-multigrid-cycle with one post-smoothing sweep on all grids;
3W22	:	3 (= k, s) W-cycles with two pre- and two post-smoothing sweeps on all grids, corresponding to the exact solving;
E_{IC}	:	hierarchical extension [11];
BPS / BPX	:	Schur complement preconditioners based on [2] / [3].

We use lexicographically forward and backward Gauss-Seidel sweeps in the pre- and post-smoothing, respectively. The interpolation and the restriction operators are defined by the bilinear (TRANSCGI) or linear (FEM$\bigcirc\!\!\!\!\bigcirc$BEM) interpolation $I^k_{I;k-1}$ and by the canonical restriction $I^{k-1}_{I;k} = (I^k_{I;k-1})^T$ for all $k = 0, 1, \ldots, l$.

The ParPCG-iteration was stopped if the relative accuracy $\varepsilon = 10^{-6}$, i.e. $(C^{-1}\underline{r}^j, \underline{r}^j) \leq \varepsilon\,(C^{-1}\underline{r}^0, \underline{r}^0)$ had been attained. We tested our algorithms on a MultiCluster-II with 32 T805 (8MB) processors and on a GCel with 128 T805 (4MB) processors.

4.1 An academic example

We want to solve the Dirichlet problem for the Poisson-equation with homogenious Dirichlet boundary conditions in the rectangle $(0, 2) \times (0, 1)$ which was decomposed into 32 subdomains ($p = 32$), and each subdomain was divided into three-nodes (linear) triangular elements. These calculations were carried out by the test code TRANSCGI [16].

In comparision to a modified Algorithm 1 (ASM) which uses also the hierarchical extension technique the Algorithm 3 (MSM) attained much better iteration counts by the same choice of the iterative components M_I, $\overline{M}_I$. The following tables show the behavior of the Algorithm 3

components			l = # level - 1					
E_{IC}	$\overline{M}_I$	M_I	2	3	4	5	6	7
not used	$V11$	$V11$	13	13	15	16	18	22
used	$s = 0$	$V11$	13	14	15	17	18	20
used	$s = 0$	$V22$	13	13	15	16	17	18
used	$V11$	$V11$	13	13	14	15	16	18
not used	$3W22$	$3W22$	13	13	14	15	16	18

Table 1: Number of iterations for Algorithm 3 (MSM) , C_C = BPS (MultiCluster-II)

components			l = # level - 1					
E_{IC}	$\overline{M}_I$	M_I	2	3	4	5	6	7
not used	$V11$	$V11$	0.90	2.50	7.13	24.4	102.5	484.8
used	$s = 0$	$V11$	0.88	1.75	5.04	19.4	76.5	326.0
used	$s = 0$	$V22$	0.87	2.16	5.78	21.7	86.6	357.1
used	$V11$	$V11$	0.98	2.73	6.39	25.1	98.5	431.3
not used	$3W22$	$3W22$	2.28	6.04	23.46	97.6	419.8	1900.0

Table 2: CPU-time in seconds for Algorithm 3 (MSM) , C_C = BPS (MultiCluster-II)

Looking at Tables 1 and 2 we see that the best choice of the components, with respect to the practically interesting CPU-time, is not the choice with the iteration counts near the limit using exact solvers (3W22-cycles). The time of 326 seconds for solving the 8-grid ($l = 7$) problem with 2.100.225 unknowns is even better than the best one obtained by Algorithm 1. We have got nearly the same behavior of iteration counts and CPU-time in the case of subdividing the rectangle into 128 subdomains.

Now let us study the scale-up and the corresponding efficiency for the algorithm giving the best results in our first example, namely the Algorithm 3 (MSM) with E_{IC} (is used), $s = 0$ and M_I defined by V11 and C_C = BPS. We use 16, 64 and 8, 32, 128 processors corresponding to the decomposition of $\Omega = (0, 1) \times (0, 1)$ into 16, 64 subdomains and to the decomposition of $\Omega = (0, 2) \times (0, 1)$ into 8, 32 and 128 subdomains. We measure the scale-up $S(i, j)$ and the efficiency $E(i, j)$ in the 7-level ($l = 6$) case (Table 3) by increasing the number of processors from i to j. These experiments were carried out on a GC-system with 192 T805 transputers and 4 MByte per node (up to 128 processors were used in our experiments). Note that the 7-level case for 128 subdomains has **2.100.225** unknowns totally and **16.641** unknowns per subdomain $\overline{\Omega}_i$.

$\rightarrow j$ / $\downarrow i$ — S(i,j) / E(j,i) —	8	16	32	64	128
8	—	—	**3.96**	—	**14.35**
16	—	—	—	3.92	—
32	**0.99**	—	—	—	**3.67**
64	—	0.98	—	—	—
128	**0.91**	—	**0.92**	—	—

Table 3: Scale-up $S(i,j)$ and Efficiency $E(i,j)$ for the 7-level ($l=6$)case

We see that there is a very good efficieny by increasing the number of processors by the factor 4 (and also increasing the global number of unknowns) for the fastest algorithms. On the other hand this was just an academic example which gives us hope that the good behavior will be attained in practical examples (see the next section).

4.2 The Electric Motor

As a more challenging example we consider an electric motor. It consists of four permanent magnets of alternating magnetization, a rotor with 4 wings, regions of air and an iron coating. Figure 1 shows the initial mesh (grid 1) being used.

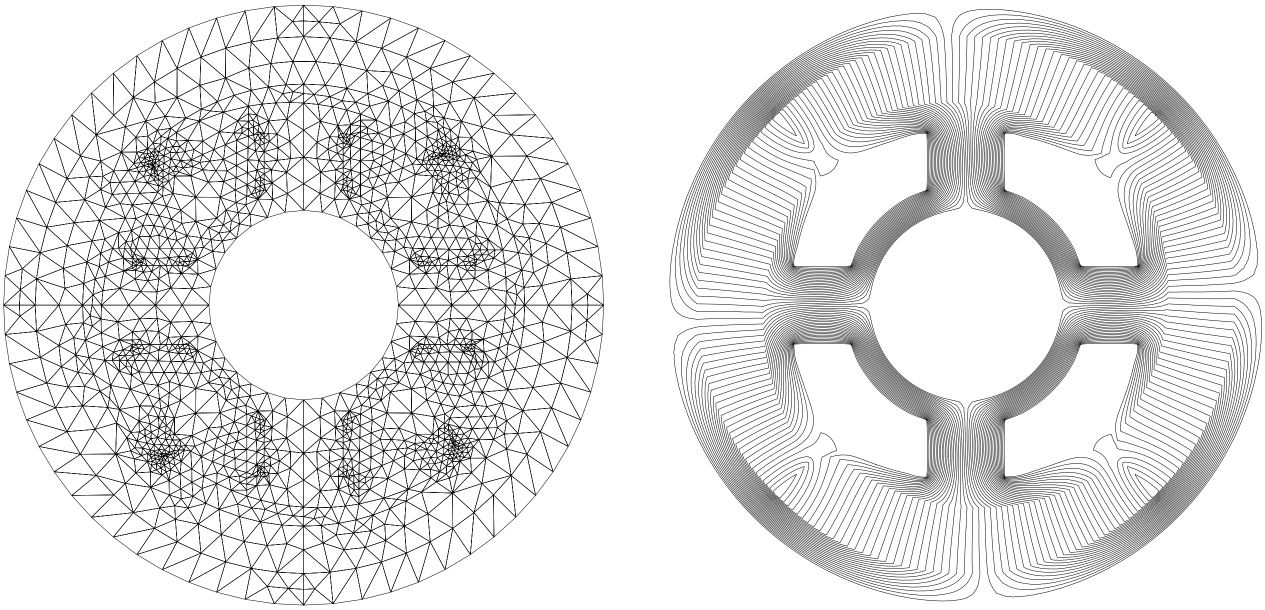

Figure 1: FEM discretization of the full motor and equipotential lines of the solution.

We solve the following boundary value problem in variational formulation :

Find the z-component $u = A_z \in \mathbf{V}_0$ of the vector potential $\vec{A} = (A_x, A_y, A_z)^T$: $\qquad \forall v \in \mathbf{V}_0 = \overset{0}{\mathbf{H}}{}^1(\Omega)$

$$\int_\Omega \frac{1}{\mu_0 \mu_r(x,y)} \nabla^T u \, \nabla v \, \mathrm{dx}\,\mathrm{dy} = \int_\Omega S_z \, v \, \mathrm{dx}\,\mathrm{dy} + \int_\Omega \frac{1}{\mu_0 \mu_r(x,y)} \left(\frac{\partial v}{\partial y} B_{0x} - \frac{\partial v}{\partial x} B_{0y} \right) \mathrm{dx}\,\mathrm{dy} \qquad (11)$$

where $\mu_r(\cdot)$ usually depends on $|\nabla u|$ for ferromagnetic materials [17]. In our example, we assume that $\mu_r(\cdot)$ is independent of $|\nabla u|$! In the numerical tests just the upper part of the motor domain was used.

These calculations were performed on a MultiCluster-II with 32 processors by the code FEM⊕BEM [18] which can solve nonlinear symmetric problems in a coupled BEM/FEM-discretization via a parallelized version of the Bramble/Pasciak-cg [19]. For this example just the linear FEM-branch of the code was used.
First we use the Algorithm 3 with C_C = BPS as in the forgoing section and obtain the iteration counts (CPU-times include only the solving of the linear system) :

components			l = # level - 1			
E_{IC}	$\overline{M}_I$	M_I	2	3	4	5
used	$s=0$	$V01$	58	> 200	> 200	> 200
used	$s=0$	$V11$	32	38	44	50
	CPU	sec.	6.2	19.9	81.6	359.6
used	$s=0$	$V02$	33	40	46	52
used	$s=0$	$V22$	32	38	44	49
used	$3W22$	$3W22$	32	37	43	48

Table 4: Number of iterations for Algorithm 3 (MSM) , C_C = BPS (MultiCluster-II)

Again, we see with exeption of M_I =V01, the rather cheap choice $\overline{M}_I = I_I$ ($s = 0$) produces iteration counts near the optimum but in distinction to the academic example in 4.1 the behavior of the iteration counts is much worse than in the case of using exact solvers. That means there is no chance for attaining better iteration counts with the BPS Schur-complement preconditioner.

So, we implemented the BPX Schur-complement preconditioner and got promising iteration counts.

components			l = # level - 1			
E_{IC}	$\overline{M}_I$	M_I	2	3	4	5
used	$s=0$	$V01$	33	39	46	53
used	$s=0$	$V11$	34	38	44	49
used	$s=0$	$W01$	28	30	33	35
used	$s=0$	$V02$	28	33	35	39
used	$s=0$	$V22$	30	31	33	35
used	$3W22$	$3W22$	21	22	23	23

Table 5: Number of iterations for Algorithm 3 (MSM) , C_C = BPX (MultiCluster-II)

The BPX Schur-complement preconditioner brings us much lower iteration counts if exact solvers are used. Also the iteration counts for M_I = V22 and V02 are moderate. In difference to Table 4 the choice M_I = V01 behaves not much worse than the choise M_I = V11. Rather remarkable is the fact that the influence of the pre-smoothing sweeps is neglectable. The reason seems to be in the behavior of the lower frequencies in the modified Schur-Complement $S_C + T_C$.

components			l = # level - 1			
E_{IC}	$\overline{M}_I$	M_I	2	3	4	5
used	$s=0$	$V01$	6.9	16.5	57.4	241.4
used	$s=0$	$V11$	7.8	20.8	80.3	344.6
used	$s=0$	$W01$	6.8	16.8	59.6	249.2
used	$s=0$	$V02$	6.4	17.3	60.2	256.8
used	$s=0$	$V22$	8.1	24.3	95.5	405.0

Table 6: CPU-time in seconds for Algorithm 3 (MSM) , C_C = BPX (MultiCluster-II)

The iteration count is heavily weighted by the arithmetical costs of the iterative techniques. Because of this we recognize the effect that the cheap V01-cycle is the best one (solves a linear system with 185.863 unknowns in 241 seconds !) although the V22-cycle results in much better iteration counts. Why the choice of the V01-cycle for defining M_I works well with the BPX-preconditioner but not with the BPS-preconditioner will be the scope of further investigations.

References

[1] M. Dryja. A capacitance matrix method for Dirichlet problems on polygonal regions. *Numerische Mathematik*, 39(1):51–64, 1982.

[2] J. H. Bramble, J. E. Pasciak, and A. H. Schatz. The construction of preconditioners for elliptic problems by substructuring I – IV. *Mathematics of Computation*, 1986, 1987, 1988, 1989. 47, 103–134, 49, 1–16, 51, 415–430, 53, 1–24.

[3] Charles H. Tong, Tony F. Chan, and C.C. Jay Kuo. A domain decomposition preconditioner based on a change to a multilevel nodal basis. *SIAM J. SciṠtat. Comput.*, 12(6):1486–1495, 1991.

[4] G. Haase and U. Langer. On the use of multigrid preconditioners in the domain decomposition method. In W. Hackbusch, editor, *Parallel Algorithms for PDEs*, pages 101–110, Braunschweig, 1990. Vieweg. Proc. of the 6th GAMM–Seminar, Kiel, 1990.

[5] A. Meyer. A parallel preconditioned conjugate gradient method using domain decomposition and inexact solvers on each subdomain. *Computing*, 45:217–234, 1990.

[6] Gundolf Haase, Ulrich Langer, and Arnd Meyer. A new approach to the Dirichlet domain decomposition method. In Sabine Hengst, editor, *Proceedings of the "5-th Multigrid Seminar" held at Eberswalde, GDR, May 14-18, 1990*, pages 1–59, Berlin, 1990. Academy of Science. Report-Nr. R-MATH-09/90.

[7] G. Haase, U. Langer, and A. Meyer. The approximate dirichlet domain decomposition method. Part I: An algebraic approach. Part II: Applications to 2nd-order elliptic boundary value problems. *Computing*, 47:137–151 (Part I), 153–167 (Part II), 1991.

[8] Gundolf Haase, Ulrich Langer, and Arnd Meyer. Parallelisierung und Vorkonditionierung des CG-Verfahrens durch Gebietszerlegung. In *Parallele Algorithmen auf Transputersystemen*, Teubner-Scripten zur Numerik III, Stuttgart, 1992. Teubner. Tagungsbericht der GAMM-Tagung, 31. Mai- 1. Juni 1991, Heidelberg.

[9] Gerd Kunert. On the choice of the basis transformation for the definition of DD Dirichlet preconditioners. Preprint-Reihe der Chemnitzer DFG-Forschergruppe "'Scientific Parallel Computing"' SPC 94-9, TU Chemnitz-Zwickau, 1994.

[10] A. M. Matsokin and S. V. Nepomnyaschikh. A Schwarz alternating method in a subspace. *Soviet Mathemathics*, 29(10):78–84, 1985.

[11] Gundolf Haase, Ulrich Langer, Arnd Meyer, and S. V. Nepomnyaschikh. Hierarchical extension operators and local multigrid methods in domain decomposition preconditioners. *East-West J. Numer. Math.*, 2(3), 1994.

[12] K. H. Law. A parallel finite element solution method. *Computer and Structures*, 23(6):845–858, 1989.

[13] J. H. Bramble, J. E. Pasciak, and J. Xu. Parallel multilevel preconditioners. *Mathematics of Computation*, 55(191):1 – 22, 1990.

[14] Gundolf Haase and Ulrich Langer. The non-overlapping domain decomposition multiplicative Schwarz method. *International Journal of Computer Mathemathics*, 44:223–242, 1992.

[15] Gundolf Haase and Ulrich Langer. Domain decomposition vs. adaptivity. In *Finite Element Methods: Fifty Years of the Courant Element*, Lecture notes in pure and applied mathematics, pages 243–257. Marcel Dekker Publ. Inc., 1993.

[16] Gundolf Haase and Michael Kuhn. TransCG - a program for testing parallel algorithms. DFG-Schwerpunkt "'Randelementmethoden"' 94-13, University Stuttgart, 1994.

[17] Bodo Heise. Analysis of a fully discrete finite element method for a nonlinear magnetic filed problem. *SIAM J. Numer. Anal.*, 31(3):745–759, 1994.

[18] Gundolf Haase, Bodo Heise, Michael Jung, and Michael Kuhn. FEM$\oplus$BEM - a parallel solver for linear and nonlinear coupled FE/BE-equations. DFG-Schwerpunkt "'Randelementmethoden"' 94-16, University Stuttgart, 1994.

[19] J. H. Bramble and J. E. Pasciak. A preconditioning technique for indefinite systems resulting from mixed approximation of elliptic problems. *Math. Comput.*, 50(181):1–17, 1988.