

FRIEDRICH-ALEXANDER-UNIVERSITÄT ERLANGEN-NÜRNBERG
INSTITUT FÜR INFORMATIK (MATHEMATISCHE MASCHINEN UND DATENVERARBEITUNG)

Lehrstuhl für Informatik 10 (Systemsimulation)



**Comparison of two implementations of AMG-preconditioned CG
considering an electrostatic problem**

Uwe Fabricius

Technical Report 04-1

Comparison of two implementations of AMG-preconditioned CG considering an electrostatic problem

Uwe Fabricius

Lehrstuhl für Informatik 10 (Systemsimulation)
Institut für Informatik
Friedrich–Alexander–Universität Erlangen–Nürnberg
Germany
uwe@cs.fau.de
14th January 2004

Abstract

Physical computational problems often lead to large sparse linear systems of equations, the solution of which is the most time-consuming part of the program. After an appropriate algorithm is chosen for this task, it either can be self-implemented or called from an already existing library. Due to this decision in this report three possibilities are evaluated, using an internal software and the libraries *PETSc* and *BoomerAMG* to solve an electrostatic field model problem.

1 Introduction

Finite element discretisations of partial differential equations usually lead to large sparse linear systems of equations. There exist various well-understood algorithms for solving these systems, that exploit more or less typical properties of the resulting matrices, e.g. sparsity, eigenvalue distribution, diagonal dominance. After an algorithm has been chosen to be suitable for the given problem, it must be either implemented from scratch or simply be plugged in from an already existing software library. This decision must be taken individually regarding the specific requirements of the respective application.

The physical problem ends up in the solution of a linear system containing a sparse, positive definite matrix, for which the preconditioned CG method is well known as a good solver and AMG as an excellent preconditioner. The AMG algorithm consists of two parts, a setup phase and a solution phase. The setup phase is known to be time-consuming and therefore AMG only pays, if the acceleration of the solving phase or the improved convergence of the CG compensate this overhead. Therefore AMG is a particular good choice, if the system must be solved several times with different right hand sides and identical or only slightly modified matrices (for an example see [6]).

In this report the applicability of three implementation variants of the conjugated gradient method, preconditioned with algebraic multigrid (AMG-pc CG) is evaluated, for an electrostatic model problem. An internal implementation (*LAS*) from the department of Sensor Technology, University of Erlangen–Nuremberg, and its combination with the *BoomerAMG* [5] implementation of AMG contained in the *HYPRE* [3] package. A third attempt will be to hand the whole job to the *PETSc* package, that implements its own CG and uses *BoomerAMG* for preconditioning. In our case we are heading for a parallel version of AMG-pc CG. The *BoomerAMG* algorithm is already implemented in parallel, and the *PETSc* library additionally offers a comfortable interface for several Krylov subspace methods and preconditioners, also including the *HYPRE* preconditioners. The goal of this study is to evaluate the suitability of existing software as compared to a possible specialized implementation. As *LAS* is only a serial implementation, only single process comparisons are made. Additionally, a brief evaluation of the parallel performance of *BoomerAMG* will be given.

2 The model problem

The test problem is the solution of the electrostatic field equation inside a domain Ω :

$$-\nabla(\epsilon(\nabla\Phi, \vec{r})\nabla\Phi(\vec{r})) = \rho(\vec{r}), \quad \vec{r} \in \Omega \subset \mathbb{R}^3, \quad (1)$$

with boundary conditions on Γ_D and Γ_N ($\Gamma_D \cup \Gamma_N = \partial\Omega$, $\Gamma_D \cap \Gamma_N = \emptyset$)

$$\begin{aligned} \Phi(\vec{r}) &= \Phi_D(\vec{r}) && \text{on } \Gamma_D \\ -\epsilon(\nabla\Phi, \vec{r})\frac{\partial\Phi(\vec{r})}{\partial n} &= \sigma_N && \text{on } \Gamma_N \end{aligned} \quad (2)$$

where Φ : the electric potential,
 ϵ : the electric permittivity, depending on the electric field strength $\vec{E} = -\nabla\Phi$ and \vec{r} ,
 ρ : the electric volume charge,
 σ_N : the surface charge on Γ_N

If ϵ does not depend on \vec{E} , equation (1) is linear in Φ (Even if $\epsilon = \epsilon(\vec{E})$, we will not get necessarily rid of linear systems, but have to solve them several times in fixed-point-schemes for non-linear equations [6]). The linear algebraic system arises from the variational formulation of (1) in the Sobolev space $\mathbb{H}^1(\Omega)$

$$\text{Find } \Phi \in \mathbb{H}^1(\Omega) : a(\Phi, \varphi) = \langle Q, \varphi \rangle \quad \forall \varphi \in \mathbb{H}^1(\Omega), \quad (3)$$

with corresponding symmetric, non-negative bilinear form $a(\cdot, \cdot) : \mathbb{H}^1(\Omega) \times \mathbb{H}^1(\Omega) \rightarrow \mathbb{R}$ and inner product $\langle \cdot, \cdot \rangle : \mathbb{H}^1(\Omega)^* \times \mathbb{H}^1(\Omega) \rightarrow \mathbb{R}$. On a discretized Ω , Ω_M (with suitable mesh M), $\mathbb{H}^1(\Omega_M)$ is finite dimensional, and in an instantiation of (3) with functions φ spanning a basis of $\mathbb{H}^1(\Omega_M)$ the finite element isomorphism ensures the equivalence of (3) to the linear system of equations

$$\begin{aligned} Ax = b, \quad A &\in \mathbb{R}^{n \times n}, \quad n = \dim(\mathbb{H}^1(\Omega_M)) \\ x, b &\in \mathbb{R}^n \end{aligned} \quad (4)$$

A is sparse and, in our case, symmetric and strictly positive definite. Here Ω is simply an air-filled block domain with $\Phi = 0$ on a part of the bottom surface (Γ_{D_0}). Only a towering bar is cut away, of which the bottom surface Γ_{D_1} lays on potential $\Phi = 1$. On the remaining boundary of Ω we assume homogeneous Neumann boundary conditions. As Ω is filled with air and does not contain any charge, $\epsilon \equiv id_{\mathbb{R}^3}$, and $\rho \equiv 0$, and (1) simplifies to the Laplace equation

$$\Delta\Phi(\vec{r}) = 0 \quad (5)$$

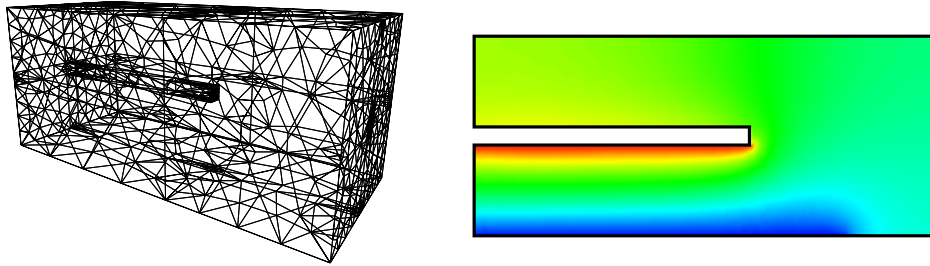


Figure 1: mesh of Ω (898 nodes), color-coded cutplane through the solution of (5)

3 Algorithm and implementations

3.1 AMG-pc CG

As we have seen in the previous section we have to solve the large linear system (4) $Ax = b$, with the sparse, symmetric, positive definite matrix A , and therefore AMG-preconditioned conjugated

gradient is an appropriate algorithm for its solution. The preconditioner step consists of a single V-cycle of AMG. For more details see [4].

```

k = 0
r0 = b - Ax0
while rk > ε do
  zk = AMGstep(A, rk)
  k = k + 1
  if k == 1 then
    pk = zk-1
  else
    βk =  $\frac{r_{k-1}^T z_{k-1}}{r_{k-2}^T z_{k-2}}$ 
    pk = zk-1 + βkpk-1
  fi
  αk =  $\frac{r_{k-1}^T z_{k-1}}{p_k^T A p_k}$ 
  xk = xk-1 + αkpk
  rk = rk-1 - αkA pk
od
x = xk

```

The AMG–algorithm builds a hierarchy of linear systems which allows to approximate the solution of the residual equation for an approximate solution u_h of the exact x_h on hierarchy level h

$$A_h e_h = b_h - A_h u_h, \quad e_h := x_h - u_h \quad . \quad (6)$$

The levels are created by finding an appropriate interpolation operator I_H^h between the coarse level H and the fine level h and defining the coarse level operator A_H as GALERKIN product

$$A_H := (I_H^h)^T A_h I_H^h \quad (7)$$

In a two–level hierarchy the correction e_h in equation (6) is calculated as

$$\begin{aligned}
\tilde{u}_h &\longleftarrow S_h^{\nu_1}(A_h, u_h, b_h) \\
e_h &\longleftarrow I_H^h A_H^{-1} (I_H^h)^T (b_h - A_h \tilde{u}_h) \\
\tilde{\tilde{u}}_h &\longleftarrow S_h^{\nu_2}(A_h, \tilde{u}_h + e_h, b_h)
\end{aligned} \quad (8)$$

Here S_h denotes a smoothing operator, applied ν_1 and ν_2 times in a pre- and postsmoothing step, respectively. Assume that S_h reduces the error $e_h = A_h^{-1} b_h - u_h$ like $e_h \leftarrow E_h e_h$, with operator E_h . If $\text{img}(E_h) = \text{img}(I_H^h)$, the coarse–level correction step yields the exact solution. If the inversion of A_H is too costly, the correction scheme is applied recursively with a zero initial guess u_H . In a point–based coarsening the transfer operator I_H^h is set up by first choosing a set of so called *coarse* variables C_h as a subset of all unknowns Ω_h on level h , then defining appropriate weights $[I_H^h]_{ij}$ for the interpolation of the *fine* variable F_{hi} from *coarse* variables C_{hj} in the neighbourhood $N_{hi} := \{j \in \Omega_h \mid [A_h]_{ij} \neq 0\}$ of variable i . Note that $F_h := \Omega_h \setminus C_h$. (Multigrid [9, 8], AMG [7, 8]).

3.2 Implementations

The linear system of equations is set up by *CFS*, a finite element software developed at the department for sensor technology (*LSE*), university Erlangen–Nuremberg, for the calculation of coupled field systems. Dirichlet boundary conditions are implemented via penalty additions in equation (4). To solve the linear system, *CFS* uses a linear algebra framework (*LAS*), that has also been implemented at the *LSE*. *LAS* itself provides a AMG–preconditioned CG method. The second implementation under consideration is again the *LAS*–CG method, but preconditioned with the serial version of scalar *BoomerAMG* [5] from the *HYPRE* package [3] of high performance preconditioners. In a third approach we use the CG–algorithm of the *PETSc* library [1, 2], which again uses the *BoomerAMG*–routines for preconditioning.

In all cases we employ point wise Gauss–Seidel as smoother S_h and the set of *coarse* variables is chosen by the standard RUGE–STÜBEN coarsening [7]. The crucial differences of the compared algorithms consist of the way to define the interpolation weights $[I_H^h]_{ij}$. In *BoomerAMG* the interpolation is built as proposed by Ruge and Stüben [7] (the level index h is omitted, if not needed for clearness). To allow a compact notation we introduce the bijective mapping $\Theta : C_h \mapsto \Omega_H$ of fine level indices of coarse variables $\in C_h$ to their corresponding indices in Ω_H . For $i \in \Omega_h$ and $j \in C_h$

$$[I_H^h]_{i\Theta(j)} = \begin{cases} 0 & ; \quad a_{ij} = 0 \\ \left\{ \begin{array}{l} \delta_{ij} & ; \quad i \in C_h \\ w_{ij} & ; \quad i \in F_h \end{array} \right\} & ; \quad a_{ij} \neq 0 \end{cases} \quad (9)$$

with

$$w_{ij} = \begin{cases} -\frac{a_{ij} + c_{ij}}{a_{ii} + c_{ii}} & ; \quad j \in N_i \cap C_h \\ 0 & ; \quad \text{else} \end{cases} \quad (10)$$

and

$$c_{ij} = \sum_{\substack{k \notin N_i \cap C_h \\ k \neq i}} \frac{a_{ik}a_{kj}}{a_{ki} + \sum_{l \in N_i \cap C_h} a_{kl}} \quad (11)$$

In contrast *LAS*–AMG uses a simple averaging as interpolation:

$$w_{ij} = \frac{1}{\sum_{k \in C_h \cap N_{h_i}}} \quad (12)$$

Obviously this interpolation will be on the one hand cheaper in computation, on the other hand might result in a poorer interpolation and therefore harm the convergence of the AMG–cycle. In addition to this, *BoomerAMG* sparsens I_H^h by deleting entries lower than a tolerance value.

4 Experiments and Results

The domain Ω was meshed with different resolutions, and the system was solved on a Linux PC with an Intel PIV CPU, running at 2.4 GHz.

	AMG setup	AMG step	# iter	solve	setup+solve	$\ r\ _2$
n = 48919						
<i>LAS</i> –CG, <i>LAS</i> –AMG	1.4	0.02	11	0.4	1.8	5.0e-19
<i>LAS</i> –CG, <i>BoomerAMG</i>	2.4	0.2	3	0.9	3.3	1.4e-19
<i>PETSc</i> –CG, <i>BoomerAMG</i>	2.4	0.2	3	0.9	3.3	1.2e-19
n = 93859						
<i>LAS</i> –CG, <i>LAS</i> –AMG	3.3	0.1	14	1.0	4.3	2.9e-19
<i>LAS</i> –CG, <i>BoomerAMG</i>	5.7	0.4	3	1.8	7.5	1.1e-19
<i>PETSc</i> –CG, <i>BoomerAMG</i>	5.7	0.4	3	1.8	7.5	9.0e-20
n = 158852						
<i>LAS</i> –CG, <i>LAS</i> –AMG	5.0	0.9	16	1.8	6.8	2.7e-19
<i>LAS</i> –CG, <i>BoomerAMG</i>	11.2	0.8	3	3.2	14.4	1.2e-19
<i>PETSc</i> –CG, <i>BoomerAMG</i>	11.1	0.8	3	3.2	14.4	1.1e-18
n = 253047						
<i>LAS</i> –CG, <i>LAS</i> –AMG	8.9	0.3	17	6.6	15.5	3.0e-19
<i>LAS</i> –CG, <i>BoomerAMG</i>	21.8	1.4	3	5.8	27.6	1.5e-19
<i>PETSc</i> –CG, <i>BoomerAMG</i>	21.7	1.4	3	5.8	27.5	1.2e-18

Table 1: timing results [seconds] for different mesh resolutions of Ω , \mathbf{n} = number nodes in the mesh. (compiler: gcc-2.95, optimization: -O3)

Table (1) shows the runtimes (in seconds), needed for the *AMG setup* phase, one AMG V-cycle (*AMG step*), the whole solution phase of the AMG-pc CG (*solve*), and the sum of the setup and the solution process (*setup+solve*). It also contains the number of CG iterations ($\# \text{ iter}$) needed to reach the tolerance for the L_2 norm of the residual ($\|r\|_2$). This must be computed carefully, since the introduction of penalty terms in *LAS* might otherwise spoil the comparability of the values. The table shows the significant advantage of the simple *LAS*-interpolation in the setup phase. A modified interpolation algorithm in *BoomerAMG* that creates the same transfer operator as *LAS* results in an equivalently fast setup phase (7.1 s, for $n = 253047$) and an accelerated AMG step (0.8 s), whereby number of CG-iterations increases accordingly to 21. The additional introduction of the *PETSc* interface does not harm the performance, only the conversion of the matrix into the *BoomerAMG*-format is slightly faster (0.31 s) than into the *PETSc* data structure (0.44 s). The CG-algorithms itself of *LAS* and *PETSc* turn out to be equivalent.

The convergence of the both algorithms deteriorates if we simulate uniform anisotropy by scaling the x-coordinate of all elements by a factor α . For $\alpha = 10^5$ the number of iterations increases from 17 up to 165 (*LAS*-CG, *LAS*-AMG) and from 3 up to 23 (*LAS*-CG, *BoomerAMG*), but the RUGE-STÜBEN interpolation does not show a significant superiority.

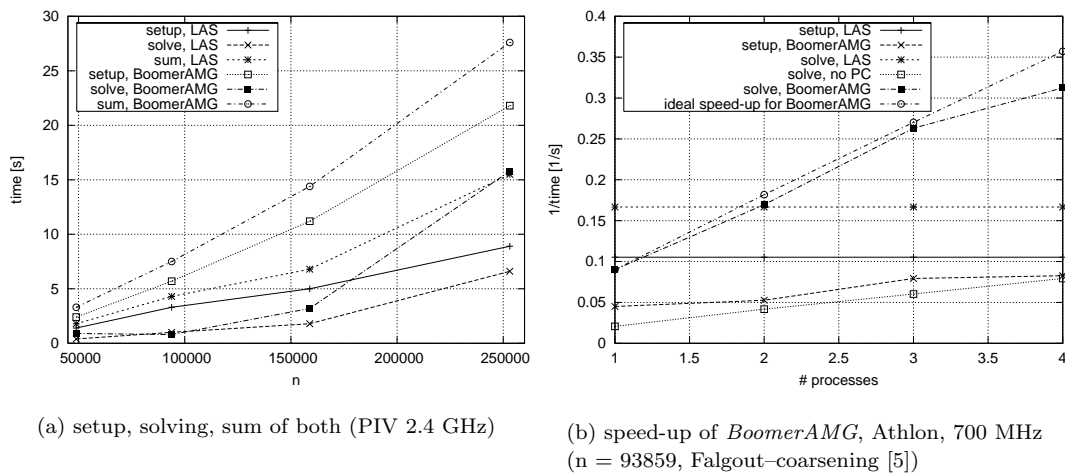


Figure 2: graphical excerpt of table 1 and speed-up of *BoomerAMG*

Figure 2(b) presents an evaluation of the parallel speed-up of the *BoomerAMG* algorithm, tested on a cluster of four identical Linux PCs, based on an AMD Athlon CPU, running at 700 MHz. As these PCs do not contain enough memory to store the largest grid, these tests are realized with a smaller one (note that reciprocal time is plotted against the number of processes).

5 Conclusions

Concerning our model problem *BoomerAMG* emerged as an efficient implementation of an AMG preconditioner that could be integrated easily into existing code. But its standard interpolation is too complicated and costly, so that its overall performance is limited by the setup phase without gaining anything for the solving step. Although the RUGE-STÜBEN interpolation results in an superior fast convergence, this cannot compensate the increased runtime per V-cycle. Even in case of uniformly strong anisotropic elements, the simple *LAS* interpolation does not lose its advantage. Of course, it must be admitted that the POISSON equation of our model problem yields a quite well-natured linear system. But in particular in this context it seems that RUGE-STÜBEN interpolation involves an unreasonably large overhead, and a loss of convergence per V-cycle is acceptable in order to accelerate setup and V-cycle. This should be kept in mind, because the POISSON equation is often solved in physical problems (pressure correction in NAVIER-STOKES, electromagnetics, acoustics, heat conduction, ...). As can be seen in figure 2(b) *BoomerAMG* solved the model problem on two processors in the same time that *LAS* used on one processor.

But we must be careful with a transfer of these serial results to a parallel program. As each V -cycle contains, independently of the chosen interpolation strategy, a certain overhead of communication, a more complicated interpolation might be the more efficient choice, since it diminishes the number of V -cycles and therefore reduces the amount of communication per solution phase. Further investigations will be necessary to determine the best strategy, depending on the problem and the implementation in parallel. In any case, the simple use of *BoomerAMG* cannot be a satisfactory approach for the parallelization of the *LAS* code.

References

- [1] Satish Balay, Kris Buschelman, William D. Gropp, Dinesh Kaushik, Matt Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 2.1.5, Argonne National Laboratory, 2002.
- [2] Satish Balay, Kris Buschelman, William D. Gropp, Dinesh Kaushik, Matt Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc home page. <http://www.mcs.anl.gov/petsc>, 2001.
- [3] Robert D. Falgout and Ulrike Meier Yang. hypre: a Library of High Performance Preconditioners. In *Computational Science – CARS 2002 Part III*, pages 632–641. Springer-Verlag, 2002. Also available as Lawrence Livermore National Laboratory technical report UCRL-JC-146175.
- [4] G.H. Golub and C.F. Van Loan. *Matrix Computations*. The John Hopkins University Press, third edition, 1996.
- [5] Van Emden Henson and Meier Yang Ulrike. BoomerAMG: a Parallel Algebraic Multigrid Solver and Preconditioner. *Applied Numerical Mathematics*, 41:155–177, 2002. Also available as Lawrence Livermore National Laboratory technical report UCRL-JC-141495.
- [6] Manfred Kaltenbacher and Stefan Reitzinger. Algebraic Multigrid for Static Nonlinear 3D Electromagnetic Field Computations. Technical report, Department of Sensor Technology, University of Erlangen, Johannes Kepler University Linz, April 2000. SFB F013.
- [7] John Ruge and Klaus Stüben. Efficient Solution of Finite Difference and Finite Element Equations by Algebraic Multigrid (AMG). *Arbeitspapiere der GMD*, 89, GMD, 1984.
- [8] Ulrich Trottenberg, Cornelis Oosterlee, and Anton Schüller. *Multigrid*. Academic Press, 2001.
- [9] Hackbusch Wolfgang. *Iterative Lösung großer schwachbesetzter Gleichungssysteme*, 1993.