ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Χωροθέτηση Ολοκληρωμένων Κυκλωμάτων με τη Μέθοδο Gordian

Διπλωματική Εργασία

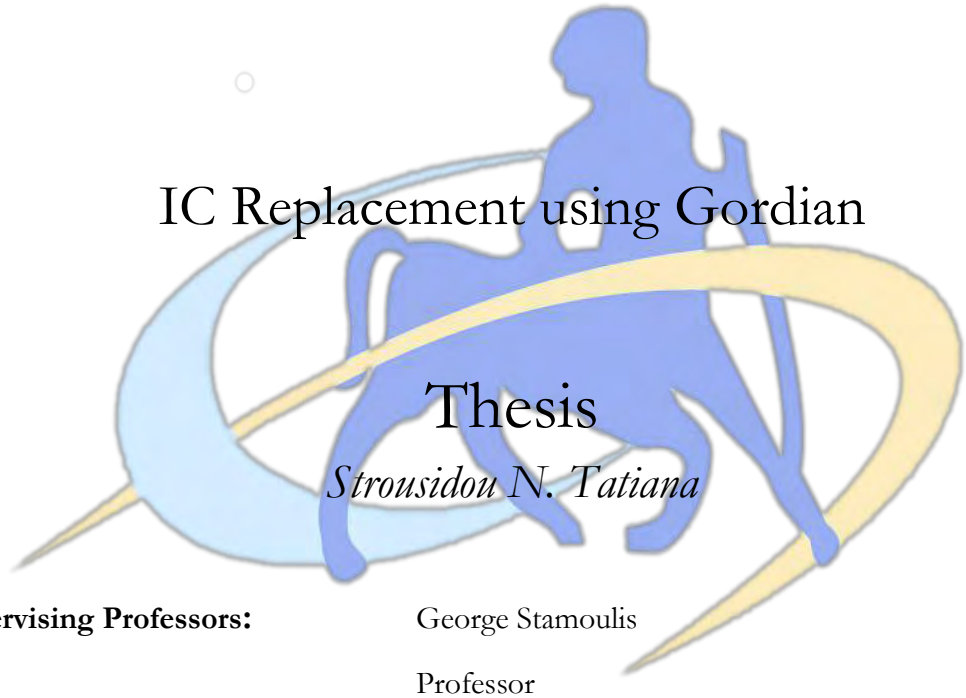
Στρουσίδου Ν. Τατιάνα

Επιβλέποντες Καθηγητές: Σταμούλης Γεώργιος
Καθηγητής

Ευμορφόπουλος Νέστωρ
Επίκουρος Καθηγητής

University of Thessaly

DEPARTMENT OF COMPUTER AND COMPUTER ENGINEERING



IC Replacement using Gordian

Thesis

Strousidou N. Tatiana

Supervising Professors: George Stamoulis

Professor

Nestor Evmorfopoulos

Assistant Professor

Approved by the two-member inquiry committee at 26 February 2015

……..	……..
George Stamoulis	Nestor Evmorfopoulos
Professor	Assistant Professor

Διπλωματική Εργασία για την απόκτηση του Διπλώματος του Ηλεκτρολόγου Μηχανικού και Μηχανικού Υπολογιστών, του Πανεπιστημίου Θεσσαλίας, στα πλαίσια του Προγράμματος Προπτυχιακών Σπουδών του Τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Πανεπιστημίου Θεσσαλίας.

.....

Τατιάνα Στρουσίδου

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών

Πανεπιστημίου Θεσσαλίας

To my family and friends

Στην οικογένειά μου και στους φίλους μου

Acknowledgements

I would like to express my wholehearted gratitude to my supervisor, Dr. George Stamoulis, who trusted me in dealing with this thesis. His patience, expertise and advising, contributed to my graduate experience and finally this thesis's completion. I strongly appreciate his in-depth knowledge and skills.

Furthermore, I would like to thank Dr. Nestor Evmorfopoulos and Charalampos Antoniadis for their assistance and support they provided at all levels of this thesis.

Finally, I have to thank my friends and especially my parents, my brother and my sister for their endless and invaluable moral support that offered me all those academic years.

Tatiana N. Strousidou

Volos, 2015

Abstract

IC Replacement using Gordian

Tatiana N. Strousidou

The University of Thessaly at Volos, 2015

Supervisors: George Stamoulis

Nestor Evmorfopoulos

Placement is considered a fundamental physical design problem in electronic design automation. It has been around so long that it is commonly viewed as a solved problem. Placement quality is at the heart of design quality in terms of timing closure, routability, area, power and most importantly, time-to-market. Additionally, the increasing scale of placement instances affects the algorithms of choice for high-performance tools. In this thesis we present the GORDIAN algorithm for global placement as well as an optimization of it with which we managed to define each cell's placement applying an alternative method.

Table of Contents

Abbreviations

EDA	**E**lectronic **D**esign **A**utomation
HPWL	**H**alf **P**erimeter **W**ire **L**ength
SPD	**S**ymmetric **P**ositive **D**efinite
CG	**C**onjugate **G**radients
QP	**Q**uadratic **P**rogrammin

1. Introduction

In this thesis we deal with GORDIAN algorithm [1], a method for global placement of standard-cell based circuit designs. More specifically, we present in detail each step of the algorithm focusing on the solution which uses the null space of a matrix. In the framework of this thesis, we developed an alternative solution method, which is based on the least squares' method, the algorithm and examples of which are thoroughly presented later.

The rest of this thesis is organized as following: Section 2 provides an overview of the placement problem. Section 3 introduces the GORDIAN algorithm. Section 4 describes the alternative solution method as well as some results of it. Section 5 gives information about the negative result of this method. Finally, in section 7 least squares' method is explained.

2. Introduction to Placement

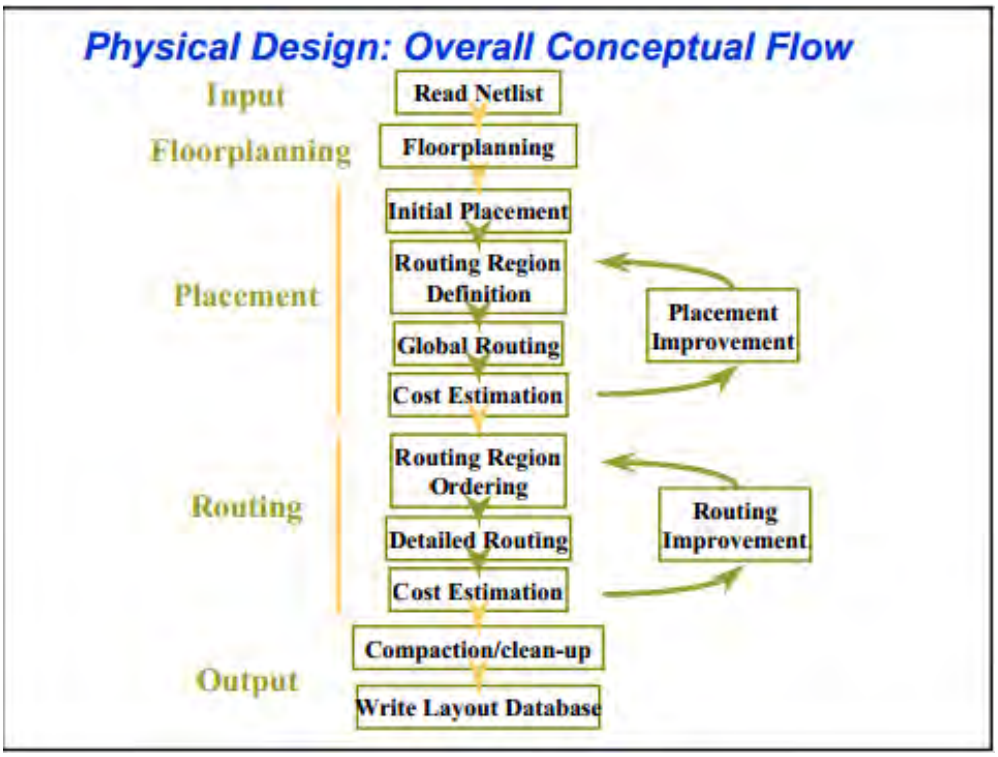
2.1. Placement problem formulation

Placement [26] is an essential step in electronic design automation - the portion of the physical design flow that assigns exact locations for various circuit components within the chip's core area. An inferior placement assignment will not only affect the chip's performance but might also make it nonmanufacturable by producing excessive wire length, which is beyond available routing resources. Consequently, a placer must perform the assignment while optimizing a number of objectives to ensure that a circuit meets its performance demands. Typical placement objectives include:

- Total wire length: Minimizing the total wire length, or the sum of the length of all the wires in the design, is the primary objective of most existing placers. This not only helps minimize chip size, and hence cost, but also minimizes power and delay, which are proportional to the wire length (This assumes long wires have additional buffering inserted; all modern design flows do this.)
- Timing: The clock cycle of a chip is determined by the delay of its longest path, usually referred to as the critical path. Given a performance specification, a placer must ensure that no path exists with delay exceeding the maximum specified delay.
- Congestion: While it is necessary to minimize the total wire length to meet the total routing resources, it is also necessary to meet the routing resources within various local regions of the chip's core area. A congested region might lead to excessive routing detours, or make it impossible to complete all routes.
- Power: Power minimization typically involves distributing the locations of cell components so as to reduce the overall power consumption, alleviate hot spots, and smooth temperature gradients.
- A secondary objective is placement runtime minimization.

2.2. Placement within the EDA design flow

A placer takes a given synthesized circuit netlist together with a technology library and produces a valid placement layout. The layout is optimized according to the aforementioned objectives and ready for cell resizing and buffering - a step essential for timing and signal integrity satisfaction. Clock-tree synthesis and routing follow, completing the physical design process. In many cases, parts of, or the entire, physical design flow are iterated a number of times until design closure is achieved.



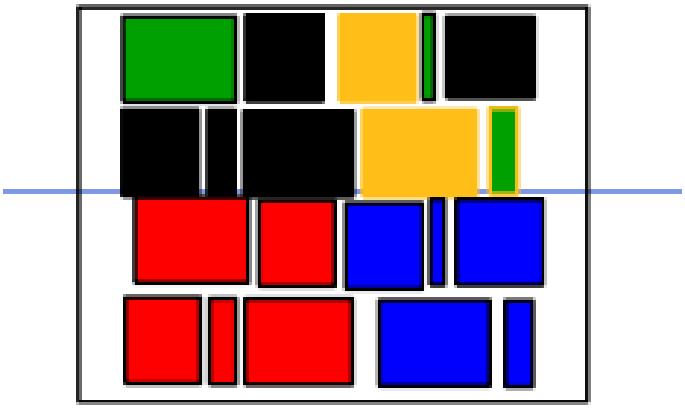
3. Gordian Overview

A. GORDIAN [7] is one of the most successful Global placement algorithms. With GORDIAN, the placement problem is formulated as a sequence of quadratic programming problems [29] derived from the entire connectivity information of the circuit. A set of constraints is imposed and as a result the reduction of the amount of overlap among the cells is achieved in a gradual way.

B. GORDIAN focuses on row oriented standard-cell placement. A standard cell represents the physical space occupied by a logical gate. In standard-cell placement all cells must be of the same height, as shown in picture 2.

C. GORDIAN does not use partitioning to reduce the problem size, but to restrict the freedom of movement of the modules

D. GORDIAN [5] uses a combination of the Analytical and Minimum cut approaches. Initially, the first step is applied globally without taking into consideration any library constraints (cell overlaps, cell outside core area, etc.) additional to minimizing the total wire length. A top-down partitioning strategy follows. Top-down partitioning-based placement algorithms seek to decompose a given placement instance into smaller instances by sub-dividing the placement region, assigning modules to subregions and cutting the netlist hypergraph. By recursively dividing the core area and assigning circuit elements to every partition, the overlaps are reduced to a point where a simple legalization algorithm can produce a legal solution.



Picture 2: standard-cell placement

```
Gordian Procedure:
l := 1
global_optimize( l );
while ( there exists |Ml|>K )
        for each r
                partition( r, r', r'' );
        l ++;
        setup_constraints( l );
        global_optimize( l );
        re-partition( l );
final_placement( l );
end_procedure;
```

Gordian Algorithm Pseudocode

3.1 Quick overview

Global optimization step

Before applying the global optimization step, an undirected weighted graph is constructed from the input circuit. Each net of size *k* in the netlist forms a *k*-clique, and all the edges in the clique receives a weight of 2/*k*, as shown on Picture 3 and Picture 4. The movable cells and fixed pins constitute the nodes of the graph [7][5].

Using the weight information, the following matrices are formed:

- *Adjacency matrix* A_{nxn}, where $A_{i,j} = W_{c_{i,j}}$
- *Pin Connection Matrix* P_{nxm}, where $P_{i,j} = W_{p_{i,j}}$
- *Degree Matrix* D_{nxn}, where $D_{i,j} = \begin{cases} \sum_{j=0}^{n} A_{i,j} + \sum_{j=0}^{m} P_{i,j} & , i = j \\ 0 & , i \neq j \end{cases}$
- *Fixed Pin Vectors* $d_{x_{mx1}}$ and $d_{y_{mx1}}$, where $d_{x_i} = -\sum_j P_{i,j}\, x_j$ and $d_{y_i} = -\sum_j P_{i,j} y_j$

Where:

- n is the number of cells
- m is the number of pins
- $W_{c_{i,j}}$ is the weight of the edge connecting cell *i* and cell *j*
- $W_{p_{i,j}}$ is the weight of the edge connecting cell *i* and pin *j*
- x_j is the x-coordinate of pin j
- y_j is the y-coordinate of pin j

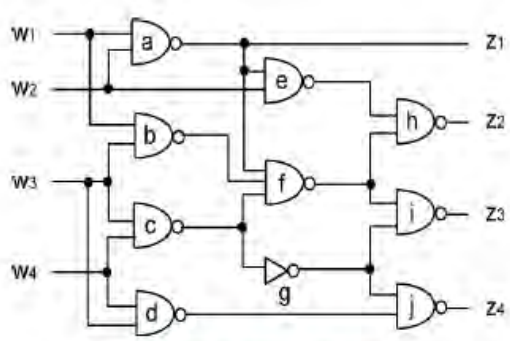
The following quadratic programming problem is stemmed from the first global step of the placement algorithm:

$$\varphi(x) = \frac{1}{2} x^T C x + d x^T x$$

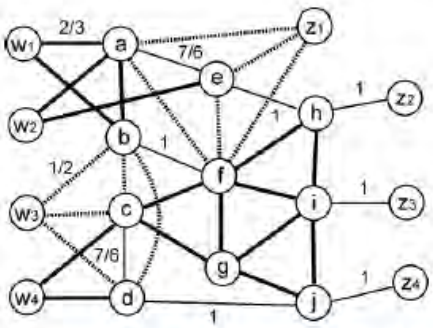
and:

$$\varphi(y) = \frac{1}{2} y^T C y + d y^T y$$

Where dx and dy are the *Fixed Pin* vectors and C is the *Laplacian* matrix.



Picture 3: Input circuit



Picture 4: Undirected weighted graph

Top-down partitioning

Firstly, the placement region is partitioned into two subregions, using either vertical or horizontal cutline, each containing a subset of movable cells. The following steps follow the same procedure resulting to additional cutline being applied in the subregions. At the i_{th} level of optimization, the placement area is divided up into at most $q \leq 2^i$ regions.

The vectors that contain the center location of these sub-partitions are computed by the equations below:

$$A^i x = u$$

And

$$A^i y = u$$

Where *i* is the number of partitioning step.

Subsequently, the constraints matrix A_{qxm} has to be computed whose contents are:

$$A_{i,j} = \begin{cases} 1, & cell\ j\ belongs\ to\ partition\ i \\ 0, & otherwise \end{cases}$$

The linearly constrained quadratic programming problems that have to be solved are the following:

$$\varphi(x) = \left\{ \frac{1}{2} x^T C x + d x^T \; x \middle| A^i x = u \right\}$$

and

$$\varphi(y) = \left\{ \frac{1}{2} y^T C y + d y^T \, y \middle| A^i y = u \right\}$$

where A is the constraints matrix, C is the Laplacian matrix and u are the x and y coordinates of the partitions respectively.

As a result, the following linear systems must be solved:

$$\begin{bmatrix} C & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x \\ \lambda \end{bmatrix} = \begin{bmatrix} -dx \\ u_x \end{bmatrix}$$

and

$$\begin{bmatrix} C & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} y \\ \lambda \end{bmatrix} = \begin{bmatrix} -dy \\ u_y \end{bmatrix}$$

Final Placement

- GORDIAN terminates after an enough number of cutlines are added so that the size of the partitions is small enough, i.e., it is used for global placement. A final placement is the last, but possibly most important, step in the GORDIAN Algorithm
- After the main body of the GORDIAN [1][7] algorithm finishes, which is the alternating global optimization and partitioning steps, each of the blocks containing k or less modules needs to be optimized.
- For the Standard Cell Design the modules are collected in rows, for the macro-cell design an area optimization is performed, packing the modules in a compact slicing structure.

4. Linear Constraints Elimination

4.1 Conjugate gradient (CG)

The conjugate gradient method [27] derives its name from the fact that it generates a sequence of conjugate (or orthogonal) vectors. These vectors are the residuals of the iterations. They are also the gradients of a quadratic functional, the minimization of which is equivalent to solving the linear system. Conjugate gradient (CG) is an extremely effective method when the coefficient matrix is symmetric positive definite (SPD) [30], since storage for only a limited number of vectors is required.

4.2 Linear System Solvers

The iterative methods are more efficient to use for solving the generated linear systems than the direct ones for reasons of time complexity. In comparison to direct solvers that usually have a time complexity of $O(N^2)$, iterative solvers have nearly linear to the size of the system time complexity.

More specifically the methods used are:

- The *Conjugate Gradient (CG)* method [27] for solving the linear system generated during the global optimization step.
- The *Bi-Conjugate Gradient (Bi-CG)* method [31] for solving the linear systems generated during the top-down partitioning procedure.

Unlike *Bi-CG*, the *CG* method can only be implemented for *Symmetric Positive Definite* (SPD) matrices. The matrix generated during global optimization step is SPD as long as the whole circuit is connected to the I/O pins. However, the matrices generated during the partitioning steps are not positive definite (and thus non-SPD) due to the zero elements on the diagonal introduced by the constraints matrix.

4.3 Solution method description

The iterative method used to solve the generated linear systems is *Conjugate Gradient (CG)* method [6]. A variety of algorithms for linearly and nonlinearly constrained optimization use the *Conjugate Gradient (CG)* method to solve subproblems of the form

$$minimize_x \; q(x) = \frac{1}{2}x^T Hx + c^T x$$

$$\text{subject to} \quad Ax = b$$

We will assume here that A is an $m \times n$ matrix, with $m < n$, and that A has full row rank so that the constraints $Ax = b$ constitute m linearly independent equations. Generally, constraints force cells to move towards the center of their partition. But, on the other hand, this quadratic program can be solved by computing a basis for the null space of A, using this basis to eliminate the constraints, and then applying the CG method to the reduced problem. More specifically, consider that Z is an $n \times (n - m)$ matrix spanning the null space of A. Then $AZ = 0$, the columns of A^T together with the columns of Z span $\boldsymbol{R}^n$, and any solution x^* of the linear equations $Ax = b$ can be written as

$$x^* = A^T x_A^* + Z x_Z^*,$$

where

$$AA^T x_A^* = b,$$

and

$$C_{ZZ}x_Z = -c_Z\,, \quad C_{ZZ} = Z^T C Z, \quad c_Z = Z^T(CA^T x_A^* + c)$$

Z is a matrix spanning the null space of A, *b* is the gravity center of x and y-axis respectively and c is the fixed pin vector for X and Y direction respectively.

With this method we managed to transform the initial constrained problem into an unconstrained one. However, we are still facing a significant problem with the C_{ZZ} matrix which remains significantly full, which increases the total runtime and memory requirements. At this point, it is necessary to explain the exact method of the computation of the null space of A.

4.4 Computing the Null Space of Linear Constraints Matrix A

Due to the special format of the constraints matrix A, its null space N(A) can be found easily. The special format of A resides on the fact that:

1) The non-zero elements existing in a row is the same number.
2) The non-zero elements appear in different columns.

Below we provide the algorithm of finding the Null Space of matrices which have the same structure with A:

a) Generally, we fill the columns of Z by elaborating the rows of the constraints' matrix A.
b) If there are only two non-zero elements in an elaborating row, only one column of Z can be computed. So, by processing the first row of A, we can construct the first column of Z etc. As a result, -1 is inserted in this position of the under construction column of Z where the first non-zero element in the elaborating row of A is located. Equally, 1 is placed in Z in the same position with the following non-zero element of A matrix.
c) Generalizing this procedure, if the elaborating row has **N** non-zero elements, **N-1** columns of Z can be reckoned by filling them with -1, 1 or 0 in the appropriate positions.
d) For example, the existence of three non-zero elements in a row of A means that two consecutive columns of Z can be constructed. The method is the same with this one explained above. As mentioned before, in both columns of Z that are expected to be constructed, there is -1 in the same position with the first appearance of the non-zero element in the specific row of A. On this occasion, the first column of Z has 1 in the row where the next non-zero element of A can be found and there is also 1 in the second column of Z in this place where the last non-zero element is located in the examining row of A.

In order to get a better understanding of the algorithm we previously described, we provide an example of null space computation.

Suppose we have the following matrix A:

$$A = \begin{bmatrix} 0 & 0 & 1/2 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/3 & 1/3 & 0 & 0 & 1/3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 0 & 1/2 \\ 0 & 0 & 0 & 0 & 0 & 1/3 & 0 & 1/3 & 1/3 & 0 \end{bmatrix}$$

Beginning from the first row we notice that there are only two non-zero elements, specifically in the 3rd and 4th place. So, -1 is inserted in the 3rd position of the first column of Z and 1 in the 4_t^h. The remaining positions are filled with zeros. Subsequently, in the second row of A exist three non-zero elements, in the 1st, 2nd and 5th position respectively, so we can construct the following two columns of Z. Therefore, the second column of Z has -1 in the 1st place and 1 in the 2nd one while the third column of Z has also -1 in the 1st row and 1 in the 5th one. The remaining columns of Z can be similarly constructed.

4.5 Results

This particular method of computing the null space of the constraints' matrix A, which was described and explained above, is much advantageous. Indeed, by making use of the special format of A matrix, we managed to construct the null space Z of A without any cost in a very simple way. In addition, Z is a sparse matrix, which is a very useful and special property because the majority of computations including such a matrix become really simple and fast. Another important reason why a sparse matrix is more practical than a dense one is that in the case of a sparse matrix, substantial memory requirement reductions can be realized by storing only the non-zero entries. Finally, the inversion of a sparse matrix is not a time-consuming procedure, which under different circumstances could be.

5. The density of C_{ZZ}

The alternative solution method of the quadratic program that was thoroughly exposed in chapter [2], reveals a crucial problem that needs to be solved. This problem is referred to the fact that C_{ZZ} matrix is full, making the use and implementation of sparse matrices' properties impossible. Thus, this particular characteristic of C_{ZZ} matrix not only renders all computations remarkably time-consuming, but it also leads to the extensive use of memory. However, there is a way, including some conceivable steps, with which we can prevent the C_{ZZ} from being dense. We focus on the fact that we want to solve the equation $C_{ZZ}x_Z = -c_Z$. So, the steps for this solution are described in detail below:

First of all we have to solve the equation below:

$$C_{ZZ}x_Z = -c_Z$$

Then we substitute with:

$$C_{ZZ} = Z' * C * Z,$$

So we have:

$$Z' * C * Z * x_Z = -c_Z$$

Afterwards we set

$$Z * x_Z = w$$

So, our system can be written in the way presented below:

$$Z' * C * w = -c_Z \quad and \quad Z * x_Z = w$$

Now we set

$$C * w = w'$$

So the system can be written:

$$A) Z' * w' = -c_Z \quad B) C * w = w' \quad C) Z * x_Z = w$$

Where C is the Laplacian matrix and Z' is the transpose of Z.

As a result, we are supposed to solve these three linear systems described above in order to find finally x_Z. The initial system $C_{ZZ}x_Z = -c_Z$ can be solved using CMG preconditioner. So, we focus on the solution of linear systems with matrix Z and Z', and we deal with the over-constrained and under-constrained problem respectively which is solved by least squares method.

6. Preconditioning

Definition of preconditioner

In mathematics, preconditioning [28] is a procedure of an application of a transformation, called the preconditioner, which conditions a given problem into a form that is more suitable for numerical solving methods. Preconditioning is typically related to reducing a condition number of the problem. The preconditioned problem is then usually solved by an iterative method.

Preconditioning for linear systems

Preconditioners are useful in iterative methods to solve a linear system $Ax = b$ for x since the rate of convergence for most iterative linear solvers increases as the condition number of a matrix decreases as a result of preconditioning.

The Combinatorial Multigrid Solver (CMG)

The present subchapter describes the Combinatorial Multigrid Solver (CMG) [3].At the beginning, we give a short review of multigrid solvers and then we describe the basic components of CMG [8].

Related work on SDD solvers

Multigrid [8] was originally conceived as a method to solve linear systems that are generated by the discretization of the Laplace (Poisson) equation over relatively nice domains [14]. The underlying geometry of the domain leads to a hierarchy of grids $A = A_0, \dots, A_d$ that look similar at different levels of detail; the picture that the word multigrid often invokes to mind is that of a tower of 2D grids, with sizes $2^{d-i} \times 2^{d-i}$ for $i = 0, \dots, d$ Its provably asymptotically optimal behavior for certain classes of problems soon lead to an effort, known as Algebraic Multigrid (AMG), to generalize its principles to arbitrary matrices. In contrast to classical Geometric Multigrid (GMG) where the hierarchy of grids is generated by the discretization process, AMG constructs the hierarchy of "coarse" grids/matrices based only on the algebraic information contained in the matrix. Various flavors of AMG, based on different heuristic coarsening strategies, have been proposed in the literature. AMG has been proven successful in solving more problems than GMG, though some times at the expense of robustness, a byproduct of the limited theoretical understanding.

A solver with provable properties for arbitrary SDD matrices, perhaps the "holy grail" of the multigrid community, was discovered only recently. The path to it was Support Theory [10], a set of mathematical tools developed for the study of combinatorial subgraph preconditioners, originally introduced by Vaidya [14] [12] .It has been at the heart of the seminal work of Spielman and Teng [13] who proved that SDD systems can be solved in nearly-linear time. Koutis and Miller [15] proved that SDD matrices with planar connection topologies (e.g. 4-connectivity in the image plane) can be solved asymptotically optimally, in $O(n)$ time for n-dimensional matrices. The complexity of the Spielman and Teng solver was recently significantly improved by Koutis, Miller and Peng [16] [17], who described an O ($mlog\ n$) algorithm for the solution of general SDD systems with m non-zero entries. It is fair to say that these theoretically described solvers are still impractical due to the large hidden constants, and the complicated nature of the underlying algorithms. Combinatorial Multigrid (CMG) [18] is a variant of multigrid that reconciles theory with practice. Similarly to AMG, CMG builds a hierarchy of matrices/graphs. The essential difference from AMG is that the hierarchy is constructed by viewing the matrix as a graph, and using the discrete geometry of the graph, for example notions like graph separators and expansion. It is, in a way, a hybrid of GMG and AMG, or a discrete-geometric MG. The re-introduction of geometry into the problem allows us to prove sufficient and necessary conditions for the construction of a good hierarchy and claim strong convergence guarantees for symmetric diagonally dominant (SDD) matrices based on recent progress in Steiner preconditioning [19] [20] [21].

SDD linear systems as graphs

In this subsection we discuss how SDD linear systems can be viewed entirely as graphs. Combinatorial preconditioning advocates a principled approach to the solution of linear systems. The core of CMG and all other solvers designed in the context of combinatorial preconditioning is in fact a solver for a special class of matrices, graph Laplacians. The Laplacian A of a graph $G = (V, E, w)$ with positive weights, is defined by:

$$A_{i,j} = A_{j,i} = -w_{i,j} \; and \; A_{i,i} = -\sum_{i \neq j} A_{i,i}$$

More general systems are solved via light-weight transformations to Laplacians. Consider for example the case where the matrix A has a number of positive off-diagonal entries, and the property $A_{i,i} = \sum_{i \neq j} A_{i,j}$. Positive off-diagonal entries have been a source of confusion for AMG solvers, and various heuristics have been proposed. Instead, CMG uses a reduction known as double-cover [19]. Let $A = A_p + A_n + D$, where D is the diagonal of A and A_p is the matrix consisting only of the positive off-diagonal entries of A. It is easy to verify that

$$Ax = b \Leftrightarrow \begin{pmatrix} D + A_n & -A_p \\ -A_p & D + A_n \end{pmatrix} \begin{pmatrix} x \\ -x \end{pmatrix} = \begin{pmatrix} b \\ -b \end{pmatrix}$$

In this way, the original system is reduced to a Laplacian system, while at most doubling the size. In practice it is possible to exploit the obvious symmetries of the new system, to solve it with an even smaller space and time overhead.

Matrices of the form $A + D_e$, where A is a Laplacian and D_e is a positive diagonal matrix have also been addressed in various ways by different AMG implementations. In CMG, we again reduce the system to a Laplacian. If de is the vector of the diagonal elements of D, we have

$$Ax = b \Leftrightarrow \begin{pmatrix} A + D_e & 0 & -d_e \\ 0 & A + D_e & -d_e \\ -d_e^T & -d_e^T & \sum_i d_e(i) \end{pmatrix} \begin{pmatrix} x \\ -x \\ 0 \end{pmatrix} = \begin{pmatrix} b \\ -b \\ 0 \end{pmatrix}$$

Again it's possible to implement the reduction in a way that exploits the symmetry of the new system, and with a small space and time overhead work only implicitly with the new system.

A symmetric matrix A is called diagonally dominant (SDD), if $A_{i,i} \geq \sum_{i \neq j} |A_{i,j}|$. The two reductions above can reduce any SDD linear system to a Laplacian system. Symmetric positive definite matrices (SPD) with non-positive off-diagonals are known as M-matrices. It is well known that if A is an M -matrix, there is a positive diagonal matrix D such that $A = DLD$ where L is a Laplacian. Assuming D is known, an M -system can also be reduced to a Laplacian system via a simple change of variables. In many application D is given, or it can be recovered with some additional work [22].

There is a one-to-one correspondence between Laplacians and graphs, so we will be often using the terms interchangeably.

7. Least squares method

The method of **least squares** [24] is a standard approach to the approximate solution of overdetermined systems, sets of equations in which there are more equations than unknowns. "Least squares" means that the overall solution minimizes the sum of the squares of the errors made in the results of every single equation. Generally, the steps that are followed in order to find a least square solution are described below:

Suppose that we want to find a solution for the linear system $x = R * \theta$, where θ is the unknown factor of the equation.

Multiply both parts of the equation with R^T. So, we have

$$R^T * x = R^T * R * \theta$$

Rearrange the parentheses

$$R^T * x = (R^T * R) * \theta$$

$$\theta = (R^T R)^{-1} R^T x$$

In our case, only two of the three systems should be solved with least squares method,

$$Z' * w' = -c_Z$$

and

$$Z * x_Z = w$$

,as they are the two over-constrained systems that appear. More specifically, the first system's solution procedure, according to the steps described above, is cited below:

$$Z^T * w' = -c_Z$$
$$Z * Z^T * w' = -Z * c_Z$$
$$(Z * Z^T) * w' = -Z * c_Z$$
$$w' = -(Z * Z^T)^{-1} * Z * c_Z$$

Similarly, the solution of the second system follows the same process:

$$Z^T * Z * x_Z = Z^T * w$$
$$(Z^T Z) * x_Z = Z^T * w$$
$$x = (Z^T * Z)^{-1} * Z^T * w$$

If we examine carefully the steps that solve the two linear systems above, it is easily observed that the only time-consuming factor could be the computation of the inverse of $Z^T Z$. In the following chapter we are going to explain why the procedure of this inversion does not require much time and how simple the construction of this inverse is.

8. The inverse of $Z^T Z$

The result of the multiplication Z^T*Z is a matrix that has a special format and allows us to take advantage of it in order to compute the inverse $(Z^T*Z)^{-1}$ with simple and applicable steps. Suppose we have the matrix

$$A = [1/5\ 1/5\ 1/5\ 1/5\ 1/5\ 0\ 0\ 0\ 0\ 0$$
$$0\ 0\ 0\ 0\ 0\ 1/5\ 1/5\ 1/5\ 1/5\ 1/5]$$

By computing the multiplication Z^T*Z for our example, we take the following result:

$$Z^T * Z = \begin{bmatrix} 2 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 2 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 2 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 2 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 2 \end{bmatrix}$$

where Z is the null space of A described in chapter [2.1]

With a careful looking at this matrix, we realize that there are two same submatrices that have 2 in their main diagonals and 1 in the remaining positions. Finally, the outer matrix's upper and lower parts are filled with zeros. This means that those matrices analyzed above appear as block matrices of Z^T*Z.

At this point, it is necessary to describe extensively what the matrix A represents. As we can see, it is a 2x10 matrix whose dimensions show that there are 2 partitions with 5 cells in each partition. The 2 submatrices that are presented above are 4x4 matrices with this special format that has been previously explicated. The inverse of $Z*Z^T$ has the following format:

$$(Z * Z^T)^{-1} = [{}^{4}/_{5} \quad {}^{-1}/_{5} \quad {}^{-1}/_{5} \quad {}^{-1}/_{5}$$
$${}^{-1}/_{5} \quad {}^{4}/_{5} \quad {}^{-1}/_{5} \quad {}^{-1}/_{5}$$
$${}^{-1}/_{5} \quad {}^{-1}/_{5} \quad {}^{4}/_{5} \quad {}^{-1}/_{5}$$
$${}^{-1}/_{5} \quad {}^{-1}/_{5} \quad {}^{-1}/_{5} \quad {}^{4}/_{5}]$$

Now we will try to generalize the format of the inverse of
Z^T*Z, $\forall\ m$, where m the number of cells in each partition, by consulting the example that was given previously.
So, for partitions with *m* cells, the dimension of the Z^T*Z is *m-1*x*m-1*. This leads to the conjecture that there is the inverse of this matrix, as it is a square one, and it has the format below:

- The main diagonal of the matrix is filled with ${}^{(m-1)}/_{m}$ and
- Both the upper and the lower part of it is filled with ${}^{-1}/_{m}$

$$Z^T * Z = \begin{bmatrix} {}^{(m-1)}/_{m} & & {}^{-1}/_{m} \\ & \ddots & \\ {}^{-1}/_{m} & & {}^{(m-1)}/_{m} \end{bmatrix}$$

9. Conclusion

With this simple method that is applied for the inverse's computation of the Z^T*Z, it is achieved for a matrix to be inverted without any cost. Not to mention the fact that only three numbers need to be maintained in memory, as the non-zero elements of the Z^T*Z are always two in quantity. The other element that is helpful to be stored in memory is the number of cells that exist in each partition which provides information about the dimension of the matrix Z^T*Z.

All in all, by applying this method we have the opportunity to confirm that there is an easy, simple and costless way to compute the inverse of a matrix, taking advantage of the special format of its null space.

10. Future work

In the future, bigger circuits can be tested by applying this method in order to define their "reaction" to it. The results of this method is definite and obvious, but further research and optimization may lead to more spectacular effects. Furthermore, more and more different methods for the computation of an inverse could be examined and applied in order to check the results, comparing them with the results of the method presented in this thesis and finally deciding which is the appropriate for this problem.

11. References

[1] J. Kleinhans, G. Sigl, F. Johannes and K. Antreich*, "GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization", IEEE Transactions on Computer-Aided Design. 10 (3), March 1991, pp. 356-365.*

[2] G. Sigl, K. Doll, F.M. Johannes, *"Analytical Placement: A Linear or a Quadratic Objective Function?", DAC'91 pp 427-423.*

[3] I. Koutis, G. L. Miller and D. Tolliver, "*Combinatorial Preconditioners and Multilevel Solvers for problems in computer vision and image processing. Computer Vision and Image Understanding", 115(12):1638–1646, 2011.*

[4] I. Koutis and G. Miler, "*The Combinatorial multigrid solver*", in: Conference Talk, March, 2009.

[5] Sung Kyu Lim, "*Practical Problems in VLSI Physical Design Automation*", *Georgia Institut of Technology, Atlanta, pp. 112-121.*

[6] N. Gould, M. Hribar, J. Nocedal, "*On the Solution of Equality Constrained Quadratic Programming Problems Arising in Optimization", pp. 1-3*, *2000*.

[7] Stavros K. Ioannidis, "*Implementation and Optimization of an Integrated Circuit Placement Algorithm in Parallel Environment*", *2014.*

[8] Dimitrios K. Garifallou, "*Simulation of large-scale circuits with Steiner node preconditioners on parallel architectures*", *2014.*

[9] Antonis Dadaliaris, "*Χωροθέτηση Ολοκληρωμένων Κυκλωμάτων με Παραμέτρους Αξιοπιστίας*", *2012.*

[10] E. G. Boman and B. Hendrickson, "Support theory for preconditioning", SIAM J. Matrix Anal. Appl. 25 (3) (2003) 694-717.

[11] P. M. Vaidya, "*Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners*", A talk based on this manuscript was presented at the IMA Workshop on Graph Theory and Sparse Matrix Computation, October 1991.

[12] A. Joshi, "*Topics in optimization and sparse linear systems*", PhD thesis, Champaign, IL, USA, 1997. UMI Order No. GAX97-17289.

[13] D. A. Spielman and S.-H. Teng, "*Nearly-Linear Time Algorithms for Preconditioning and Solving Symmetric*", Diagonally Dominant Linear Systems, May 2007.

[14] U. Trottenberg, A. Schuller and C. Oosterlee, Multigrid. Academic Press, 1st edition, 2000.

[15] I. Koutis and G. L. Miller, "*A linear work, O(n1/6) time parallel algorithm for solving planar Laplacians*", In Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, SODA '07, pages 1002–1011, Philadelphia, PA, USA, Society for Industrial and Applied Mathematics, 2007.

[16] I. Koutis and G. L. Miller, "*Approaching optimality for solving*", August 2010.

[17] I. Koutis, G. L. Miller και R. Peng, "*Solving sdd linear systems in time O(mlognlog(1/ε))*", April 2011.

[18] I. Koutis and G. Miler, "*The Combinatorial multigrid solver*", in: Conference Talk, March, 2009.

[19] K. Gremban, "*Combinatorial Preconditioners for Sparse, Symmetric, Diagonally Dominant Linear Systems*", PhD thesis, Carnegie Mellon University, Pittsburgh, October 1996. CMU CS Tech Report CMU-CS-96-123.

[20] I. Koutis, "*Combinatorial and algebraic tools for optimal multilevel algorithms*", PhD thesis, Carnegie Mellon University, Pittsburgh, May 2007. CMU CS Tech Report CMU-CS-07-131, 2007.

[21] *I. Koutis and G. L. Miller, "Graph partitioning into isolated, high conductance clusters: theory, computation and applications to preconditioning*", in Proceedings of the twentieth annual symposium on Parallelism in algorithms and architectures, SPAA '08, pages 137–145, New York, NY, USA, 2008. ACM.

[22] D. A. Spielman and S. I. Daitch, "*Faster approximate lossy generalized flow via interior point algorithms*", In Proceedings of the 40th Annual ACM Symposium on Theory of Computing, May 2008.

[23] http://pages.cs.wisc.edu/~amos/412/lecture-notes/lecture17.pdf

[24] http://en.wikipedia.org/wiki/Least_squares

[25] http://fourier.eng.hmc.edu/e176/lectures/NM/node14.html

[26] http://en.wikipedia.org/wiki/Placement_(EDA)

[27] http://en.wikipedia.org/wiki/Conjugate_gradient_method

[28] http://en.wikipedia.org/wiki/Preconditioner

[29] http://en.wikipedia.org/wiki/Quadratic_programming

[30] http://en.wikipedia.org/wiki/Positive-definite_matrix

[31] http://en.wikipedia.org/wiki/Biconjugate_gradient_method

[32] Igor L. Markov, Jin Hu, Myung-Chul Kim*, "Progress and Challenges in VLSI Placement Research", University of Michigan, Department of EECS.*

[33] Charles Alpert, Zhuo Li, Gi-Joon Nam, C. N. Sze, Natarajan Viswanathan, Samuel I. Ward, *"Placement: Hot or Not?", IBM Corporation, Austin, Texas, USA.*