N94-21474

LAYOUT OPTIMIZATION WITH ALGEBRAIC MULTIGRID METHODS*

5/0-64
197570
p. 15

Hans Regler and Ulrich Rüde
Institut für Informatik
Technische Universität München
Arcisstr. 21, D-8000 München 2, Germany
e-mail: regler/ruede @informatik.tu-muenchen.de

SUMMARY

Finding the optimal position for the individual *cells* (also called functional *modules*) on the chip surface is an important and difficult step in the design of integrated circuits. This paper deals with the problem of *relative placement*, that is the minimization of a quadratic functional with a large, sparse, positive definite system matrix. The basic optimization problem must be augmented by constraints to inhibit solutions where cells overlap. Besides classical iterative methods, based on *conjugate gradients* (CG), we show that *algebraic multigrid methods* (AMG) provide an interesting alternative. For moderately sized examples with about 10000 cells, AMG is already competitive with CG and is expected to be superior for larger problems. Besides the classical "multiplicative" AMG algorithm where the levels are visited sequentially, we propose an "additive" variant of AMG where levels may be treated in parallel and that is suitable as a preconditioner in the CG algorithm.

THE PLACEMENT PROBLEM IN INTEGRATED CIRCUIT LAYOUT OPTIMIZATION

In this paper we present some results of research in *algebraic multigrid methods* (AMG). Our interest in these methods is motivated by an application arising in the layout optimization for integrated circuits. Modern integrated circuits consist of several millions of transistors. The layout optimization for an integrated circuit is usually based on grouping the transistors into *cells* (also called functional *modules*) like NAND/NOR-gates. This leads to the problem of finding the optimal location (placement) for hundreds of thousands of such cells on the chip surface. The goal of this optimization is to find a design that uses as little surface area as possible and that minimizes the time delay caused by long connections between cells. Short connections are desirable, because they permit higher clock rates and thus faster chips.

Generally, finding the *optimal* layout for a given functional description of an integrated circuit is a formidable task. From a mathematical point of view the problem begins with the modeling of the above informal optimality conditions. Furthermore, cells cannot be positioned freely on the chip surface. Clearly, they must not overlap, so that we must consider their individual size and shape. Additionally, the manufacturing process introduces constraints on the locations permitted.

*This research is supported by the SFB 0342 of the Deutsche Forschungsgemeinschaft (DFG)

Our research is done in the context of GORDIAN, a state-of-the-art layout synthesis package[†] that has been developed at the *Institute for Electronic Design Automation, Technische Universität München*, see Kleinhans, Sigl, and Johannes [1, 2]. Within this package, the placement problem is handled by breaking it into two separate steps, the *relative placement* and the *final placement*.

The purpose of the relative placement step is to provide a good initial guess for the final placement by finding the *global* optimum of a sequence of problems with a simplified optimality condition. After relative placement, only *local effects* are considered in the final placement, much simplifying the task of positioning a cell within the constraints.

The global relative placement optimization is based on a *force model* where connections between cells are weighted according to their Euclidean length. Modules are connected by *signals* that can be interpreted as abstract connections of the cells. Implicitly, the positions of the signals are also subject to the optimization process.

The functional in the relative placement optimization is quadratic with a positive definite M-matrix C whose entries represent the graph of connections between the cells and signals. Mathematically, the problem can be stated as

$$\min_{x \in \mathbf{R}^n} x^T C x - 2b^T x, \tag{1}$$

where $x, b \in \mathbf{R}^n$, and $C \in \mathbf{R}^{n \times n}$.

An unaugmented minimization of (1), however, tends to cluster the cells in the center of the chip. This is unrealistic, because there is too much overlap between the cells so that the final placement step would not be able to find acceptable positions for the cells. Therefore, the optimization is augmented with linear constraints of the form

$$Ax = d, \tag{2}$$

that specify *centers of gravity* for groups of cells. These constraints are introduced successively by recursively *partitioning* the cells into groups with equal overall cell surface area, and assigning their center of gravity to subdomains of the chip surface. This is illustrated in Figs. 1 and 2, where the results after five successive partitioning steps with 1, 2, 4, 8, and 16 constraints are shown; see also Regler [3].

THE AMG ALGORITHM OF RUGE AND STÜBEN

Our application leads to a large, sparse, positive definite system of equations, which is in no way related to a partial differential equation. A typical matrix structure is displayed in Figure 4. The placement optimization program GORDIAN presently uses a preconditioned conjugate gradient method for the minimization in the relative placement step. We now study the suitability of algebraic multigrid as an alternative. Classical, *geometric* multigrid methods have very

[†]In fact, GORDIAN compared favorably at the 1992 "TimberWolf Hunt", an international competition for placement algorithms

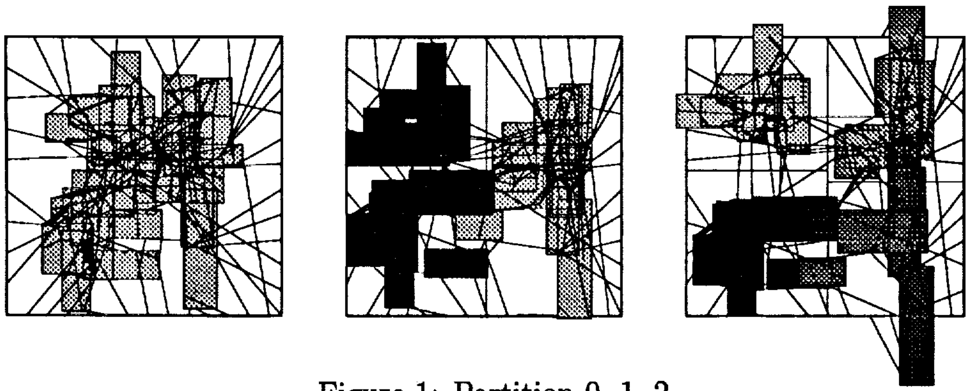
Figure 1: Partition 0, 1, 2

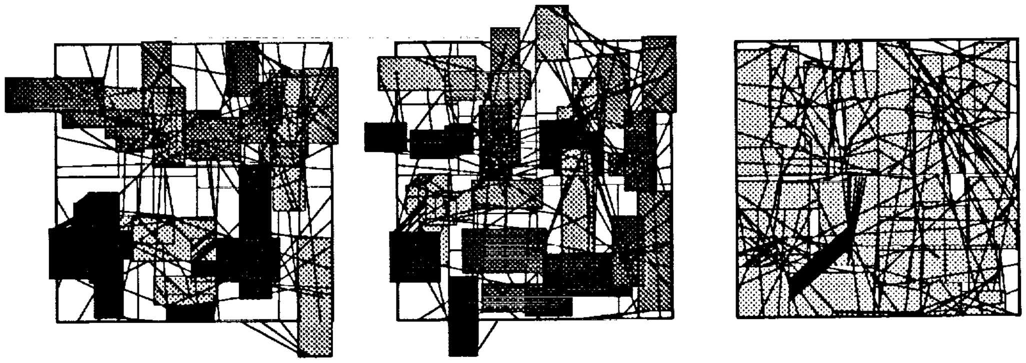
Figure 2: Partition 3, 4, and final placement

successful for solving (1) when the matrix originates from the discretization of elliptic partial differential equations. Here, however, we need an *algebraic multigrid method* that works as a black-box solver given only the system matrix C and the right hand side vector b.

In general, the key to multigrid methods is a family of smaller, *coarse level* systems

$$\min_{x^k \in \mathbf{R}^{n^k}} (x^k)^T C^k x^k - 2(b^k)^T x^k, \tag{3}$$

for $k = 1, 2, \ldots, K$, where the superscript denotes the level and where $x^k, b^k \in \mathbf{R}^{n^k}$, and $C^k \in \mathbf{R}^{n^k \times n^k}$, and where the dimensions n^k form a decreasing sequence

$$n^1 > n^2 > \cdots > n^K \geq 1.$$

The original system coincides with the first and largest problem in the family, $C = C^1$, $b = b^1$.

For an AMG algorithm, the sequence of matrices C^k must be constructed algebraically. The smaller C^k are computed successively by selecting a subset of the unknowns of the level $k-1$ system and by evaluating the *strength of the connections* between the unknowns in C^{k-1}. The basis for this paper is the AMG method of Ruge and Stüben [4] that uses the assumptions

$$\begin{aligned} C = (\gamma_{ij})_{1\leq i,j\leq n} &\ \text{symmetric positive definite,} \\ \gamma_{ij} \leq 0 &\quad \text{for } 1 \leq i,j \leq n,\ i \neq j, \\ \textstyle\sum_{j=1}^{n} \gamma_{ij} \geq 0 &\quad \text{for } 1 \leq j \leq n. \end{aligned} \tag{4}$$

With (4) the effect of Gauss-Seidel iterations on C is well understood and can be used to guide the construction of the coarser level systems C^k for $k = 2, 3, \ldots, K$.

AMG methods were first introduced in the early eighties by Brandt, McCormick, and Ruge [5, 6, 7]. AMG is necessarily less efficient than highly specialized geometric multigrid solvers for elliptic problems on uniform rectangular grids. However, for more complicated cases with complex domains, AMG has been shown to behave quite favorably in terms of operation count and CPU time. AMG also works for problems where geometric multigrid methods are impossible to design. In this paper we will show that AMG works very satisfactorily even for the matrices in chip design.

The generality of AMG must be paid for by a setup phase that may take 80% or more of the overall time. This setup is needed to construct the sequence of reduced matrices C^k together with appropriate transfer operators from level k to level $k+1$.

$$I_k^{k+1} : \mathbf{R}^{n^k} \to \mathbf{R}^{n^{k+1}}. \tag{5}$$

This step is quite expensive and contains code that does not vectorize or parallelize well.

We will briefly review the AMG algorithm, as introduced by Ruge and Stüben [4, 8]. The most interesting part may be the setup routine to build the family of systems (3) with the transfer operators (5).

The matrices C^k are constructed such that each of the unknowns x_i^k on level k, $(k > 1)$, will represent an unknown on the next finer level $k-1$. The level $k-1$ unknown represented by x_i^k on level k is denoted by $x_{j(i)}^{k-1}$, the *corresponding* finer level unknown. This naturally partitions the unknowns on each level (except the coarsest) into those that correspond to a coarser level unknown, and those that do not. These will be called the C- and F-unknowns of a level, respectively. The partitioning is performed in two phases on each level. At the beginning of the first phase, the unknowns with strictly diagonal dominant matrix rows are determined. These unknowns are not restricted to a coarser level.

SETUP PHASE I:

```
1. Set $F_d = \emptyset$
2. $\forall i \in \Omega$ : If $\bar{\alpha}\gamma_{ii} \geq \sum_{j \neq i} | \gamma_{ij} |$ then set $F_d = F_d \cup \{i\}$ endif
3. Set $C = \emptyset$ and set $F = \emptyset$
4. While $C \cup F \neq (\Omega \setminus F_d)$ do
       Pick $i \in (\Omega \setminus F_d) \setminus (C \cup F)$ with maximal $| S_i^T | + | S_i^T \cap F |$
       If $| S_i^T | + | S_i^T \cap F | = 0$
           then set $F = (\Omega \setminus F_d) \setminus C$
           else set $C = C \cup \{i\}$ and set $F = F \cup (S_i^T \setminus C)$;
       endif
```

Next, in a second phase the final C-point choice is made.

SETUP PHASE II:

```
1. Set $T = \emptyset$
2. While $T \subset F$ do
       Pick $i \in F \setminus T$ and set $T = T \cup \{i\}$
       set $\tilde{C} = \emptyset$ and set $S_i^I = S_i \cap C$
       set $P = S_i \setminus S_i^I$
       While $P \neq \emptyset$ do
           Pick $j \in P$ and set $P = P \setminus \{j\}$
           If $d(j, S_i^I) \leq \beta d(i, \{j\})$
               then if $| \tilde{C} | = 0$
                   then set $\tilde{C} = \{j\}$ and set $S_i^I = S_i^I \cup \{j\}$
                   else set $C = C \cup \{i\}$, set $F = F \setminus \{i\}$ and Goto 2
               endif
           endif
       set $C = C \cup \tilde{C}$, set $F = F \setminus \tilde{C}$
3. (Set $F = F \cup F_d$)
```

In these algorithms we use

$$d(i,S) := \frac{1}{\max\limits_{k \neq i}\{-\gamma_{ik}\}} \sum_{j \in S} -\gamma_{ij}$$

and $S_i := \{j \in N_i \mid d(i,\{j\}) \geq \alpha\}, S_i^T := \{j \mid i \in S_j\}$, where $N_i := \{j \mid j \neq i, \gamma_{ij} \neq 0\}$ is the set of *neighbors* of i.

After the unknowns of the level have been partitioned, the *interpolation* operator $I_{k+1}^k : \mathbf{R}^{n^{k+1}} \longrightarrow \mathbf{R}^{n^k}$ is defined by $v^k = I_{k+1}^k v^{k+1}$

$$v_{j(i)}^k \overset{\text{def}}{=} \begin{cases} v_i^{k+1} & \text{for } i \in C^k \\ -\sum_{l \in C^k} \gamma_{il}^k v_l^{k+1} / \gamma_{ii}^k & \text{for } i \in F^k \setminus F_d^k \\ 0 & \text{for } i \in F_d^k \end{cases} . \tag{6}$$

The coarse level system for level $k+1$ is now defined by the so-called *Galerkin* or *variational* conditions. The *restriction* operator is the transpose of the *interpolation* operator

$$I_k^{k+1} = (I_{k+1}^k)^T \tag{7}$$

and the reduced system matrix is

$$C^{k+1} = I_k^{k+1} C^k I_{k+1}^k . \tag{8}$$

Note that all coarse level matrices inherit the positive definiteness from C, provided all I_k^{k+1} have full rank.

The AMG algorithm can now be described as follows.

```
1. set $k = 0$
2. **Do** set $k = k + 1$; SETUP PHASE I and SETUP PHASE II; **until** $|\Omega^k| = 1$
3. **While** $\|b - Cx\| \geq \delta$
       MGSTEP(1)
```

MGSTEP(k):

```
1. **If** $k = K$ **then** solve (11)
2. **else** SMOOTH($x^k$)
       set $b^{k+1} = I_k^{k+1}(b^k - C^k x^k)$
       MGSTEP(k+1)
       set $x^k = x^k + I_{k+1}^k x^{k+1}$
       SMOOTH($x^k$)
3. **endif**
```

We now discuss the handling of constraints in the AMG-algorithm. Just like the system matrix, the constraints (2) must be transferred to the coarse levels. Equation (2) thus becomes a family of constraints

$$A^k x^k = d^k, \tag{9}$$

for $k = 1, 2, \ldots, K$ corresponding to the reduced systems (3), where

$$A^{k+1} = A^k I^k_{k+1}. \tag{10}$$

The original matrix coincides with the first and largest problem in the family, $A = A^1$, $d = d^1$.

The algorithm is modified such that (9) is satisfied on each level. On the coarsest level this is accomplished by solving the system with constraints directly using a Lagrange multiplier approach

$$\begin{bmatrix} C^K & (A^K)^T \\ A^K & 0 \end{bmatrix} \begin{bmatrix} x^K \\ \lambda \end{bmatrix} = \begin{bmatrix} b^K \\ d^K \end{bmatrix}. \tag{11}$$

The definition of the coarse level equations and constraints by a Galerkin condition has the effect that the finer level equations remain satisfied after a coarse grid correction, provided the coarse level constraints have been satisfied.

After each smoothing step, the constraints will be violated. This is compensated by an additional projection that enforces the constraints. Note that for general constraints the transfer can lead to coarse grid problems that are not well defined. This has been studied in detail in Bungartz [9]. Even if both A^k and I^k_{k+1} have full rank, $A^k I^k_{k+1}$ may not. In this case constraints have become linearly dependent and the subspace determined by $A^{k+1}x^k = d^{k+1}$ is either overdetermined or empty. In the case of overdetermined constraints, the number of constraints should be reduced. Numerically, however, detecting and treating this situation is difficult. Ideally, the matrix of constraints A^k should already be considered in the coarse level setup.

Here, we concentrate on the type of situation arising in the placement problem. With each constraint, a group of cells is assigned to a subdomain. Each cell is uniquely assigned to one such subdomain and the coefficients of the matrix A^k are determined by the relative surface area of the corresponding cells. Clearly, the rows of A^k are orthogonal. The coarse level constraints will remain consistent, if the interpolation I^k_{k+1} is constructed such that a coarse level variable only interpolates variables belonging to the same subdomain. Unfortunately, the constraints are still unknown in the (first) setup phase. In practice inconsistencies rarely arise, if we guarantee that the dimension of the coarsest level is larger than the number of constraints.

On the coarsest level the Lagrange multipliers λ must be calculated. This requires the solution of a full system of a dimension that is equal to the number of constraints. The number of constraints doubles with each partitioning step. Thus the coarsest permissible level may be quite large and expensive to solve exactly, making the algorithm unacceptable for large chips.

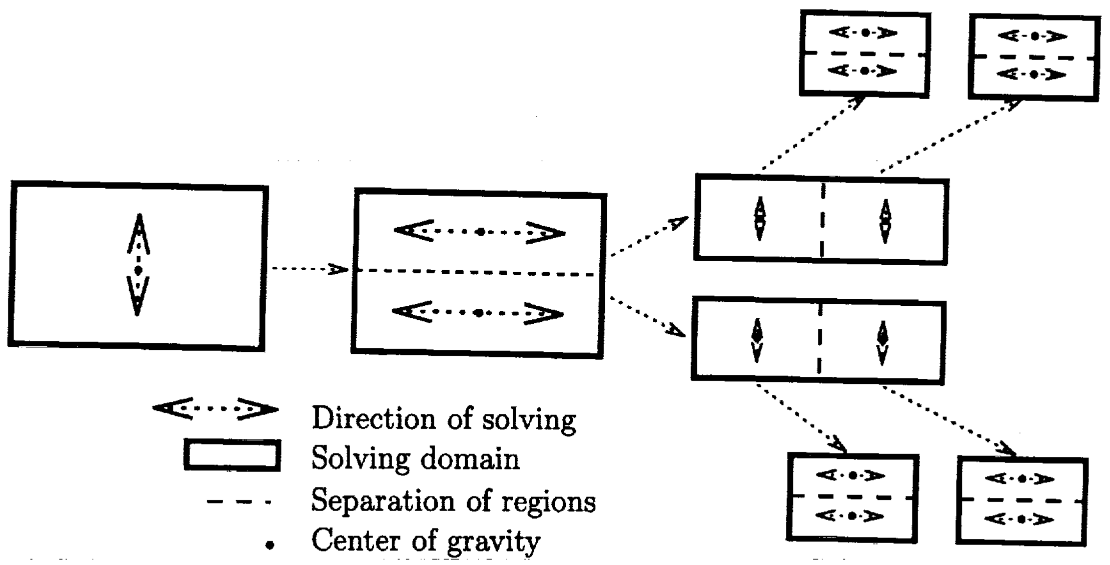

Figure 3: The double arrow shows in which direction the solution is calculated, here it is started in y direction. The box indicates the subdomains for which the computation is performed. The dashed line indicates a separation of the regions; cells cannot cross such a line during the overall placement calculation.

Experience shows that the influence of cells in different subdomains is rather small and may be neglected. Additionally, earlier experiments with GORDIAN have shown that the quadratic objective functional is only a crude approximation to the true one. It can be argued that the usual routing of connections in the final layout induces a measure of distances that is modeled better by an L_1-like norm and a linear objective functional (see Sigl [10]). This motivates an algorithm that recursively splits the problem into independent ones by partioning into subdomains. A *solution subdomain* is defined as two neighboring subdomains that have been obtained by partioning a single subdomain of the previous iterations. We can now simulate the effect of a linear objective functional by keeping the cells fixed in all subdomains except those in the current solution subdomain. This must be repeated for all solution subdomains. Thus, though the above simplification changes the mathematical model, the modified algorithm may help to produce better overall layouts. This is indicated by experimental results.

The algorithm is illustrated in Figure 3. The first two calculations are performed as before, without any change. After the second partitioning, the computation of the overall chip is split into an upper and a lower solution subdomain. When the new solution for the upper solution subdomain is computed, the cells of the lower solution subdomain are kept fixed. Simultaneously, the lower solution subdomain is computed with fixed upper domain cell positions. This is repeated recursively until the partitioning is completed. Clearly, this algorithm can be easily parallelized because each solution subdomain can be computed independently. Because no data exchange between the different solution subdomain is necessary, this is a plain *divide-and-conquer* algorithm inducing a natural parallelization. Note, that we have to solve systems with at most two simultaneous constraints. This leads to an algorithm, where it is sufficient to perform the setup once at the beginning of the computation. Before each optimization step the (at most two) constraints are tested for linear dependencies. In the case of inconsistent constraints the previous level is taken for the coarsest level.

The conventional setup of the coarse level matrices is *variational* in the sense that (7) and (8) are satisfied. Experience shows that the coarse level matrices tend to fill up rather quickly. On the other hand, the definition by equation (8) often leads to *small* matrix entries, so that one may have the idea to modify the coarser matrices by dropping small entries. More precisely, we may perturb each C^k to

$$\tilde{C}^k = C^k + B^k, \tag{12}$$

such that the matrix remains sparse. This will not only speed up each individual iteration, but also simplify coarser matrix setups. We suggest performing the perturbation such that the matrix remains symmetric and such that dropped values are added to the diagonal with the opposite sign. For an analysis of these perturbations see Muszynski, Rüde, and Zenger [11], Bungartz [9], and Chang and Wong [12].

Classical AMG is used with a single sweep of Gauss-Seidel smoothing on each level. Alternatively, we may use Jacobi-type smoothers. As usual, the Jacobi method must be damped to obtain good smoothing. Though the Jacobi method is usually a less efficient smoother than Gauss-Seidel (even with optimal damping), it may be an interesting alternative, because it has a symmetric error propagation matrix without performing sweeps in reverse order. Jacobi-AMG may thus be used directly as a preconditioner for the conjugate gradient method. Another advantage of Jacobi is parallelization. To parallelize Gauss-Seidel we would have to find a coloring scheme for a general unstructured matrix that permits the parallel execution of relaxation steps. Our experimental results (see Figure 5) indicate that two optimally damped Jacobi iterations are about as good a smoother as a single sweep of Gauss-Seidel. This is in agreement with experience for the solution of partial differential equations. In future work we intend to experiment with other smoothers, like conjugate residuals or incomplete LU decomposition; see e.g. Bank and Douglas [13].

We denote the diagonal part of C^k by D^k and can thus write a damped Jacobi iteration for level k as

$$x^k \leftarrow x^k + \omega(D^k)^{-1}(b^k - C^k x^k), \tag{13}$$

where ω is the relaxation parameter. For the error $e = x - C^{-1}b$ in the original system, a relaxation on level k has an effect that can be described by

$$e \leftarrow (I - I_k^1 (D^k)^{-1} I_1^k C)e, \tag{14}$$

where

$$I_1^k = \prod_{j=1}^{k-1} I_j^{j+1}. \tag{15}$$

The AMG algorithm in its simplest form (with a single sweep of Jacobi on each level) has an error propagation

$$e \leftarrow \prod_{k=1}^{K} (I - I_k^1 (D^k)^{-1} I_1^k C)e. \tag{16}$$

This is a typical *multiplicative* method.

All conventional multigrid methods, including AMG, are *multiplicative* algorithms in the sense that the levels are visited sequentially in a predetermined order. The recent development of multilevel methods has led to the formulation of a class of *additive* multilevel methods. These include the AFAC type algorithms (see McCormick [14]), the BPX method (see Bramble, Pasciak and Xu [15]), and the multilevel additive Schwarz methods (see Dryja and Widlund [16]). Formally, these methods do not form a product of operators as in (16), but a sum, whose terms can — in principle — be computed simultaneously.

With some exceptions (like the AFAC method), additive methods provide only preconditioners that are divergent when used as iterations by themselves. However, they define operators with improved condition numbers, and so they will lead to fast convergence when suitably damped or when they are used in combination with self-scaling iterative methods, most notably the conjugate gradient algorithm. Recent results have shown that these methods can have typical multigrid efficiency with convergence rates independent of the problem size.

We will show that for our problems

$$P^k \overset{\text{def}}{=} \sum_{j=1}^{k} I_j^1 (D^j)^{-1} I_1^j C \tag{17}$$

also has a better condition number than the original matrix C. Note that an application of P^k does not require the explicit construction of the corresponding matrix, but only the restriction of the residuals to all levels, just like in conventional AMG. An iteration based on P^k, like

$$x = x + \omega \sum I_k^1 (D^k)^{-1} I_1^k (b - Cx) \tag{18}$$

will only converge, when suitably damped with $\omega < 1$. Preferably (18) is used as a preconditioner for a conjugate gradient iteration.

NUMERICAL EXPERIMENTS

Our first example is a typical benchmark chip called *Primary I* with 752 cells, 81 fixed cells, and 902 signals. Figure 4 shows the corresponding matrix structure, and Figure 5 displays the convergence history of (multiplicative) AMG using different smoothers for the solution of (1). Clearly two sweeps of damped Jacobi are almost as good a smoother as Gauss-Seidel (GS). In Table 1 the minimal and the maximal eigenvalue $(\lambda_{min}, \lambda_{max})$ plus the condition number $\kappa = \lambda_{max}/\lambda_{min}$ of P^k are shown. On the coarsest level ($k = 6$) D^k is replaced by C^k. This means that the coarsest level equations are solved exactly. In Figure 6 we present the corresponding spectrum of the eigenvalues for $k = 1, 3, 6$. In each case, the first few eigenvalues are marked by asteriks(*) and diamonds($\diamond$), respectively. In column 5 and 6 of Table 1, the density and dimension of the coarse level system C^k are displayed additionally.

In Figure 7 we show the convergence history for preconditioned CG in analogy to Figure 5 in comparison to the AMG-solver with Gauss-Seidel smoothing. Conventional AMG is superior to AMG-preconditioned CG, partly because Gauss-Seidel is a better smoother than Jacobi. However,

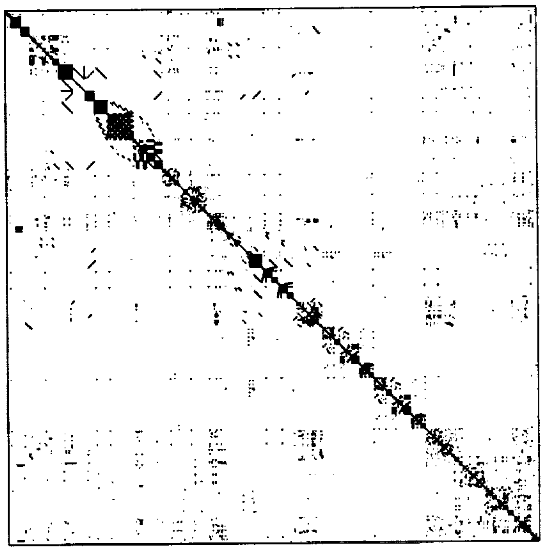

Figure 4: Sparsity pattern of system matrix of Primary I

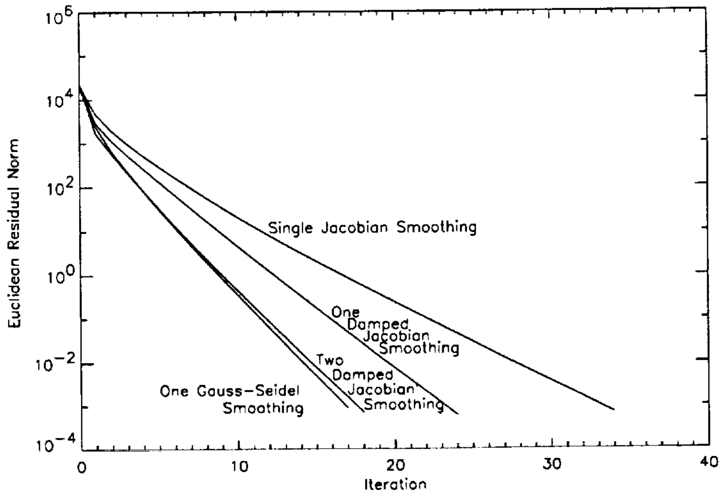

Figure 5: Convergence history for AMG with different smoothers

k	λ_{min}	λ_{max}	κ	density	dim
1	0.0085	1.8301	215.3	0.02	752
2	0.0302	3.4020	133.4	0.09	343
3	0.0598	4.2936	71.8	0.33	147
4	0.1117	4.9196	44.0	0.72	59
5	0.2500	5.9495	23.8	0.95	26
6	0.4060	6.1167	15.1	1.00	10

Table 1: Eigenvalues and characteristics of Primary I

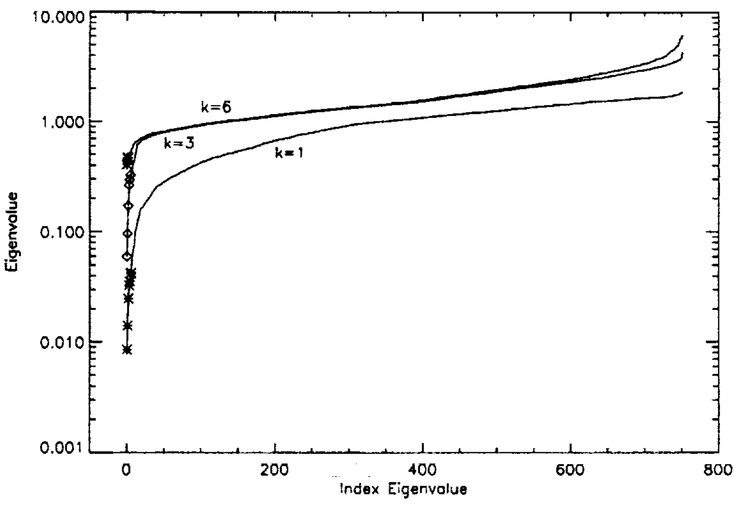

Figure 6: Eigenvalues of P^k for Primary I matrix

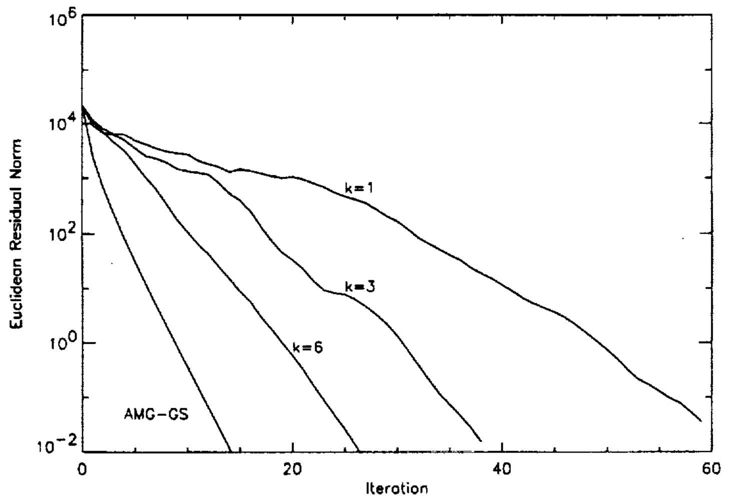

Figure 7: Convergence history of preconditioned CG for Primary I

AMG-preconditioned CG is an interesting alternative when we consider its potential for parallelization.

In further tests we have applied the AMG algorithm to a problem of similar size arising from the discretization of a partial differential equation and have found that the behavior is surprisingly similar.

Finally, we present results for a real-life chip with 25178 cells. The original preconditioned CG solver (CG) in GORDIAN is replaced by the AMG routines combined with the *divide and conquer* strategy. In Table 2 we compare the CPU times for CG and AMG for the optimization after each partitioning step. The first AMG step includes the setup time, which is 9 times as expensive as the iteration itself, but still faster than CG. AMG outperforms conventional CG for almost all subproblems, except the very last six partitions. In the overall time AMG is still clearly superior to CG.

Partition	CG	AMG	sol	par
0	126.3	43.7	4.2	
1	104.5	5.4	4.9	5.4
2	91.1	33.8	8.3	15.2
3	85.1	29.7	8.2	6.9
4	79.2	26.7	7.8	3.4
5	76.5	45.0	25.8	19.7
6	61.7	35.4	17.8	9.6
7	58.1	24.2	9.0	0.6
8	58.2	50.4	34.3	9.4
9	115.0	47.9	30.3	3.0
10	91.5	38.2	15.2	0.9
11	72.9	114.4	33.4	0.5
12	30.4	137.0	37.8	0.4
13	15.7	41.7	13.2	0.1
14	12.7	19.4	11.0	0.0
15	6.2	11.2	9.5	0.0
16		9.0	8.7	0.0
total	1084.9	713.2	279.4	–

Table 2: avq : time [s] spent per partition

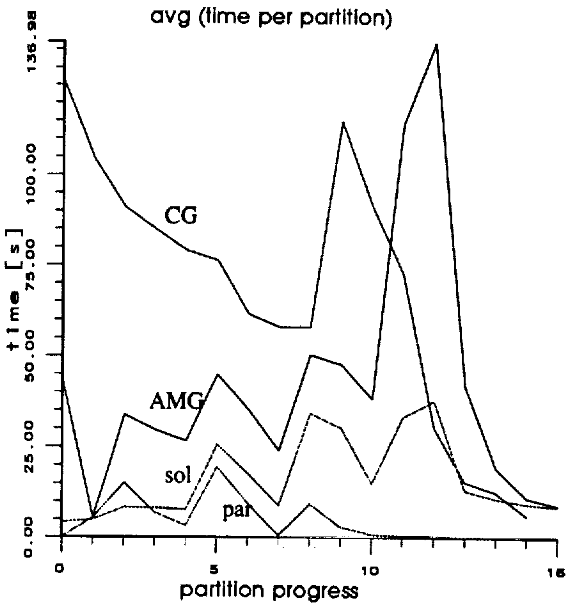

Figure 8: avq 1: time [s] spent per partition

Following the divide and conquer strategy, we transform the system into separately solvable subproblems after the second partition. This requires a transformation of the data that is not yet optimally implemented. The column labeled "sol" therefore shows the time for the AMG solution process without the overhead for this data transformation. The overhead for the transformation increases with the number of partitions, adding to the cost of the AMG method.

However, each subdomain can be computed in parallel. To illustrate the potential for parallelization, the "par" column shows the maximal time needed for computation of a subdomain, thus simulating the effect of an optimal parallelization. The example chip for this calculation is a *standard cell chip.* This type of chip has a fixed number of rows of cells. Thus subdomains with height below a certain minimum are not permitted. To avoid this, GORDIAN computes the partition for both directions until the maximal number of rows is reached. Here, this applies to partition 9,10,11 during conventional CG; for the AMG method this happens during partition 11,12,13. As the partition progresses, the original AMG setup may not be suitable any more and must be repeated for the subdomains that cause trouble. In our example this has been the case in partitions 5,8, and 9. For further discussion see Regler [3].

CONCLUDING REMARKS

We have discussed the application of algebraic multigrid methods and have proposed several variants and extensions of the classical AMG method of Ruge and Stuben, including constrained optimization and a new additive algorithm. We have shown that the AMG method is a highly competitive alternative for the layout optimization of real life chips.

Acknowledgements: We wish to thank H. Bungartz, K. Doll, F. M. Johannes, G. Sigl, and C. Zenger for many helpful discussions.

REFERENCES

[1] J. Kleinhans, G. Sigl, F. Johannes, and K. Antreich. Gordian: VLSI placement by quadratic programming and slicing optimization. *IEEE Trans. Computer-Aided Design*, CAD-10:356–365, March 1991.

[2] G. Sigl. Plazierung der Zellen bei der Layoutsynthese mittels Partitionierung und quadratischer Optimierung. Dissertation, Lehrstuhl für Rechnergestütztes Entwerfen, Technische Universität München, 1992.

[3] H. Regler. Algebraic multilevel methods in chip design. In *Proceedings of the GAMM-Seminar on Multigrid Methods, Sept. 21 – 25, 1992 in Gosen, Germany*, Berlin, 1993. Institut für Angewandte Analysis und Stochastik. Report 5, ISSN 0942–9077.

[4] J. Ruge and K. Stüben. Efficient solution of finite difference and finite element equations by algebraic multigrid (AMG). *Arbeitspapiere der GMD*, 89, 1984.

[5] A. Brandt. Algebraic multigrid theory: The symmetric case. In S. McCormick and U. Trottenberg, editors, *Preliminary Proceedings of the International Multigrid Conference, Copper Mountain, Colorado, April 6-8, 1983*, 1983.

[6] A. Brandt, S. McCormick, and J. Ruge. Algebraic multigrid (AMG) for automatic algorithm design and problem solution. Report, . Comp. Studies, Colorado State University, Ft. Collins, 1982.

[7] A. Brandt, S. McCormick, and J. Ruge. Algebraic multigrid (AMG) for automatic multigrid solution with applications to geodetic computations. In Evans, editor, *Sparsity and its Applications*. Cambridge University Press, 1984.

[8] J. Ruge and K. Stüben. Algebraic multigrid (AMG). Arbeitspapiere der GMD, Gesellschaft für Mathematik und Datenverarbeitung, 1986.

[9] H. Bungartz. Beschränkte Optimierung mit algebraischen Mehrgittermethoden. Diplomarbeit, Institut für Informatik, Technische Universität München, 1988.

[10] G. Sigl, K. Doll, and F. Johannes. Analytical placement: A linear or a quadratic objective function? In *ACM/IEEE Proceedings 28th Design Automation Conference*, 1991.

[11] P. Muszynski, U. Rüde, and C. Zenger. Application of algebraic multigrid (AMG) to constrained quadratic optimization. Bericht I-8801, Institut für Informatik, TU München, January 1988.

[12] Qianshun Chang and Yau Shu Wong. Recent developments in algebraic multigrid methods. In T. Manteuffel and S. McCormick, editors, *Preliminary proceedings of the 2nd Copper Mountain Conference on Iterative Methods, Copper Mountain, April 9-14, 1992*. University of Colorado at Denver, 1992.

[13] R. Bank and C. Douglas. Sharp estimates for multigrid rates of convergence with general smoothing and acceleration. *SIAM J. Numer. Anal.*, 22:617–633, 1985.

[14] S.F. McCormick. *Multilevel Adaptive Methods for Partial Differential Equations*, volume 6 of *Frontiers in Applied Mathematics*. SIAM, Philadelphia, 1989.

[15] J. Bramble, J. Pasciak, and J. Xu. Parallel multilevel preconditioners. *Math. Comp.*, 31:333–390, 1990.

[16] M. Dryja and O. Widlund. Multilevel additive methods for elliptic finite element problems. In W. Hackbusch, editor, *Parallel Algorithms for Partial Differential Equations, Proceedings of the Sixth GAMM-Seminar, Kiel, January 19-21, 1990*, Braunschweig, 1991. Vieweg-Verlag.