An Algebraic Multigrid Solver for Analytical Placement with Layout Based Clustering

Hongyu Chen[1], Chung-Kuan Cheng[1], Nan-Chi Chou[2], Andrew B. Kahng[1],
John F. MacDonald[3], Peter Suaris[4], Bo Yao[1], Zhengyong Zhu[1]

[1] University of California, San Diego, CSE Dept., La Jolla, CA 92093-0114
[2] Mentor Graphics Corporation, 1001 Ridder Park Drive, San Jose, CA 95131
[3] Mentor Graphics Corporation, 11232 El Camino Real, Suite 200, San Diego, CA 92130
[4]Mentor Graphics Corporation, 8005 SW Boeckman Road, Wilsonville, OR, 97070

ABSTRACT
An efficient matrix solver is critical to the analytical placement. As the size of the matrix becomes huge, the multilevel methods turn out to be more efficient and more scalable. Algebraic Multigrid (AMG) is a multilevel technique to speedup the iterative matrix solver [10]. We apply the algebraic multigrid method to solve the linear equations that arise from the analytical placement. A layout based clustering scheme is put forward to generate coarsening levels for the multigrid method. The experimental results show that the algebraic multigrid solver is promising for analytical placement.

Categories and Subject Descriptors
B.7.2 [**Design Aids**]: Layout.

General Terms
Algorithms, Performance, Experimentation.

Keywords
Analytical placement, Algebraic multigrid method, Layout based clustering

1. INTRODUCTION
The analytical placement formulates the placement problem into some mathematical programs, and solves a set of simultaneous equations to provide the placement solution. Quadratic placement [2,3,4] and force-directed placement [1] are examples of the analytical placement. Lots of computations are involved in solving the linear equations. An efficient matrix solver, therefore, is crucial to such a placement system.

As the number of components in a placement system keeps growing, the size of the matrix is also getting larger. There could be millions of unknowns in the equations in the near future. To solve such a huge linear system, the single-level method, like the Successive Over Relaxation method (SOR), has reached its limit. The multilevel methods are more efficient and scalable.

[4] and [6] are among the first to introduce multilevel method into placement problems. [4] solves the placement for the clustered circuit first, and uses it as an initial guess of the fine level placement. [6] incorporates the V-cycle optimizing scheme: optimization on a hierarchy of coarsening graphs. It uses non-linear programming at the coarsest level, and adopts heuristics for mapping solutions between adjacent levels.

In circuit/hypergraph partitioning, multilevel partitioners usually get very good results [5,7]. For solving large partial differential equations (PDEs), multigrid technique can work hundreds times faster than the one-level method [9]. The multigrid method is also introduced to power mesh analysis [11] recently to speed up the calculation.

The idea behind multigrid is to decompose the error of a solution into the low frequency part and the high frequency part. The one-level iterative method, like Gauss-Seidel iterations, can effectively eliminate the high frequency errors on each level. The low frequency errors are left for the coarser levels. A solution mapping from the fine level to the coarse level translates the low frequency error at the fine level into the high frequency error at the coarse level, in which it can be efficiently eliminated.

The traditional multigrid methods assume that there is a uniform geometric grid structure for the problem, so the interpolation and restriction can be defined geometrically [9]. A new branch called algebraic multigrid (AMG) was developed to handle the case where no geometric grid structure is available. The AMG method defines a hierarchy of coarsening levels solely from the matrix itself. Thus it can be applied to a broader range of problems. Moreover, the convergence of the algebraic multigrid method is guaranteed when the smoothing iteration on each level converges.

The matrix arising from the analytical placement is symmetric positive-definite. This nice property makes the AMG method a good candidate. But the irregularity of the matrix makes it difficult to find a good hierarchy for the AMG method.

We focus on the solver for the linear systems arising from analytical placement. We incorporate the complete algebraic multigrid scheme in our solver, including the V-cycle iteration, the smoothing iterations on each level, and the restrictions and interpolation operations based on the clustering hierarchy. Moreover, We propose a new layout based clustering method to generate a hierarchy that is necessary for the AMG method.

Our contributions include the following:

1. Introduce for the fist time the AMG method to the analytical placement solver. Our implementation of the AMG solver runs 5 times faster than the SOR solver, and 2 times faster than the PCG solver.
2. Give a circuit theory interpretation of how the algebraic multigrid solver works.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
DAC 2003, June 2-6, 2003, Anaheim, California, USA.
Copyright 2003 ACM 1-58113-688-9/03/0006…$5.00.

3. Propose a new cell clustering method based on the layout.
4. Demonstrate that the proposed clustering approach is superior to other clustering approach in terms of the performance and the ability to work with arbitrary circuit structures.

The rest of the paper is organized as follows: Section 2 gives the problem formulation of the analytical placement. Section 3 introduces our algebraic multigrid solver. Section 4 reveals our layout based clustering method. Section 5 gives the experimental results. The conclusion is drawn in section 6.

2. ANALYTICAL PLACEMENT

In this section we provide the formulation of the analytical placement and outline the approach to find the solution.

2.1 Problem Formulation

We formulate the analytical placement the same way as in [1]. A circuit is represented by a weighted graph, $G=\{V, E\}$. Each node $v_i \in V$ corresponds to a cell in the circuit. Each edge e_i corresponds to a two-pin signal net. A star net model [8] is used to convert the multi-pin net to a set of two pin nets if necessary.

Let n be the number of movable cells in the circuit, (x_i, y_i) be the coordinate of each node v_i, X and Y be the vectors $(x_1, x_2, ..., x_n)$ and $(y_1, y_2, ..., y_n)$, respectively. The cost of each edge is the squared Euclidean distance between the two cells connected by the edge. The object function is to minimize the sum of the edge costs. The analytical placement problem can be written in the matrix form as the following mathematical program *QP*.

$$QP: \min\{\Phi(X,Y) = \tfrac{1}{2}X^TCX + d_x^TX + \tfrac{1}{2}Y^TCY + d_y^TY\} \quad (1)$$

Where C is an $n \times n$ symmetric matrix. Matrix C is sparse and positive definite as long as graph G is connected and at lease one node linked to some fixed point [1]. d_x and d_y are the n-dimensional vectors.

2.2 Solving Analytical Placement

The object function contains two parts: $\Phi(X) = \frac{1}{2}X^TCX + d_x^T$ and $\Phi(Y) = \frac{1}{2}Y^TCY + d_y^T$, which can be minimized independently. In the following we focus on the $\Phi(X)$ part only.

$\Phi(X)$ can be minimized by solving the linear equation system

$$CX + d_x = 0 \quad (2)$$

To place the cells evenly, [1] introduces additional forces on each cell, which are derived from cell density. Equation (2) is extended to incorporate the additional force:

$$CX + d_x + e_x = 0 \quad (3)$$

where e_x is the additional force vector on cells in the x direction.

Several iterations are needed to solve the analytical placement. In each iteration, the additional force vector e_x is updated according to the current placement. Equation (3) is then solved to get the new placement.

One important property of the placement process is that the equations in the iteration share the same left hand matrix C. We take advantage of this property in the proposed algebraic multigrid solver: We derive the coarsening levels from the matrix C only once, and reuse them in the AMG solver for all the equations (3).

Conjugate gradient methods are used in [1,2,4] to solve the linear equations. SOR solver is also widely used because of its simplicity and efficiency for small circuits [2]. We propose a new multigrid-based method that is more efficient on large cases.

3. ALGEBRAIC MULTIGRID SOLVER

This section presents the AMG solver to solve equation (3). A circuit theory based interpretation is also described.

3.1 Elements of the multigrid method

The algebraic multigrid method works on a hierarchy of coarsening levels. It maps the equations and solutions between the levels. It iterates on all levels to reduce the error of the solution efficiently. The elements of the AMG solver are as follows:

3.1.1 Coarsening levels and the clustering

A hierarchy of coarsening levels, Ω^0 to Ω^K, is necessary for the AMG method. Following [9], we assume that Ω^0 is the finest level and Ω^K is the coarsest level. To each level Ω^k, we associate with it a linear equation $A^kX^k = b^k$, and a graph $G^k = \{V^k, E^k\}$. The finest level equation is the original equation we want to solve. For example, if we want to solve equation (3), we have $A^0 \equiv C$, $X^0 \equiv X$ and $b^0 \equiv -(d_x + e_x)$.

The levels are generated by a hierarchical clustering process. A clustering from level Ω^k to Ω^{k+1} is defined as a mapping of the nodes in V^k to the nodes in V^{k+1}: Each node in level k is mapped to one and only one node in level $k+1$. We may also define the clustering relationship by a *restriction matrix* I_k^{k+1}. Let $|V^k| = n^k$.

The n^k by n^{k+1} matrix $I_k^{k+1} = (i_{ij})$ is defined by

$$i_{ij} = \begin{cases} 1 & \text{if node } v_j^k \text{ is mapped to node } v_i^{k+1} \\ 0 & \text{Otherwise} \end{cases}$$

In each column of I_k^{k+1} there is one and only one "1". Note that $n^{k+1} \le n^k$, so the higher level has less number of nodes.

If a cluster node v_i in level k is mapped to a cluster node v_j in level $k+1$, the node v_j is called the *parent* of node v_i, and the node v_i is called the *child* of the node v_j.

Given the clustering scheme, we get the linear equations on each level by the Galerkin operation [10]:

$$A^{k+1} = I_k^{k+1} A^k I_{k+1}^k \quad (4)$$

$$b^{k+1} = I_k^{k+1} b^k \quad (5)$$

Where $I_{k+1}^k = (I_k^{k+1})^T$ is the *interpolation matrix*.

Property of A^k: The matrix A^k is symmetric positive-definite (s.p.d.).

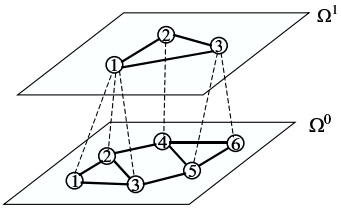
The matrix $A^0 \equiv C$ is s.p.d. by definition [1]. The transformation from A^k to A^{k+1} (equation (4)) guarantees A^{k+1} to be s.p.d. if A^k is s.p.d., because $I_{k+1}^k = (I_k^{k+1})^T$, and I_{k+1}^k is of full rank.

Figure 1 gives an example of a two level clustering. In Figure 1(a), the original circuit is drawn on the bottom level Ω^0. The clustered circuit is drawn on the top level Ω^1. The clustering relationships are represented by the dashed lines: The nodes 1, 2 and 3 in level Ω^0 are clustered to node 1 in level Ω^1. The node 4 in level Ω^0 is clustered into node 2 in level Ω^1. The nodes 5 and 6 in level Ω^0 are clustered into node 3 in level Ω^1. The interpolation matrix I_0^1 is shown in Figure 1(b).

We will introduce how to get the K-level clustering in section 4. Here we assume that the clustering relations are given, i.e., all the matrices I_k^{k+1} are known for $k = 0, 1, \ldots, K-1$.

3.1.2 Restriction and interpolation

Restrictions and interpolations are defined to map the solutions between two adjacent levels, based on the clustering structure. We



(a)

$$I_0^1 = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

(b)

Fig. 1 A two level clustering example.

adopted similar interpolation and restriction approaches as in the aggregate-type AMG [14]. *Interpolation* is a mapping of a level *k+1* solution X^{k+1} on to a level *k* solution X^k as follows:

$$X^k = I_{k+1}^k X^{k+1} + R^k \quad (6)$$

Where I_{k+1}^k is the restriction matrix. R^k is the offset position vector of level *k* nodes to their parent nodes in level *k+1*.

Restriction is a mapping of a level *k* solution to a level *k+1* solution. We adopt the following correction process.

$$X^{k+1} = I_k^{k+1} X^k N^{k+1} \quad (7)$$

Where $N^{k+1} = (n_{ii})$ is the diagonal matrix of $n_{ii} = 1/m_{ii}$, and m_{ii} is the number of children of a cluster c_i in level *k+1*.This restriction operation puts the level *k+1* nodes at the average position of their children in the level *k*.

In restriction operation, we also update the vector R^k and the coarse level right hand side b^{k+1} as follows:

$$R^k = X^k - I_{k+1}^k X^{k+1} \quad (8)$$

$$b^{k+1} = I_k^{k+1} (b^k - A^k R^k) \quad (9)$$

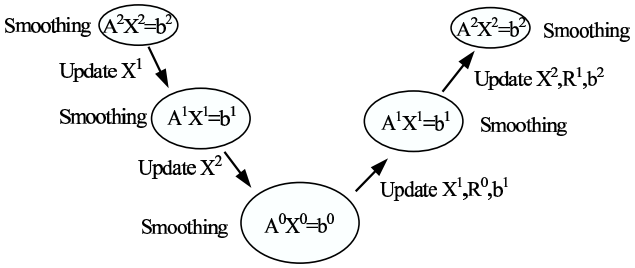


Fig. 2. A V-cycle iteration on 3 levels.

Algorithm V-cycle iteration
Input: $A^0, b^0, \ldots, A^k, b^k$
Output X^0
Begin
  Initialize $X^0, \ldots, X^k$ to zero vectors;
  Initialize $R^0, \ldots, R^{(k-1)}$ to zero vectors;
  For i = k down to 1 do
    Iterate on $A^iX^i=b^i$
    $X^{(i-1)} = I_i^{(i-1)}X^i + R^i$
  Endfor
  Iterate on $A^0x^0 = b^0$;
  Update $R^0 = X^0 - I_1^0X^1$;
  Update $b^1 = I_0^1(b^0 - A^0R^0)$
  For i = 1 to k-1 do
    Iterate on $A^iX^i=b^i$;
    Update $R^i = X^i - I_{i+1}^i X^{i+1}$;
    Update $b^{i+1} = I_i^{i+1}(b^i - A^iR^i)$
  endfor
End

Fig. 3. The pseudo code for a V-cycle iteration

3.1.3 Smoothing

The iteration on each level to refine the solutions is called smoothing. Gauss-Seidel iterations are usually used for smoothing. We use SOR iterations to speed up the convergence [14]. The experiments show that the AMG method convergence faster with SOR iteration as the smoothing operation.

3.1.4 The V-cycle iteration

The V-cycle iteration begins at the coarsest level, Ω^K. We got the estimation of X^K after some smooth iterations. We then transform the solution X^K to X^{K-1} by equation (6). The smooth iteration is carried out at level K-1 and another transformation by equation (6) is done to get the solution X^{K-2}. We keep on going downwards until we reach the finest level. After we iterate on level 0, we call a correction procedure (equation (7)) to update X^1 from X^0. We also update vectors b^1 and R^0 by equation (8) and (9). The smoothing iteration is performed at level 1 before we move to level 2. We keep on going upwards until the top level is reached. Figure 2 illustrates the process of a V-cycle with 3 levels. The pseudo code for a V-cycle iteration is shown in Fig. 3.

3.1.5 Overall solving procedure

The overall solving procedure begins with the clustering as a preprocess, and constructs a hierarchy first. Then for each equation (3) to be solved, several V-cycle iterations are carried out one after another until the solution converges.

3.1.6 Convergence issues

The solution error after *k* V-cycle iterations can be defined as

$$e_{(k)} = \| X^0_{(k)} - X^0_{(k-1)} \|_\infty$$

Where $X^0_{(k)}$ is the solution at the finest level. We define the convergence rate at iteration *k* to be $\rho_{(k)} = e_{(k)} / e_{(k-1)}$. We say that the V-cycle iteration *converges* if $\rho_{(k)} < 1$.

The rigorous convergence analysis for the AMG method is still an open problem. Our experiments show that the AMG method is globally convergent, i.e., it always gets closer to the exact solution after each V-cycle. We have the following theorem on the convergence rate of the two-level AMG method [10]:

Theorem 1: Let $A=(a_{ij})$ be a symmetric, positive-definite matrix.

Let R_i be the set of other nodes that are clustered with node *i*, and N_i be the set of neighbor nodes of *i*. If there is a fixed $0<\tau\leq 1$ that for each node *i* who belongs to a cluster with more than one node, the following relation holds: $\max_{k\in N_i \cap R_i} |a_{ik}| \geq \tau \sum_{j\in N_i} |a_{ij}|$, then the convergence rate of the two-level AMG method can be bounded by $\rho \leq \sqrt{1-\tau/4}$.

Theorem 1 tells us that the convergence of the AMG method is independent of the problem size.

3.2 A Circuit Theory Interpretation

We explain how the multigrid solver works by an analogy to the circuit analysis. An example with two level clustering is used.

The level 0 placement system is illustrated in Fig. 4(a). It contains 5 movable nodes, which are represented by dots, and 2 fixed pad cells, which are represented by the squares. Fig. 4(b) depicts the resistive network circuit analogs to the placement system in Fig. 4(a). Each node in Fig. 4(b) corresponds to a cell in Fig. 4(a); each branch corresponds to an edge. The resistance on each branch equals to the weight of the corresponding edge in Fig. 4(a). The voltage on each node corresponds to the position of each cell.

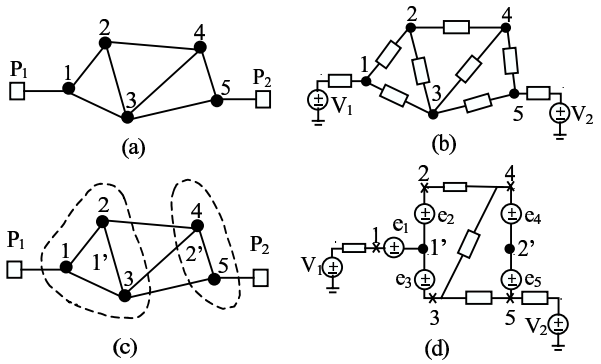


Fig. 4. A placement system and the corresponding circuit.

The cluster is defined by merging nodes 1, 2 and 3 on level 0 into node 1' on level 1, and merging cell 4 and 5 into node 2'. The clustering is illustrated in Fig. 4(c) with the dashed line circle. Fig. 4(d) shows the corresponding circuit for the clustered placement system. Two new nodes, 1' and 2', are introduced. The old nodes are also marked with crosses. The voltages on the two new nodes are the unknowns for level 1. Five voltage sources, e_1 to e_5, are introduced to represent the voltage difference between the level 0 nodes and the level 1 nodes.

We assume that all the resistance in Figure 4(b) equals to 1. The KCL equations on level 0 is $\boldsymbol{A}^0\boldsymbol{X}^0 = \boldsymbol{b}^0$, where

$$A^0 = \begin{bmatrix} 3 & -1 & -1 & 0 & 0 \\ -1 & 3 & -1 & -1 & 0 \\ -1 & -1 & 4 & -1 & -1 \\ 0 & -1 & -1 & 3 & -1 \\ 0 & 0 & -1 & -1 & 3 \end{bmatrix}, \quad X^0 = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{bmatrix} \text{ and } b^0 = \begin{bmatrix} V_1 \\ 0 \\ 0 \\ 0 \\ V_2 \end{bmatrix}.$$

The interpolation matrix is $I_0^1 = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$. According to our AMG method, the level 1 equation is $\boldsymbol{A}^1\boldsymbol{X}^1 = \boldsymbol{b}^1$, where

$$A^1 = I_0^1 A^0 (I_0^1)^T = \begin{bmatrix} 4 & -3 \\ -3 & 4 \end{bmatrix}, \quad X^1 = \begin{bmatrix} v_{1'} \\ v_{2'} \end{bmatrix} \text{ and}$$

$$b^1 = I_0^1 (b^0 - Q^0 R^0) = \begin{bmatrix} V_1 - e_1 - e_2 - e_3 + 2e_4 + e_5 \\ V_2 + e_2 + 2e_3 - 2e_4 - 2e_5 \end{bmatrix}.$$

Please note that this equation $\boldsymbol{A}^1\boldsymbol{X}^1 = \boldsymbol{b}^1$ is also the level 1 KCL equation for the two nodes 1' and 2'.

For the V-cycle iteration, we first iterate on the level 1 circuit to solve the two voltages V_1' and V_2'. At level 1, we assume that the voltage sources e_1 to e_5 are constants. We then interpolate by equation (5) to get the guess of the level 0 voltage on each node. The smoothing iterations are carried out on level 0 after that to refine the solution vector X^0. We then restrict the level 1 voltages, V_1' and V_2', to be the average voltage of each cluster (equation (6)). The voltage sources e_1 to e_5 and the right hand side vector b^1 are also updated according to the current values of X^0 and X^1. This brings us to the starting point of the next V-cycle iteration.

The iteration on level 1 will smooth the low frequency error, or the error of the average voltage of each cluster, while the iteration on level 0 will smooth the high frequency errors, or the errors of the voltages at each node.

4. CLUSTERING METHODS

The clustering methods provide a hierarchy of levels, which is necessary for the AMG solver. We propose a new layout based clustering method. We also tried the hMETIS clustering [5] and a random clustering. The best clustering scheme for the AMG solver, however, remains open.

4.1 Layout Based Clustering

The layout based clustering is inspired by the idea that the placement without any additional force tells us the coupling between the cells: The cells are close to each other in the layout and also logically connected should be strongly coupled. We try to cluster the strongly coupled cells together. On the other hand, we need a fast clustering process, so a greedy approach is chosen.

The distance information between the cells is generated by some tentative placement: We start from placing all the cells at the same place. A random initial placement is not adopted because the initial distance between the cells introduces some bias to the measurement. The SOR iteration is then used to drive the analytical placement. We take a snapshot of the placement when the standard deviation of the solution reaches its maximum. We call the placement at that time a blown up placement.

The distance d_{ij} for each edge e_{ij} connecting cell c_i and c_j is calculated as follows:

$$d_{ij} = d_{ij}^{(ll)} + d_{ij}^{(lr)} + d_{ij}^{(ul)} + d_{ij}^{(ur)} \tag{10}$$

where $d_{ij}^{(ll)}$, $d_{ij}^{(lr)}$, $d_{ij}^{(ul)}$ and $d_{ij}^{(ur)}$ are the geometric distances between cell c_i and c_j in the four blown up placements derived from initial settings of placing all the cells at the lower left, lower right, upper left and upper right corners of the chip, respectively.

The cost of each edge e_{ij} is defined as follows:

$$c_{ij} = \alpha \cdot d_{ij} + (1-\alpha) \cdot (s_i + s_j) \tag{11}$$

Where s_i is the size of the cluster c_i; α is a parameter for balancing the cost of the distance and the cluster size.

We follow the edge-coarsening scheme as described in [5]. At each level, we sort the edges by their cost c_{ij}. The edge with the smallest cost is collapsed first. The two clusters connected by that edge merge into a single cluster. The edge collapsing is repeated until the number of clusters is reduced by a preset factor. Figure 5 gives the pseudo code for the layout based clustering.

4.2 hMETIS Clustering

We also implemented the clustering method in the hMETIS

```
Algorithm Layout Based Clustering
Input systems A^0, b^0, G^0
Output : A hierarchy of coarsening levels
Begin
    Generate blown up placements by SOR iterations;
    Calculate the d_ij for each edge e_ij ;
    Calculate the cost c_ij of each edge e_ij;
    Num_clusters = num_of_cells;
    NumThreshold = num_clusters / RATIO; // RATIO = 4
    k = 0;
    While ( num_clusters > 100 )
        While (num_clusters > NumThreshold )
            Choose an edge e_ij^k with the smallest cost;
            Merge the two clusters c_i and c_j;
            Num_cluster --;
        Endwhile
        NumThreshold /= RATIO;
        k = k +1;
        Construct level k;
        Update cost of each edge in level k;
    Endwhile
End
```

Figure 5. Pseudo code for the layout based clustering

Table 1 Comparison of CPU time for SOR, PCG and AMG-L solvers to reduce the relative error to 10^{-2}

Circuit Name	# Cells	# Nets	# Non-zeros	SOR (ω= 1.95)		PCG		AMG-L		AMG-L Speedup	
				# iter	Time (s)	# iter	Solving Time (s)	# V-cycle	Solving Time (s)	Over SOR	Over PCG
IBM01	12752	14111	73126	70	2.75	40	2.47	7	1.76	1.56	1.40
IBM02	19601	19584	127684	90	6.73	25	3.09	4	1.70	3.96	1.82
IBM03	23136	27401	142327	80	8.14	25	3.41	4	2.12	3.84	1.61
IBM04	27507	31970	159248	90	10.18	30	4.56	4	2.73	3.73	1.67
IBM05	29347	28446	185286	90	12.76	20	2.91	4	3.75	3.40	0.78
IBM06	32498	34826	206190	80	13.66	25	5.12	3	2.24	6.10	2.28
IBM07	45926	48117	270759	90	21.33	45	12.10	3	4.27	4.99	2.83
IBM08	51309	50513	315927	90	24.13	30	9.86	3	4.66	5.17	2.11
IBM09	53395	60902	334798	150	47.34	50	16.95	4	6.73	7.04	2.52
IBM10	69429	75196	443343	90	37.68	45	22.71	4	8.28	4.55	2.74
IBM11	70558	81454	421424	120	53.50	50	23.57	3	6.90	7.76	3.42
IBM12	71076	77240	462449	90	41.97	45	21.89	3	7.05	5.95	3.11
IBM13	84199	99666	539781	100	57.46	50	29.75	3	8.19	7.01	3.63
IBM14	147605	152772	843858	240	208.69	60	68.81	3	15.90	13.13	4.33
IBM15	161570	186608	1097957	390	502.37	65	90.26	3	16.07	31.27	5.62
IBM16	183484	190048	1175600	290	385.48	60	92.11	3	21.28	18.11	4.33
IBM17	185495	189581	1264240	230	361.83	60	99.63	3	24.67	14.67	4.04
IBM18	210613	201920	1273165	770	1184.77	60	112.91	4	30.96	38.27	3.65

partitioning. Some key points of this method are as follows:

- Edge clustering. Each time we choose an edge to collapse.
- Random seed. Each time we randomly choose a seed cluster, and try to cluster it with another cluster that is connected.
- Edge weight based cost function. For a clustering seed, we choose the edge with the largest weight to collapse.
- Slow clustering scheme: The cluster number ratio between two adjacent levels is constrained to be 1.7.
- Tie breaking: When two edges have the same cost, we break the tie by favoring the edge connecting to a smaller cluster.

4.3 Random Clustering

The random clustering is the same with the hMETIS clustering except that the edge is chosen randomly each time for collapsing.

5. EXPERIMENT RESULTS

We implemented the algebraic multigrid solver and different clustering methods in C. These algorithms were tested using a set of placement benchmarks published in ISPD 2002 [13]. The statistical information of the benchmarks is listed in table 1. Equation (2) in section 2.2 was solved to test the different solvers. The experiments were run on a Sun Ultra 60 workstation with 360 MHz CPU and 512 Mega-bytes memory.

We combined the algebraic multigrid solver described in section 3 with the clustering methods in section 4, and tested the following three algorithms: **AMG-L, AMG-H** and **AMG-R** are the algebraic multigrid solvers with layout based clustering, with hMETIS clustering, and with random clustering, respectively.

For the AMG-L algorithm, we set ω to be 1.8 for the SOR smoothing in each level. We set α to be 0.15 for equation (11).

The first experiment compared the performance of the AMG-L solver with an SOR and a PCG solver. The SOR solver is based on an algorithm in [12]. We set the ω to be 1.95 for the SOR solvers. The PCG solver is based on an algorithm in [12]. It adopts incomplete Cholesky factorization as the preconditioner.

To test the convergence of the solvers, we examined the relative error $e_r = \|X-X^*\|_\infty/\|X^*\|_\infty$ on the fine level solution X. The exact solution X* was got by running AMG-L solver for a sufficient long time. We made check points every V-cycle for AML-L solver, every 10 iterations for SOR solver and every 5 iterations for PCG solver to get the relative error e_r.

Table 2. CPU time for solvers to reduce relative error to 10^{-3}

Circuit Name	SOR		PCG		AMG-L	
	#iter	Time(s)	#iter	Time(s)	#V-cyc	Time(s)
IBM01	120	4.54	45	2.98	10	2.84
IBM02	130	10.79	35	3.88	6	3.32
IBM03	120	11.66	35	4.34	7	4.16
IBM04	130	16.00	40	5.73	7	5.14
IBM05	140	19.03	20	3.53	7	6.64
IBM06	120	19.72	35	6.56	5	5.18
IBM07	120	31.33	55	15.41	5	7.81
IBM08	130	35.33	40	12.39	5	9.47
IBM09	220	71.04	55	20.11	7	11.96
IBM10	140	57.85	60	29.67	6	16.16
IBM11	180	79.05	60	29.71	6	14.30
IBM12	130	59.86	55	27.58	6	15.69
IBM13	140	81.27	60	37.36	5	17.39
IBM14	350	314.79	75	84.75	6	34.42
IBM15	590	757.81	75	109.24	6	40.94
IBM16	440	591.25	80	124.44	5	43.83
IBM17	350	553.20	70	118.12	6	51.25
IBM18	-	-	80	149.90	7	67.29

We compared the CPU time needed for each solver to reduce the relative error to 10^{-2} and 10^{-3} in solving equation (2) one time. The results are listed in table 1 and table 2, respectively. In Figure 6, we plot the CPU time to reach 10^{-2} relative error for all the solvers on various test cases. In Figure 7, we give the convergence histories for the three solvers on the test case IBM17. For AMG-L solver, the clustering time is excluded from the CPU time, since the clustering time can be amortized because the clustering is done once for solving all the equations (3). For PCG solver, we also exclude the CPU time for calculating the precondition matrix.

The results show that AMG method converges much faster than PCG and SOR at the beginning several iterations, and converges consistently at different error values. To reach 10^{-2} relative error, the AMG-L solver runs up to 38 times faster than the SOR solver, and 5 times faster than PCG solver on large cases. For 10^{-3} relative error, AMG solver also outperforms PCG solver by a factor of 2 to 3 on large cases. The results verify that for AMG solver, the number of V-cycles needed to converge is independent of the matrix size. This almost constant number of V-cycles strongly demonstrates that the method is scalable for huge designs.

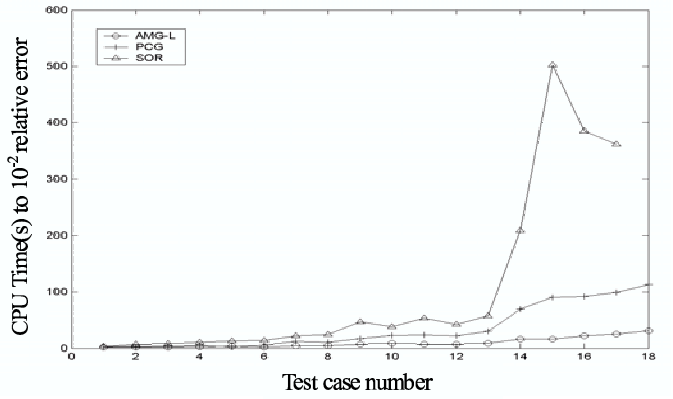


Fig. 6 Convergence time comparison for three solvers

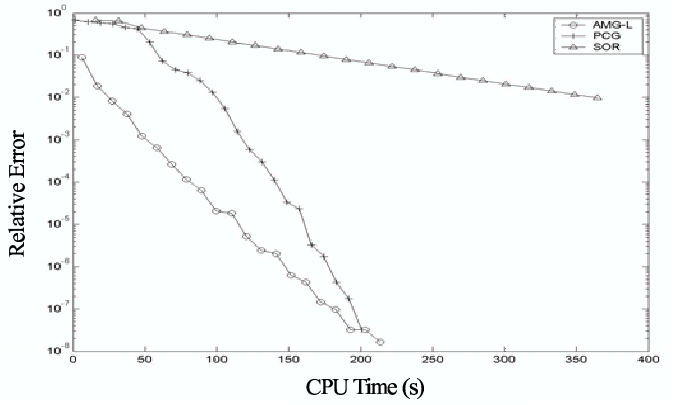


Fig. 7 Convergence histories for test case IBM17

The second experiment tests the impact of different clustering method. We test the AMG-L, AMG-H and AMG-R algorithms. The implementations of the multigrid iterations are the same in all three algorithms. They differ only in the way they do the clustering. The CPU time each algorithm needs to get a 10^{-3} relative error is reported in table 3. The clustering time is the time spent for deriving the whole clustering hierarchy. For AMG-L algorithm, the clustering time also covers the time to find the blown up placements.

The results show that the clustering method makes a difference in the convergence. Layout based clustering outperforms hMETIS clustering in large cases. Both layout based and hMETIS clustering outperform random clustering. Comparing the results in table 3 to table 2, we find that algebraic multigrid method works better than the SOR method even with random clustering.

The clustering speed was not the goal so far. The random clustering sometimes takes longer time than hMETIS clustering because hMETIS clustering tends to pick edges connecting un-clustered parts first, which makes the clustering faster.

6. CONCLUSION

The multilevel methods are more efficient to solve the huge linear equation system. We apply the algebraic multigrid method to solve the huge linear equations arise from the analytical placement. A layout based clustering scheme is proposed to generate coarsening levels for arbitrary circuit structures. The experimental results show that algebraic multigrid solver converges faster than PCG solver, and much faster than the one-level SOR solver on large test cases.

Table 3. CPU time for different AMG algorithms to $e_r < 10^{-3}$

Circuit Name	AMG-L		AMG-H		AMG-R	
	Clst Time(s)	Solving Time(s)	Clst Time(s)	Solving Time(s)	Clst Time(s)	Solving Time(s)
IBM01	4.58	2.84	2.74	1.79	2.32	3.17
IBM02	15.71	3.32	10.79	3.84	8.43	5.57
IBM03	8.05	4.16	7.02	3.65	7.52	5.16
IBM04	19.0	5.14	7.48	4.77	6.67	7.84
IBM05	19.92	6.64	20.89	6.24	19.12	9.46
IBM06	10.64	5.18	10.9	6.42	16.05	7.30
IBM07	41.92	7.81	14.39	15.78	17.18	19.35
IBM08	33.81	9.47	39.06	12.97	33.39	17.39
IBM09	70.15	11.96	14.87	17.23	19.94	28.21
IBM10	80.41	16.16	28.5	21.95	36.37	36.83
IBM11	106.86	14.30	18.19	16.42	27.09	25.40
IBM12	78.8	15.69	34.87	21.63	48.73	32.75
IBM13	103.32	17.39	25.95	26.70	41.88	72.48
IBM14	455.45	34.42	57.0	77.41	103.9	96.14
IBM15	339.54	40.94	73.64	126.97	182.6	174.90
IBM16	427.86	43.83	86.47	138.86	191.8	130.51
IBM17	384.51	51.25	117.68	68.88	268.4	116.51
IBM18	319.22	67.29	112.26	259.65	186.1	237.36

7. ACKNOWLEDGMENTS

This work was supported in part under grants from NSF project number MIP-9987678, the California MICRO program and SRC support.

8. REFERENCES

[1] H. Eisenmann, F. M. Johannes, Generic global placement and floorplanning. Proc. 35th Design Automation Conference, San Francisco, California, 1998, pp. 269-274.

[2] C. J.Alpert, et al, Quadratic Placement Revisited, Proc. 34th Design Automation Conference, Anaheim, California, 1997, pp. 752-757.

[3] J. M. Kleinhans, et al, GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization, IEEE Trans. Computer-Aided Design, Vol. 10, No. 3, Mar. 1991, pp. 356-365.

[4] X. Hong, et al, CASH: A Novel Quadratic Placement Algorithm for Very Large Standard Cell Layout Design Based on Clustering, Proc. ASP-DAC, 2000, Yokohama, Japan, pp. 271-276.

[5] G. Karypis and V. Kumar, Multilevel k-way Hypergraph partitioning, Univ. of Minnesota, Technical Report #98-036.

[6] T. Chan, J. Cong, T. Kong and J. Shinnerl, Multilevel Optimization for Large-scale Circuit Placement, *Proc. IEEE International Conference on Computer Aided Design*, San Jose, California, pp. 171-176, Nov. 2000.

[7] C. J. Alpert, et al, Multilevel Circuit Partitioning, IEEE Trans. CAD, Vol. 17, No. 8, August, 1998, pp. 655-667

[8] C. J. Alpert and A.B.Kahng, Recent Directions in Netlist Partitioning: A Survey, Integration, the VLSI Journal, 19(1-2), 1995, pp. 1-81.

[9] W. L. Briggs, V. E. Henson, and S. F. McCormick, A Multigrid Tutorial, 2nd Ed. SIAM, 2000.

[10] K. Stüben， A Review of Algebraic Multigrid, GMD Report No. 69. Nov. 1999.

[11] J. N. Kozhaya, S. R. Nassif, F. N. Najm, Multigrid-like Technique for Power Grid Analysis. Proc. ICCAD 2001, pp. 480-487.

[12] G. H. Golub and C. F. V. Loan, Matrix Computations, 2nd edition. Johns Hopkins, 1993.

[13] S.N. Adya, I.L. Markov, Consistent Placement of Macro-Blocks Using Floorplanning and Standard-Cell Placement, Proc ISPD 2002, pp.12-17.

[14] P. Vanek, J. Mandel, M. Brezina, Algebraic Multigrid by Smoothed Aggregation for Second and Fourth Order Elliptic Problems, Computing 56, 1996, pp. 179-196.