Numerical study of the performance of preconditioners based on algebraic multigrid method and approximate sparse inverses

Veselin Dobrev, Richard Ewing, Raytcho Lazarov, and Joseph Pasciak *

Abstract

Application of algebraic multigrid method and approximate sparse inverses are applied as preconditioners for large algebraic systems arising in approximation of diffusion-reaction problems in 3-dimensional complex domains. Here we report the results of numerical experiments when using highly graded and locally refined meshes for problems with non-homogeneous and anisotropic coefficients that have small features and almost singular solutions. For the discretization of the domain and the finite element approximation we have used the system AGGIEFEM, a universal computational tool for PDEs developed in the VIGRE seminar in Introduction to Scientific Computing at TAMU. For solving the algebraic system we have used ParaSails and BoomerAMG preconditioners that are part of the HYPRE (High Performance Preconditioners) library developed in CASC at Lawrence Livermore National Laboratory.

1 Introduction

In this report we summarize the results of computational experiments intended to test and compare different types of preconditioners for solving large-scale systems of linear equations arising from the discretization of 3-D self-adjoint elliptic partial differential equations. Our aim was to make test problems that are related to realistic situations in the modeling of flows in porous media that have some specific characteristics:

- the domain has a complex structure in both geometry and physics involving the porous media;
- the media inside the domain has layered structure and the diffusion coefficients vary by orders of magnitude from one layer to another;

*This research has been supported by Saudi Aramco Oil Company through a gift grant to the Institute of Scientific Computation, TAMU

- the layers have different thicknesses: there are layers with thickness comparable to the size of the domain as well as ones that are very thin;
- the very thin layers might be highly anisotropic (the permeability in the vertical direction is much lower); and
- there are small features inside the domain, these are needle-like regions with very high permeability.

All of these characteristics of the differential equation give rise (after discretization) to large ill-conditioned linear systems of equations, which are computationally challenging to solve, especially if considered as an intermediate step in a non-linear and/or time dependent iteration.

To accurately capture the jumps of the coefficients, we consider discretizations on unstructured meshes that are aligned with the jumps. Because of the character of these jumps, such discretizations generate elements that are in general not quasi-uniform: very small or very large elements and also geometrically anisotropic (thin) elements (inside the thin layers). Additionally, since the typical solutions of such equations have singularities, we also consider locally refined meshes.

For the purpose of our tests, we consider a set of meshes with varying mesh sizes (both globally and locally) which give rise to linear systems with a relatively wide range of sizes. In our comparison of the preconditioners, we consider the two stages that are involved in their use:

- construction of the preconditioner – measured in terms of the setup time;
- solving the linear system – measured in number of iterations needed for the iterative method to converge and the total time for solving.

We measure these characteristics of the preconditioners on our set of test problems, and we solve them on 1, 2, 4, 6, and 8 processors. Thus, we can measure the scalability with respect to the size of the problem as well as the parallel efficiency.

We tested the following two preconditioners from the HYPRE (High Performance Preconditioners) library developed in Lawrence Livermore National Laboratory [1]:

- ParaSails – parallel implementation of a sparse approximate inverse preconditioner (see the references [3] and [4]) and
- BoomerAMG – parallel implementation of algebraic multigrid (see, e.g. [5, 6, 7].

In both cases, the default set of parameters of the preconditioners was used.

In all computations, the preconditioned conjugate gradient (PCG) method was used with the default (for the HYPRE implementation of PCG) absolute stopping criterion

[1] `http://www.llnl.gov/CASC`

and tolerance 10^{-9}. All computations were performed on a SGI ORIGIN 2000 computer with 8 MIPS R10000 processors running at 250MHz with 4MB L2 cache and total main memory of 4GB.

The algebraic multigrid (AMG) was first introduced in the 1980's [1, 2, 9]. The main advantage of the AMG algorithm when compared to standard multigrid is that it does not require access to any additional information (such as hierarchy of grids, coarse-grid and interpolation matrices) — the algorithm constructs them itself. Thus, it is applicable to linear systems arising from the discretization of PDEs on large unstructured grids. In the same time its performance is comparable to the performance of standard multigrid. Also, AMG has been successfully applied to problems with large jumps in the coefficients and/or anisotropic coefficients as well as to non-symmetric problems.

The application of an algorithm to the solution of problems involving very large number of unknowns naturally leads to the necessity of parallelizing that algorithm. The original formulation of AMG contained part (the coarse-grid selection) that is sequential in nature. To overcome this drawback new coarsening algorithms were proposed [5, 6]. Subsequently, these algorithms as well as some other approaches were implemented in the HYPRE library with the name BoomerAMG. Their description along with a wide range of numerical experiments can be found in [7].

2 Description of the Model Problem

We consider the following model problem:

$$\begin{aligned} -\nabla \cdot (\mathbf{k} \cdot \nabla u) &= 0, && \text{in } \Omega \\ \mathbf{n} \cdot (\mathbf{k} \cdot \nabla u) &= 0, && \text{on } \Gamma_N \\ u &= u_0, && \text{on } \Gamma_D, \end{aligned}$$

where the domain Ω is the cube $\Omega = (-1, 1)^3$. The domain includes a well in its center $\Omega_5 = (-0.01, 0.01)^2 \times (0, 1)$,and has 4 layers with different thicknesses (see Figure 1),

$$\begin{aligned} \Omega_1 &= \{\Omega \cap \{0.5 < z < 1\}\} \setminus \overline{\Omega}_5 \\ \Omega_2 &= \{\Omega \cap \{0.45 < z < 0.5\}\} \setminus \overline{\Omega}_5 \\ \Omega_3 &= \{\Omega \cap \{0.2 < z < 0.45\}\} \setminus \overline{\Omega}_5 \\ \Omega_4 &= \{\Omega \cap \{-1 < z < 0.2\}\} \setminus \overline{\Omega}_5. \end{aligned}$$

The Dirichlet part of the boundary, Γ_D, consists of the bottom of the domain and the top of the well, i.e. $\Gamma_D = \partial\Omega \cap \{\{z = -1\} \cup \partial\Omega_5\}$, while the rest of the boundary forms the Neumann boundary $\Gamma_N = \partial\Omega \setminus \Gamma_D$. The permeability coefficient $\mathbf{k}$ has different values in the different layers and inside the well:

$$\mathbf{k}(x, y, z) = \mathbf{k}_i(x, y, z) \quad \text{for } (x, y, z) \in \Omega_i, \quad i = 1, 2, 3, 4, 5$$

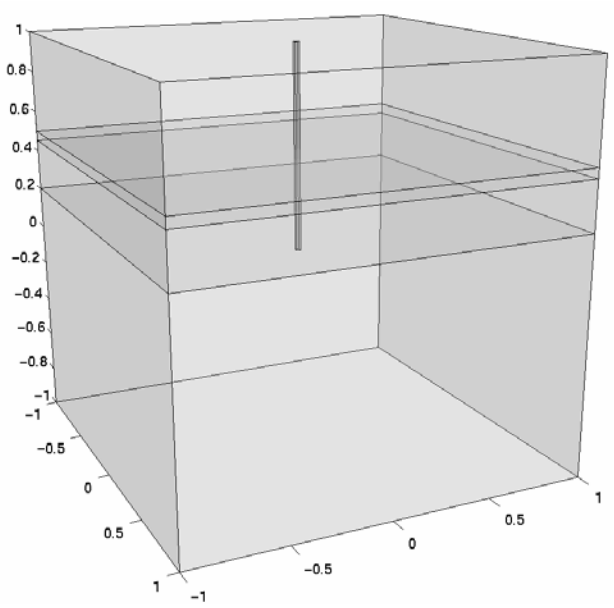

Figure 1: Plot of the domain with the subdomains

namely

$$\begin{aligned}
\mathbf{k}_1 &= \mathrm{diag}(4, 1, 1),\\
\mathbf{k}_2 &= \mathrm{diag}(0.2, 0.2, 0.01),\\
\mathbf{k}_3 &= \mathrm{diag}(1, 3, 1),\\
\mathbf{k}_4 &= \mathrm{diag}(1, 1, 1),\\
\mathbf{k}_5 &= \mathrm{diag}(10^4, 10^4, 10^4).
\end{aligned}$$

The Dirichlet boundary conditions are

$$u_0 = \begin{cases} 1, & \text{on } \Gamma_D \cap \{z = -1\} \text{ (bottom of the domain)} \\ 2, & \text{on } \Gamma_D \cap \{z = 1\} \quad \text{(top of the well)}. \end{cases}$$

This problem is a prototype of a steady-state flow in layered porous media with a well that penetrates part of the reservoir.

3 Discretizations of the Problem

The domain Ω was discretized into tetrahedra using the mesh generator NETGEN [2] (see also [8, 10]). The mesh was generated in such a way that each tetrahedron is contained

[2] `http://www.sfb013.uni-linz.ac.at/~joachim/netgen/`

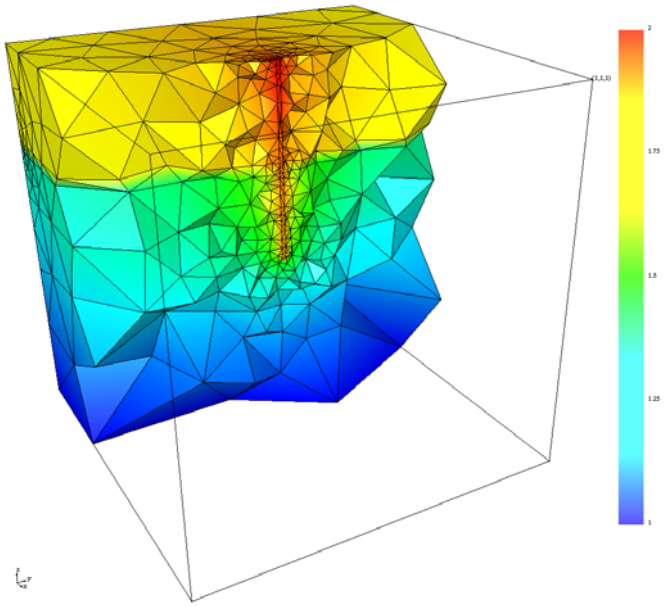

Figure 2: Cross section of the initial discretization of the domain, i.e., mesh $0-0$

completely in just one subdomain, which means that the mesh is aligned with the jumps of the coefficient $\mathbf{k}$ (see Figure 2.) Starting with this initial mesh, the mesh "$m-n$" (or discretization on the domain) is obtained by two steps:

- m times uniform refinement of the whole mesh (each tetrahedron is divided into 8 tetrahedra on each step) and then
- $(n-m)$ times refinement of all tetrahedra inside the well (i.e., in subdomain Ω_5) while the tetrahedra in the other subdomains are refined only to maintain the mesh conforming.

We considered the meshes "$m-n$" for $m=0,1,2$, and $n=m,\ldots,4$. The characteristics of the mesh, such as number of finite elements, number of mesh points, and number of finite elements in the "well"-region are given in Table 1.

4 Computational Results with ParaSails

All figures in this Section have logarithmic scale axes in both x and y directions. Thus, the functions $c.x^{\alpha}$ are represented as straight lines with slope α and also a function $f(x)$ that is bounded from below by $c_1.x^{\alpha}$ and from above by $c_2.x^{\alpha}$ and has its graph situated between the two parallel lines representing the graphs of the functions $c_i.x^{\alpha}$, $i=1,2$. So, to illustrate such behavior of the plotted quantities, we include the graph of a function $c.x^{\alpha}$ for some particular α.

	# of FEs	# of FE in Ω_5	# of unknowns
$0-0$	7,339	147	1,475
$0-1$	25,939	1,176	4,847
$0-2$	52,514	9,408	9,481
$0-3$	173,806	75,264	30,361
$0-4$	892,252	602,112	152,438
$1-1$	58,712	1,176	10,693
$1-2$	82,048	9,408	14,799
$1-3$	202,358	75,264	35,543
$1-4$	920,592	602,112	157,626
$2-2$	469,696	9,408	81,717
$2-3$	580,352	75,264	100,854
$2-4$	1,291,124	602,112	221,700

Table 1: Mesh characteristics

In Figure 3, the number of the iterations in the PCG method, N_{it}, (as a function of the size of the system of linear equations, N) are given for all test problems and when $p = 1, 2, 4, 6$, and 8 processors were used. Clearly, the graphs for different p overlap, which means that the ParaSails preconditioner is independent of the number of processors used. The dashed line illustrates the observed dependence of the number of the iterations on the size of the problem:

$$N_{it} \sim c.\sqrt{N}.$$

In Figure 4, the time for the construction of the preconditioner, t_c, (or setup time) is given as a function of N. Here, we see that:

$$t_c \sim c(p).N,$$

where $c(p)$ is a function depending on the number of processors, p.

In Figure 5, the time for solving the linear system with the PCG, t_s, is given as a function of the size of the system, N. In this case, the observed dependence is:

$$t_s \sim c(p).N^{\frac{3}{2}},$$

which is natural in view of the dependency of N_{it} on N.

Next, we explore how the setup time, t_c, and the solution time, t_s, depend on the number of the processors, p. Since both these times change considerably with the change of N, we plot the results only for the largest three problems. Figures 6 and 7 give t_c and t_s correspondingly as functions of p. Taking into account our earlier observations, we see that for the setup time we have the relation:

$$t_c \sim \frac{c.N}{\sqrt{p}}.$$

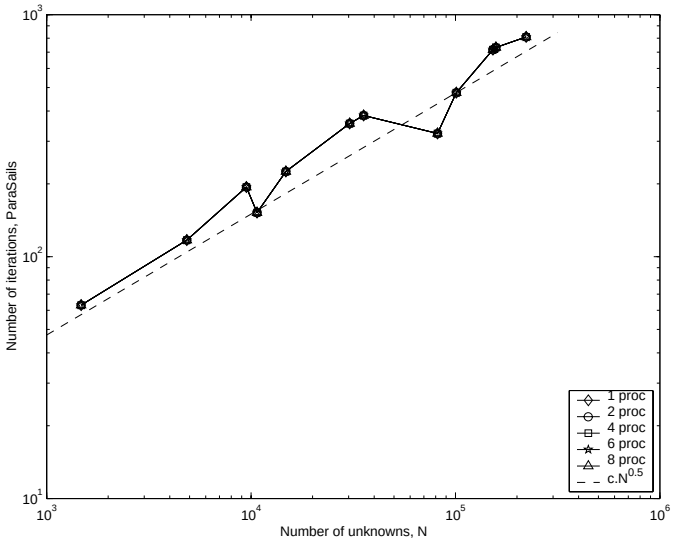

Figure 3: Number of iterations on a multi-processor computer

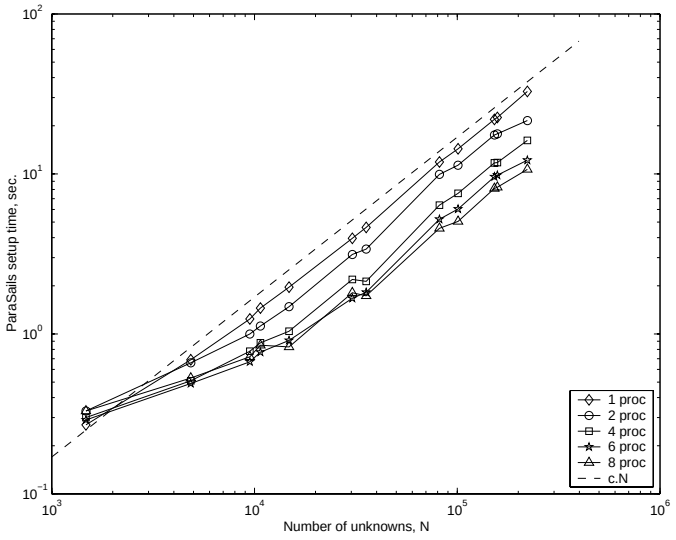

Figure 4: Setup time

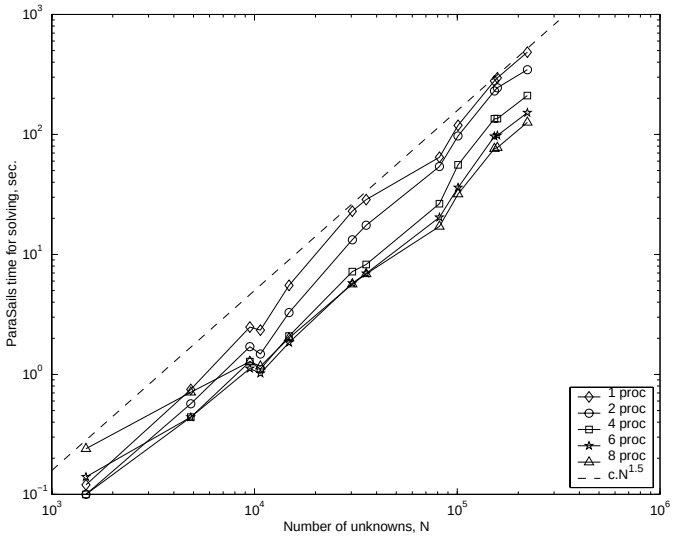

Figure 5: Time for solving

It is more difficult to determine the dependency of t_s on p, but one good choice is:

$$t_s \sim \frac{c.N^{\frac{3}{2}}}{p^{\alpha}}, \quad \text{with} \quad 1/2 < \alpha < 1.$$

5 Computational Results with BoomerAMG

For BoomerAMG, we consider an algebraic problem with the same characteristics as in the case of ParaSails.

The number of iterations in the PCG algorithm is given in Figure 8. The plot is quite chaotic, but still we can observe two facts: the preconditioner (and therefore the number of the iterations) changes a little when the number of the processors used in the computation is changed, and more importantly the number of the iterations is bounded: for any size of the problem, N, and any number of processors, p, it is less than 17.

In Figure 9 we give the setup time, t_c, as a function of the size of the problem, N, and for the different values of p. We can clearly see that the setup time depends linearly on the size of the problem

$$t_c \sim c(p).N,$$

but the dependency on the number of processors is more difficult to determine. For all problems, the fastest setup time was on one processor. Another observation is that for the smallest problem, the time is increasing with the number of the processors, whereas for the largest problem, the time decreases when p increases from 2 to 8, which is what one usually expects. In Figure 11, the setup time as a function of p is given for the

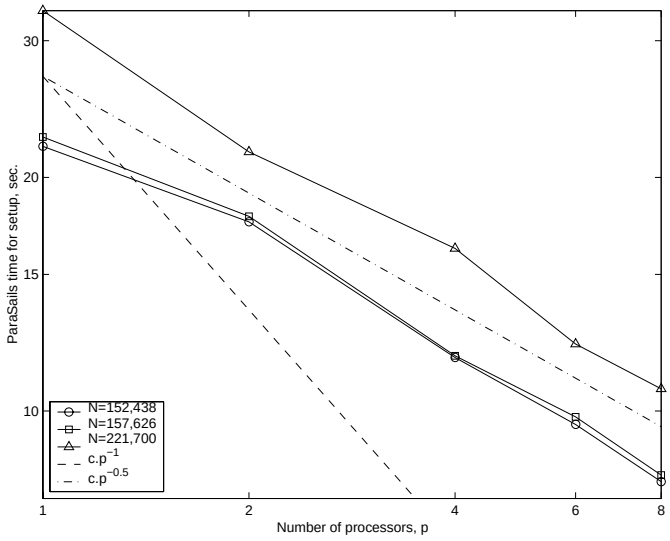

Figure 6: Parallel efficiency of setup

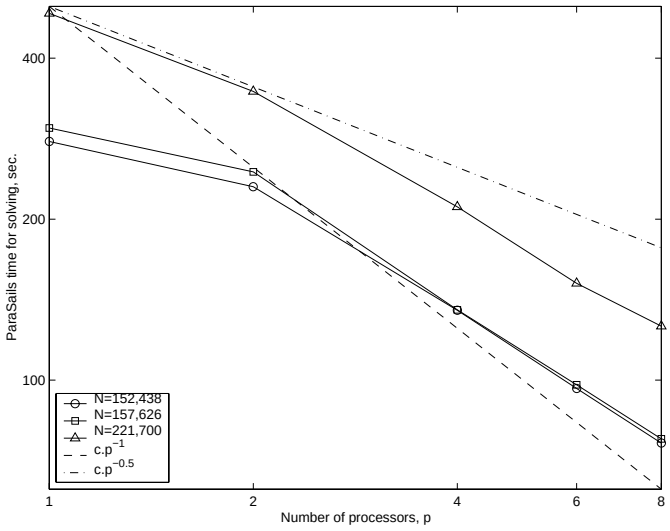

Figure 7: Parallel efficiency of solving

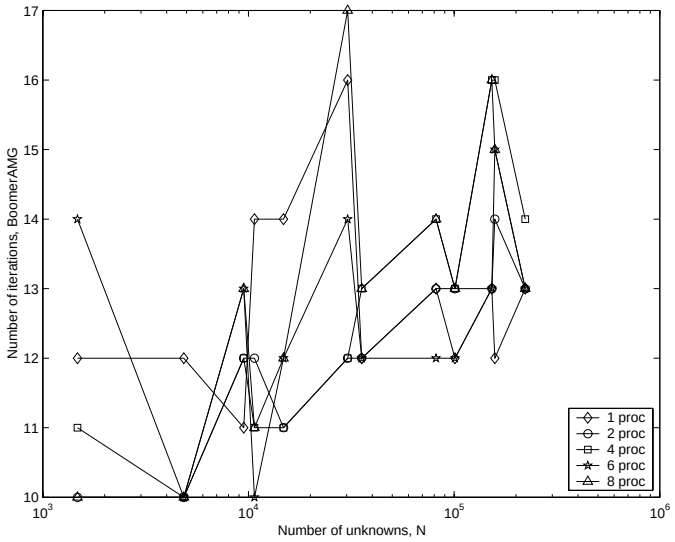

Figure 8: Number of iterations

largest three problems. From these plots, we cannot draw any definite conclusion about the dependency of t_c on p.

The results for the solution time, t_s, are presented in Figure 10, where we can see that the dependency of t_s on the size N is linear

$$t_s \sim c(p).N\,.$$

Once again the dependency on p is different for the different sizes of the corresponding linear system: for the smallest problem the solution time increases with p, whereas for the largest problem, it decreases when p increases. In Figure 12, the dependency of t_s on p for the largest three problems is given, and at least for the largest problem, t_s decreases proportionally to $1/\sqrt{p}$.

6 Comparison of ParaSails and BoomerAMG

We compare the two preconditioners by plotting the ratios (either BoomerAMG/ParaSails or ParaSails/BoomerAMG) of the measured times: setup time, solution time and in addition, the total time, i.e., setup time plus solution time, $t_c + t_s$.

We start with the comparison of the setup times, which is given in Figure 13. We see that for small problems on one processor, BoomerAMG is faster, but with the increasing of the size of the problem and the number of processors used, ParaSails is getting better and in most of the considered cases, its setup time is smaller.

The solution times are compared in Figure 14. The graphics show that for small problems ParaSails is faster, but with the increase of the size of the problem BoomerAMG is

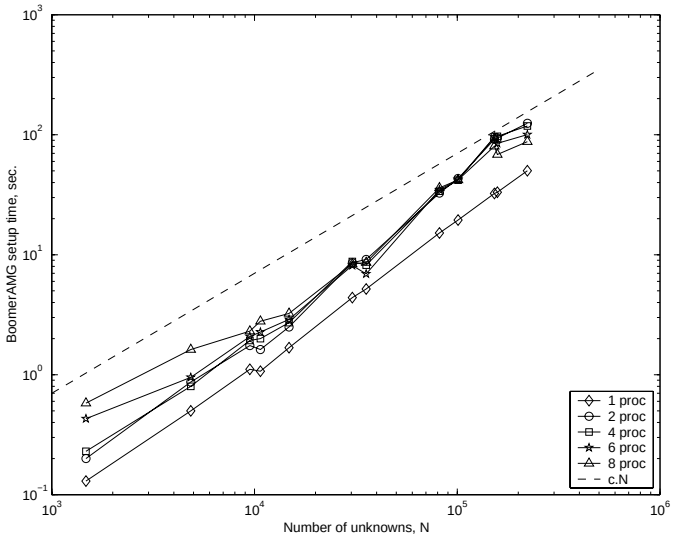

Figure 9: Setup time

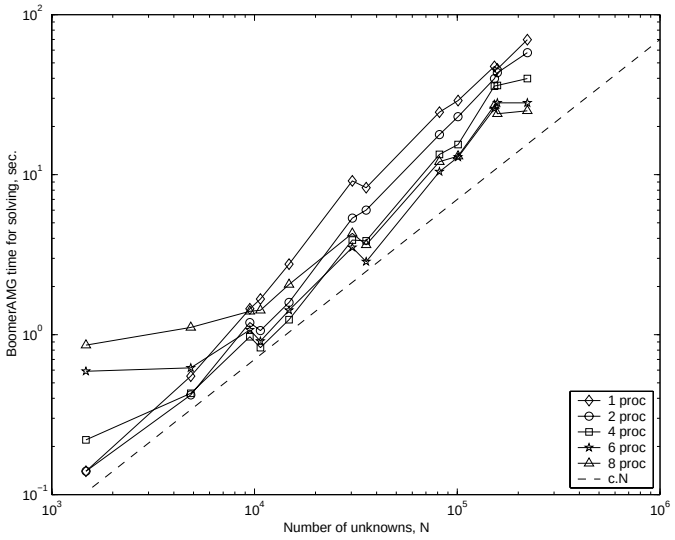

Figure 10: Time for solving

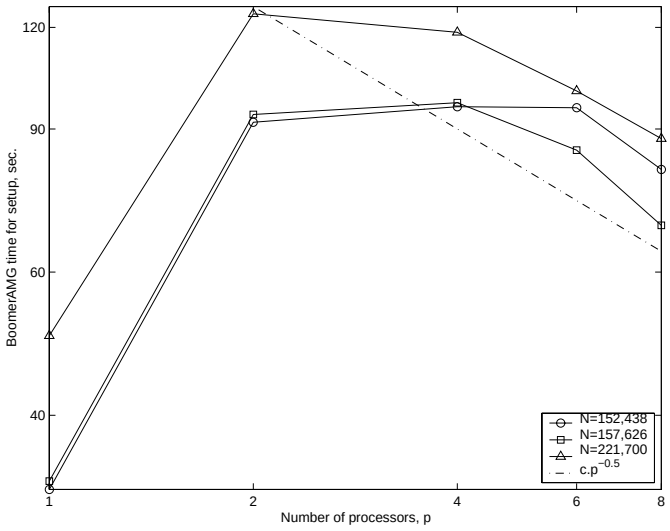

Figure 11: Parallel efficiency of the setup part

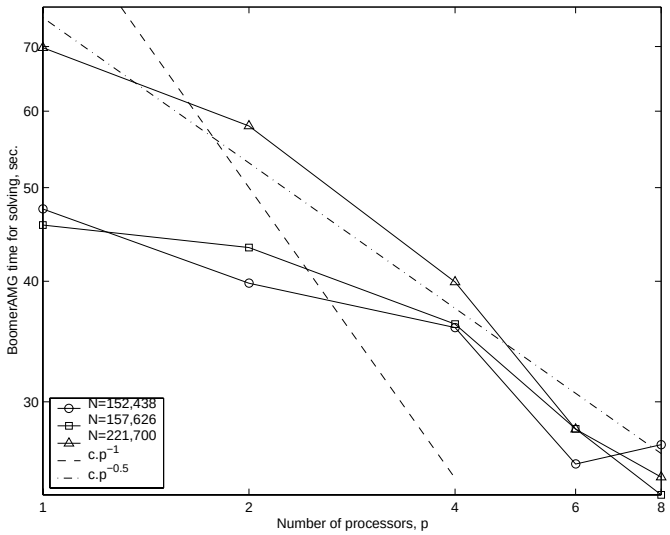

Figure 12: Parallel efficiency of the solution part

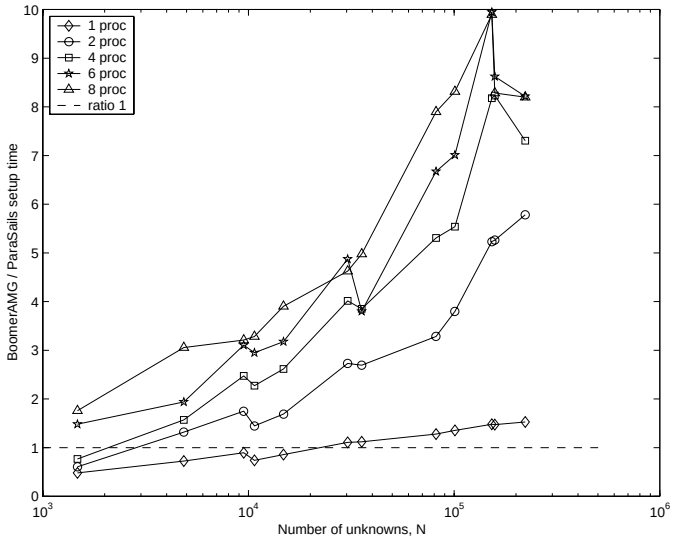

Figure 13: Ratio of setup time (BoomerAMG/ParaSails)

performing better. Another observation is that with the increase of the number of processors, the time for solving with ParaSails decreases faster than that with BoomerAMG, and therefore, as seen in the Figure 14, the plotted ratios decrease.

The comparison of the total (setup + solution) times is presented in Figure 15. For the most of the problems on one and two processors, BoomerAMG performs better than ParaSails. If more processors are used, the latter has the advantage for most of the problems. Exceptions are the largest problem and some other large problems where BoomerAMG is again faster.

In the next three Figures, we present the comparison of the three times as functions of the number of processors. We consider the results only for the largest three problems.

The ratios of the setup times are given in Figure 16. In all cases, ParaSails is better: on one processor it is about 50% faster and with the increase of p its advantage increases and on 6 and 8 processors it is about 8 to 10 times faster.

The comparison of the time for solving is given in Figure 17. Here, BoomerAMG has the advantage in all cases, but the ratios decrease with the increase of the number of processors, p, that is the solution time with ParaSails decreases faster than that with BoomerAMG.

In Figure 18, the ratios of the total times are presented. On one processor, BoomerAMG is about 4 to 5 times faster, but with the increase of p, its big advantage decreases, and on 6 and 8 processors, the times of the two preconditioners are comparable with a difference that is within 30%.

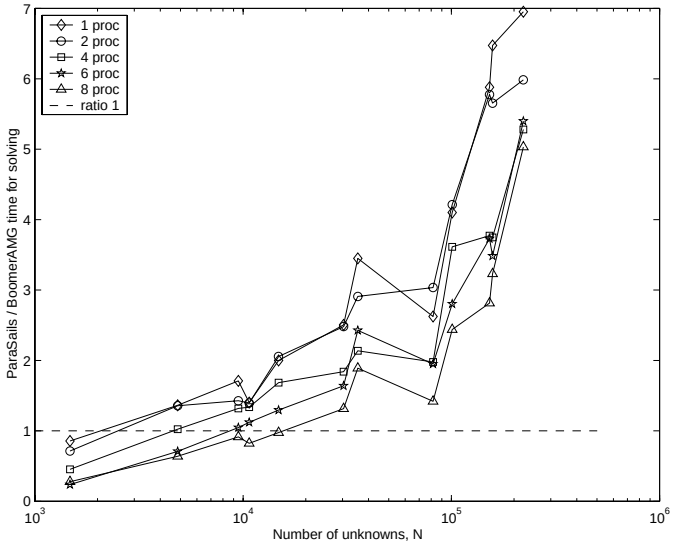

Figure 14: Ratio of time for solving (ParaSails/BoomerAMG)

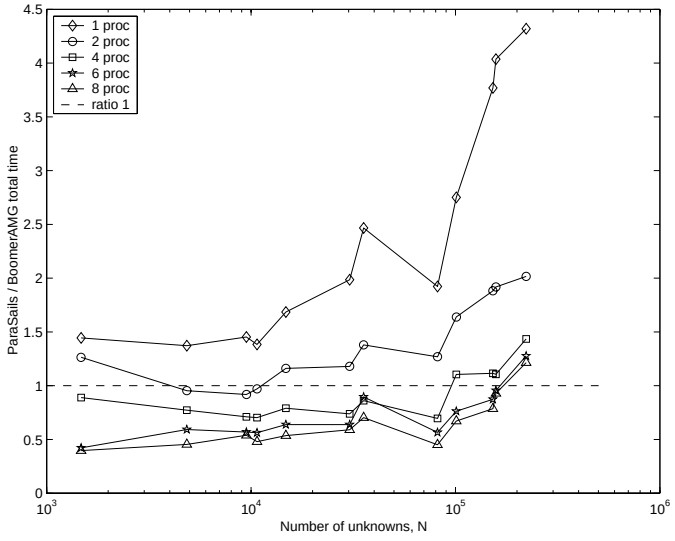

Figure 15: Ratio of total time (ParaSails/BoomerAMG)

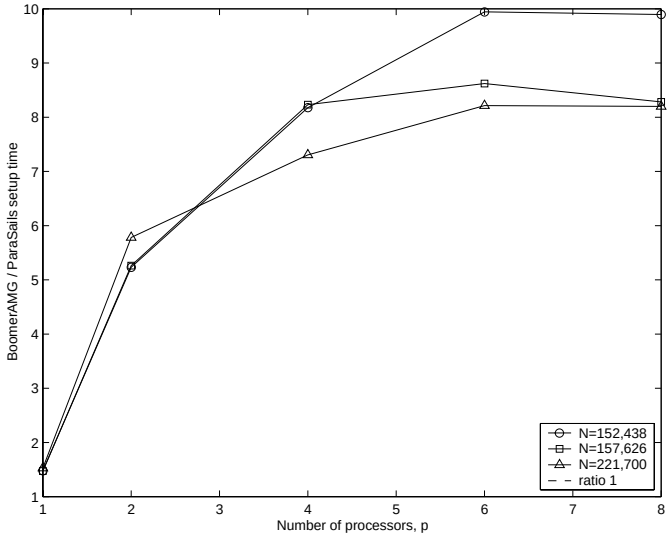

Figure 16: Ratio of setup time (BoomerAMG/ParaSails)

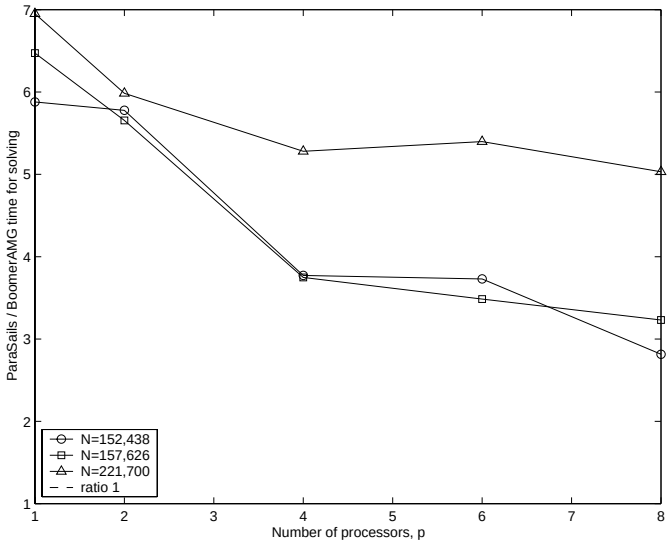

Figure 17: Ratio of time for solving (ParaSails/BoomerAMG)

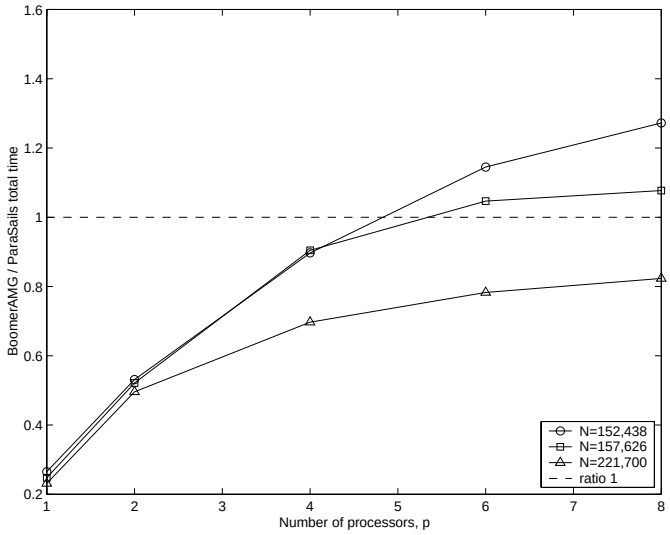

Figure 18: Ratio of total time (BoomerAMG/ParaSails)

7 Sensitivity of the preconditioners to increase of the anisotropy of the permiability coefficient

In this section we consider a slightly modified model problem, namely we increase the anisotropy in the thin layer Ω_2 from 5% to 1%, i. e. we set

$$\mathbf{k}_2 = \mathrm{diag}(0.2, 0.2, 0.002)$$

and we also interchange the Dirichlet boundary conditions on the top of the "well" region Ω_5 and on the bottom of the domain:

$$u_0 = \begin{cases} 2, & \text{on } \Gamma_D \cap \{z = -1\} \text{ (bottom of the domain)} \\ 1, & \text{on } \Gamma_D \cap \{z = 1\} \quad \text{(top of the well).} \end{cases}$$

In Tables 2 – 7 the raw results (on 1, 4, and 8 processors) for both the original model problem and the modified one are given.

The effect of the increase of the anisotropy on the performance of ParaSails and BoomerAMG (in terms of number of iterations, setup time, and solution time) is better illustrated on Figures 19 – 24 where we plot the ratios of the corresponding quantities – "result with 1%" / "result with 5%".

For ParaSails the increase of the number of the iterations is relatively small – it is at most 3.5%, the change in the setup time is between −10% and +15%, and the change in the solution time is between −10% and +17%. Note that the ratios are closer to 1 for the large problems.

problem	iter		setup		solve	
	5%	1%	5%	1%	5%	1%
$0-0$	63	65	0.27	0.28	0.12	0.12
$0-1$	117	120	0.69	0.69	0.75	0.77
$0-2$	194	196	1.24	1.24	2.48	2.51
$0-3$	355	358	3.96	3.93	22.92	22.60
$0-4$	717	719	21.93	21.86	279.41	290.62
$1-1$	152	154	1.45	1.45	2.34	2.32
$1-2$	225	225	1.96	1.96	5.52	5.41
$1-3$	383	389	4.63	4.70	28.60	29.75
$1-4$	732	735	22.55	22.52	296.10	301.01
$2-2$	323	326	11.88	11.85	64.58	66.24
$2-3$	476	484	14.37	14.38	119.06	122.62
$2-4$	809	820	32.80	32.56	485.30	481.69

Table 2: ParaSails, $p = 1$

The relative change in the number of the iterations for BoomerAMG is larger than that for ParaSails – it is between -30% and $+30\%$, but in absolute terms the difference is small – at most 3 iterations. The change in the setup time is in the range -15% – $+10\%$, and it is closer to 0% for the large problems. The change in the solution time is between -30% and $+25\%$ which is mainly due to the change in the iteration numbers.

8 Notations

- Ω – domain
- Ω_i – various subdomains (layers, "well" subregion, etc.)
- Γ – boundary of Ω
- Γ_D – part of the boundary Ω where Dirichlet boundary conditions are specified
- Γ_N – part of the boundary Ω where Neumann boundary conditions are specified
- N – size of the linear system of equations
- N_{it} – number of iterations for solving the linear system
- p – number of processors
- t_c – time for setting up the precondotioner
- t_s – time for solving the linear system

	iter		setup		solve	
problem	5%	1%	5%	1%	5%	1%
$0-0$	12	10	0.13	0.14	0.14	0.12
$0-1$	12	11	0.50	0.50	0.55	0.53
$0-2$	11	11	1.11	1.14	1.45	1.46
$0-3$	16	13	4.39	4.42	9.15	7.70
$0-4$	13	14	32.42	33.07	47.52	51.95
$1-1$	14	11	1.07	1.09	1.67	1.37
$1-2$	14	13	1.68	1.71	2.76	2.54
$1-3$	12	15	5.18	5.10	8.29	9.99
$1-4$	12	13	33.19	33.50	45.74	49.28
$2-2$	13	14	15.18	15.33	24.60	26.30
$2-3$	12	12	19.46	19.36	29.04	29.38
$2-4$	13	14	50.11	50.01	69.81	77.47

Table 3: BoomerAMG, $p = 1$

	iter		setup		solve	
problem	5%	1%	5%	1%	5%	1%
$0-0$	63	64	0.30	0.34	0.10	0.10
$0-1$	117	120	0.51	0.51	0.44	0.45
$0-2$	194	196	0.78	0.74	1.28	1.23
$0-3$	355	358	2.19	2.13	7.18	7.64
$0-4$	717	719	11.72	11.76	135.09	133.82
$1-1$	152	154	0.88	0.89	1.11	1.12
$1-2$	225	225	1.04	1.05	2.09	2.15
$1-3$	383	388	2.13	2.12	8.25	8.43
$1-4$	732	735	11.78	11.91	135.34	136.26
$2-2$	323	326	6.38	6.46	26.47	26.08
$2-3$	476	484	7.56	7.78	55.73	57.53
$2-4$	809	820	16.20	16.17	210.86	213.89

Table 4: ParaSails, $p = 4$

	iter		setup		solve	
problem	5%	1%	5%	1%	5%	1%
$0-0$	11	9	0.23	0.24	0.22	0.20
$0-1$	10	10	0.80	0.80	0.43	0.39
$0-2$	12	12	1.93	1.89	0.97	0.96
$0-3$	12	14	8.79	8.75	3.90	4.36
$0-4$	16	15	95.84	95.56	35.81	33.45
$1-1$	11	11	2.00	1.96	0.83	0.86
$1-2$	11	12	2.72	2.67	1.24	1.34
$1-3$	13	14	8.20	8.09	3.86	4.12
$1-4$	16	15	96.96	95.61	36.11	33.83
$2-2$	14	13	33.86	33.72	13.38	12.47
$2-3$	13	13	41.87	41.57	15.42	15.37
$2-4$	14	14	118.34	117.73	39.93	39.48

Table 5: BoomerAMG, $p = 4$

	iter		setup		solve	
problem	5%	1%	5%	1%	5%	1%
$0-0$	63	65	0.33	0.30	0.24	0.28
$0-1$	117	120	0.53	0.50	0.71	0.65
$0-2$	194	196	0.72	0.68	1.28	1.28
$0-3$	355	358	1.81	1.92	5.66	5.66
$0-4$	717	719	8.11	8.02	76.26	79.04
$1-1$	152	154	0.85	0.89	1.17	1.11
$1-2$	225	225	0.83	0.85	2.01	1.90
$1-3$	383	388	1.73	1.72	6.88	6.84
$1-4$	732	735	8.27	8.10	77.64	77.97
$2-2$	323	326	4.57	4.84	17.13	17.23
$2-3$	476	484	5.05	4.96	31.86	32.07
$2-4$	809	820	10.68	10.65	126.14	130.36

Table 6: ParaSails, $p = 8$

	iter		setup		solve	
problem	5%	1%	5%	1%	5%	1%
$0-0$	10	8	0.58	0.51	0.86	0.66
$0-1$	10	11	1.62	1.46	1.11	1.08
$0-2$	13	13	2.31	2.53	1.40	1.39
$0-3$	17	14	8.37	8.34	4.30	3.53
$0-4$	16	16	80.26	79.63	27.09	27.20
$1-1$	11	12	2.79	2.74	1.42	1.44
$1-2$	12	12	3.24	3.46	2.06	1.99
$1-3$	13	13	8.61	8.72	3.64	3.53
$1-4$	15	15	68.50	68.15	24.03	23.96
$2-2$	14	13	36.10	35.70	12.06	11.27
$2-3$	13	14	41.99	41.50	13.06	13.75
$2-4$	13	16	87.57	86.45	25.07	29.90

Table 7: BoomerAMG, $p = 8$

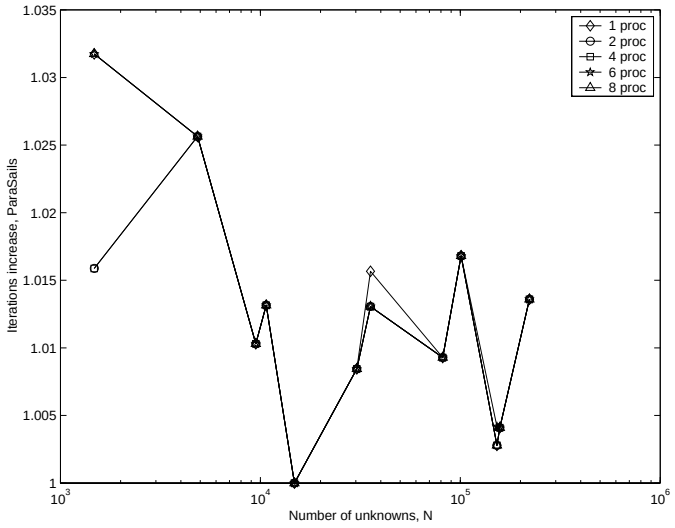

Figure 19: Increase of iterations with ParaSails

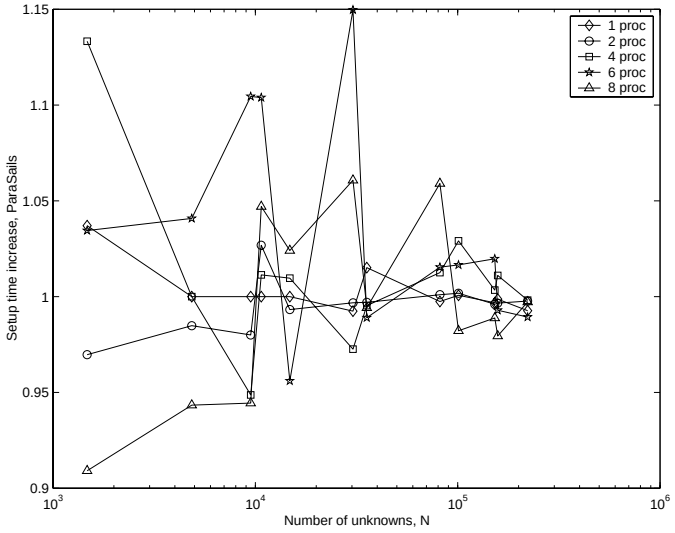

Figure 20: Increase of setup time with ParaSails

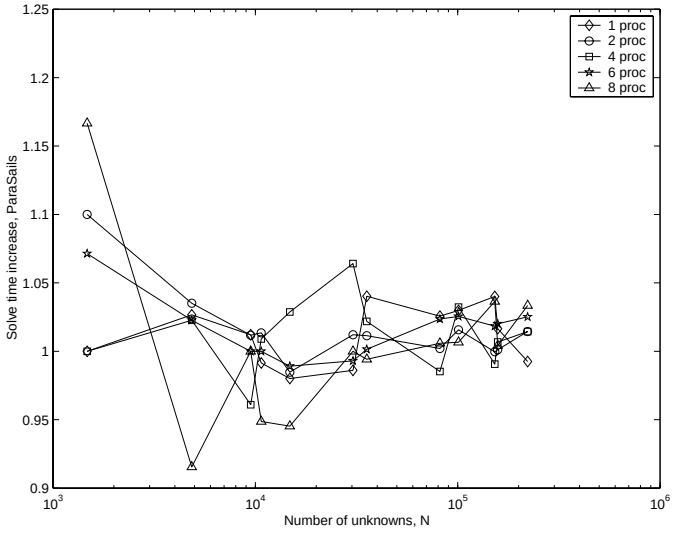

Figure 21: Increase of solve time with ParaSails

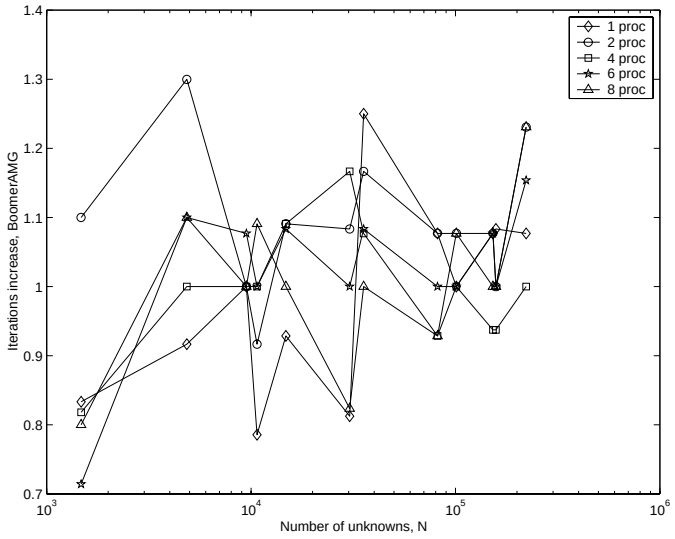

Figure 22: Increase of iterations with BoomerAMG

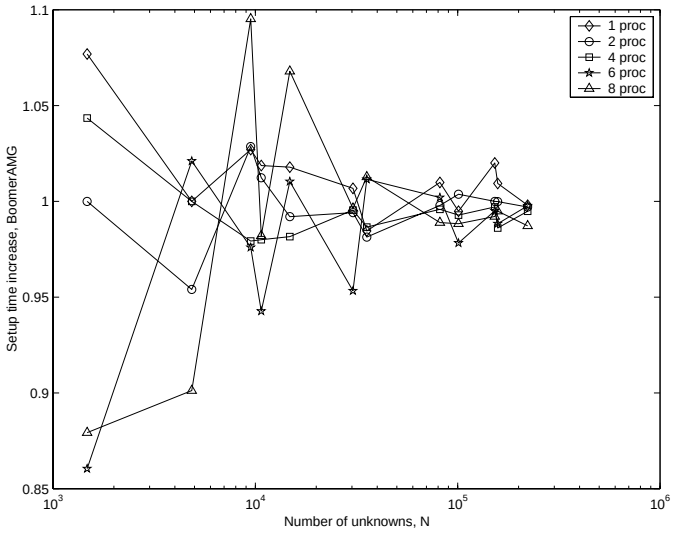

Figure 23: Increase of setup time with BoomerAMG

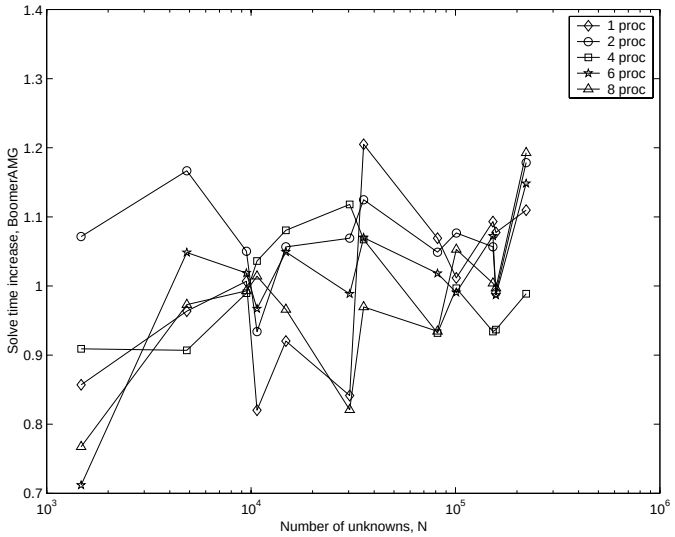

Figure 24: Increase of solve time with BoomerAMG

References

[1] A. Brandt, S. F. McCormick, and J. W. Ruge, *Algebraic multigrid (AMG) for automatic multigrid solutions with application to geodetic computations*, Report, Inst. for Computational Studies, Fort Collins, Colo., October 1982.

[2] A. Brandt, S. F. McCormick, and J. W. Ruge, *Algebraic multigrid (AMG) for sparse matrix equations*, in Sparsity and Its Applications, D. J. Evans, ed., Cambridge University Press, Cambridge, 1984.

[3] Chow, E., *Parallel Implementation and Practical Use of Sparse Approximate Inverses With A Priori Sparsity Patterns*, Int'l J. High Perf. Comput. Appl., 15, pages 56–74, 2001 (`http://www.llnl.gov/CASC/linear_solvers/pubs.html`)

[4] Chow, E., *A Priori Sparsity Patterns for Parallel Sparse Approximate Inverse Preconditioners*, SIAM Journal on Scientific Computing, 21(5), pages 1804–1822, 2000 (`http://www.llnl.gov/CASC/linear_solvers/pubs.html`)

[5] Cleary, A. J., R. D. Falgout, V. E. Henson, J. E. Jones, *Coarse-Grid Selection for Parallel Algebraic Multigrid*, in Proc. of the Fifth International Symposium on Solving Irregularly Structured Problems in Parallel, volume 1457 of Lecture Notes in Computer Science, pages 104–115, New York, 1998. Springer-Verlag. (`http://www.llnl.gov/CASC/hypre/pubs.html`)

[6] Cleary, A. J., R. D. Falgout, V. E. Henson, J. E. Jones, T. A. Manteuffel, S. F. McCormick, G. N. Miranda, and J. W. Ruge, *Robustness and Scalability of Algebraic*

Multigrid, SIAM Journal on Scientific Computing, 21(5), pages 1886–1908, 2000. (http://www.llnl.gov/CASC/hypre/pubs.html)

[7] Henson, V. E. and U. M. Yang, *BoomerAMG: a Parallel Algebraic Multigrid Solver and Preconditioner*, Applied Numerical Mathematics, 41, pages 155–177, 2002. (http://www.llnl.gov/CASC/hypre/pubs.html)

[8] M. Kuhn, U. Langer and J. Schoeberl, *Scientific Computing Tools for 3D Magnetic Field Problems*, SFB-Report No. 99-13, Johannes Kepler Universität Linz, Spezialforschungsbereich F013, 4020 Linz, Austria, 1999.

[9] J. W. Ruge and K. Stüben, *Algebraic multigrid (AMG)*, in: S. F. McCormick, ed., Multigrid Methods, vol. 3 of Frontiers in Applied Mathematics (SIAM, Philadelphia, 1987) 73–130.

[10] J. Schoeberl, *NETGEN - An advancing front 2D/3D-mesh generator based on abstract rules*, Computing and Visualization in Science, **1** (1997), 41–52. (http://www.sfb013.uni-linz.ac.at/~joachim/netgen/)