ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Υλοποίηση και βελτιστοποίηση αλγορίθμου για χωροθέτηση ολοκληρωμένων κυκλωμάτων σε παράλληλο περιβάλλον

Μεταπτυχιακή Διατριβή

Ιωαννίδης Κ. Σταύρος

Επιβλέποντες Καθηγητές: Σταμούλης Γεώργιος
Καθηγητής

Ευμορφόπουλος Νέστωρ
Επίκουρος Καθηγητής

Τσομπανοπούλου Παναγιώτα
Επίκουρη Καθηγήτρια

Βόλος, Μάρτιος 2014

UNIVERSITY OF THESSALY
DEPARTMENT OF ELECTRICAL
AND COMPUTER ENGINEERING

Implementation and optimization of an integrated circuit placement algorithm in parallel environment

Master Thesis

Stavros K. Ioannidis

Supervising Professors: George Stamoulis
Professor

Nestor Evmorfopoulos
Assistant Professor

Panagiota Tsompanopoulou
Assistant Professor

Approved by the three-member inquiry committee at March 19, 2014

George Stamoulis Professor	Nestor Evmorfopoulos Assistant Professor	Panagiota Tsompanopoulou Assistant Professor

To my family and friends.

Acknowledgments

I would like to express my gratitude to my supervisor, Dr. George Stamoulis, whose expertise, understanding, and patience, added considerably to my graduate experience. I appreciate his vast knowledge and skill in many areas. Finally, I would like to thank the other members of my committee, Dr. Nestor Evmorfopoulos and Dr. Panagiota Tsompanopoulou for the assistance they provided at all levels of this thesis.

Stavros K. Ioannidis
Volos, 2014

Περίληψη

Η συνεχής αύξηση του πλήθους των στοιχείων σε ένα σύγχρονο κυκλωματικό σχέδιο, θέτει ένα δύσκολο έργο στα εργαλεία χωροθέτησης, τα οποία απαιτείται να βρουν νέους τρόπους να χειρίζονται εκατομμύρια στοιχεία μέσα σε λογικά χρονικά πλαίσια. Λογισμικά χωροθέτησης που βασίζονται σε προσέγγιση «ωμής βίας» δεν μπορούν να αντιμετωπίσουν την πολυπλοκότητα των σύγχρονων σχεδίων. Από την άλλη, λογισμικά χωροθέτησης βασισμένα σε «διαίρει και βασίλευε» μεθόδους δεν είναι δυνατό να επιτύχουν αξιόλογα αποτελέσματα, καθώς έχουν έλλειψη της καθολικής εικόνας του κυκλώματος. Στην παρούσα διατριβή αξιολογούμε την βασισμένη στον αλγόριθμο GORDIAN υλοποίησή μας, η οποία είναι δυνατόν να παράγει πολύ γρήγορη λύση, διατηρώντας παράλληλα την καθολική εικόνα του κυκλώματος.

Abstract

The continuous increase of the cell count in modern designs, poses a challenging task to the placers that need to find efficient ways to handle millions of cells in reasonable time frames. Placers based on a brute force approach, cannot handle the complexity of modern circuit designs. On the other hand, divide-and-conquer methods cannot achieve remarkable results as they lack the global picture of the circuit. In this thesis we evaluate out GORDIAN based implementation, which can produce very fast placement solutions while maintaining the global scope of the design.

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
Bi-CG	**Bi-C**onjugate **G**radients
QP	**Q**uadratic **P**rogramming

1. Introduction

1.1. Thesis description

In this thesis we evaluate our implementation of GORDIAN [1], a method for Global Placement of standard-cell based circuit designs. Various algorithmic and parallel optimizations are applied in order to reduce the total runtime and memory requirements and improve the solution quality. Experimental results are presented, comparing GODRIAN to other state-of-the-art academic placers.

The vast execution speed and the limited memory footprint are GORDIAN's main advantages. GORDIAN runs faster than any other proven placer while still producing acceptable results. Million-sized designs can be placed in few minutes time.

The rest of this thesis is organized as follows: Section 2 provides an overview of the placement problem. Section 3 introduces the GORDIAN algorithm. Section 4 describes the GORDIAN implementation. Experimental results and conclusions are presented in Section 5.

1.2. Related work

A great variety of placers is used by modern placement industry and academic. Academic placers similar to GORDIAN that can produce solutions significantly fast, are *FastPlace* [3] and *SimPL* [2]. Like GORDIAN, they use the circuit's connectivity information to formulate and solve a mathematical minimization problem and apply various runtime and solution quality optimizations.

2. Introduction to Placement

2.1. Placement problem formulation

Placement is a procedure 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 non-manufacturable 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 [23]. 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.
- *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 (Picture 1). In many cases, parts of, or the entire, physical design flow are iterated a number of times until design closure is achieved.

2.3. Stages of Placement

The placement procedure is usually separated in Global placement and Detailed placement. An extra step of Legalization may be applied after Global placement or as a part of Detailed placement.

Global placement

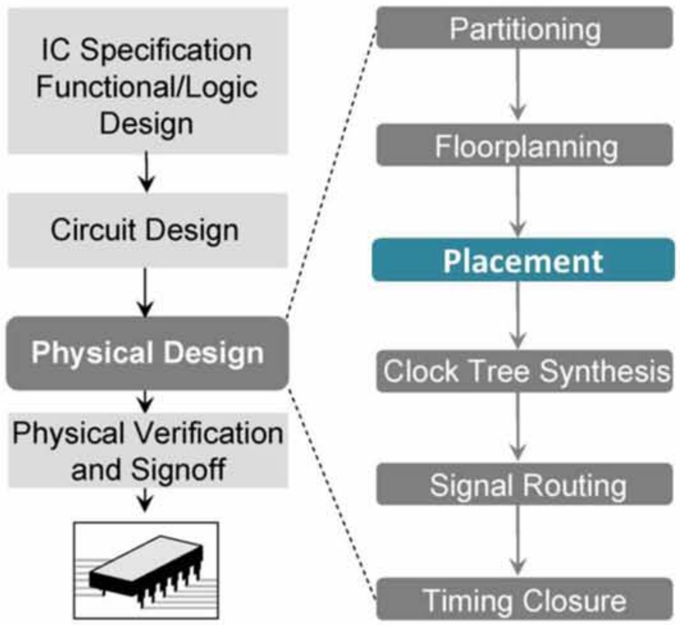
Global placement makes an initial placement of an un-placed netlist. The goal is to generate a near-optimal placement of the whole chip.

Legalization

After Global placement all circuit components are distributed over the chip area but their placement may not be legal. A Legalizer is responsible for eliminating any overlaps between the circuit components, enforcing all components to fit inside the defined core area and thus ensuring that the generated solution is feasible. Other technological constraints are also met during the Legalization stage.

Detailed placement

Fine adjustments to the positions of the circuit components are performed during Detailed placement, towards the improvement if the overall solution



Picture 1: Placement within the EDA design flow

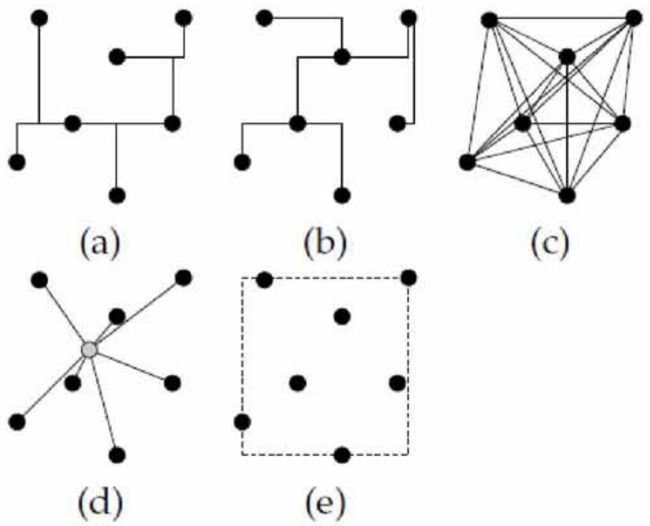
2.4. Complexity of the Placement problem

Although the legality of a given instance of the placement problem can be verified in polynomial time, the decision of the existence of such fusible instance is an NP-complete problem. Since we are interested in finding the optimal solution of the aforementioned NP-complete problem, the complexity of the placement problem can be classified as NP-hard. Thus, the placement problem is addressed by using heuristic methods and approximation algorithms.

2.5. Evaluation of a Placement solution

The primary component under evaluation is the solution's total wire length. Because the placement stage precedes the routing stage, any real routing information is not yet available during placement. Thus, various net models are used in order to estimate the wire length of a net and whereby the total wire length of the chip. Such net models are:

- *Steiner tree* (Picture 2a)
- *Minimum spanning tree* (Picture 2a, Picture 2b)
- *Clique* (Picture 2c)
- *Star* (Picture 2d)
- *Bounding box* (Picture 2e)



Picture 2: Common net models

2.6. State-of-the-art Placement approaches and algorithms

Simulated annealing

Simulated annealing is a general scheme that can be applied to a wide variety of optimization problems. Starting with any feasible solution, simulated annealing algorithms apply iteratively local changes to the solution. The changing steps are chosen randomly and steps that make the solution worse are allowed, so it is possible to leave local optima. In early steps, bigger worsening changes are allowed, while in later steps only small worsening changes are. Although simulated annealing is too slow for a global optimization of a placement, it is still in use to solve sub problems or for local optimization. A placement tool that is based mainly on simulated annealing is TimberWolf [14].

Minimum cut

Top-down recursive partitioning is used in many placement algorithms. The main idea consists of recursively dividing both the chip area and the set of circuits into subsets and to assign each circuit subset to a subarea of sufficient capacity. The step is repeated until the regions are small enough to run legalization. Algorithms that exploit the minimum cut technique are Capo [6] and FastPlace [3][4][5].

Analytical Placement

In Analytical Placement the wire length is minimized ignoring the overlaps among the circuit components. Then, the placement is modified in order to reduce those overlaps.

Modern algorithms usually exploit more than one of the above approaches. Dragon placement tool [8][10] for example, treats whole designs using a minimum cut approach while handles the smaller-scale blocks using an analytical approach.

3. Introduction to GORDIAN

GORDIAN is one of the most successful Global placement algorithms. With GORDIAN, the placement problem is formulated as a sequence of quadratic programming problems derived from the entire connectivity information of the circuit. An increasing number of constraints are imposed, reflecting the results of successively refined partitioning.

GORDIAN uses a combination of the Analytical and Minimum cut approaches. A global optimization step is initially applied, ignoring any library constraints (cell overlaps, cell outside core area, etc.) while minimizing the total wire length. A top-down partitioning strategy follows, while ignoring the capacity constraints of the partitions. 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.

3.1. GORDIAN Specifications

Standard-cell Placement

Our GORDIAN based implementation focuses on standard-cell placement. Standard-cell placement is a row oriented placement of standard-cells on a rectangular core area. A standard cell represents the physical space occupied by a logical gate. The type of the gate itself is often irrelevant during the placement and routing procedure. In standard-cell placement all cells must be of the same height. The width of the cells may vary.

Fixed I/O pins/pads

The positions of the I/O pins/pads must be fixed in order to be fed as input to GORDIAN. On the other hand, the definition of the dimensions of the rectangular core area is not essential. In that case the rectangular core area will be defined by the rectangular box formed by the fixed pins/pads.

3.2. GORDIAN Description

Global optimization step

Before the global optimization step is applied, the input circuit is converted to an undirected weighted graph. Each net is treated as a *k*-clique and a weight of $\frac{2}{k}$ is assigned to each edge involved in the clique, as shown on Picture 3 and Picture 4. The movable cells and fixed pins constitute the nodes of the graph.

Using the weight information the following matrices are formed:

- *Adjacency* matrix A_{nxn}, where $A_{i,j} = Wc_{i,j}$
- *Pin Connection* matrix P_{nxm}, where $P_{i,j} = Wp_{i,j}$
- *Degree* matrix D_{nxn}, where $D_{i,j} = \begin{cases} \sum_{j=0}^{n} A_{i,j} + \sum_{j=0}^{m} P_{i,j}, & i = j \\ 0, & i \neq j \end{cases}$
- *Laplacian* matrix C_{nxn}, where $C_{i,j} = D_{i,j} - A_{i,j}$
- *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,
- $Wc_{i,j}$ is the weight of the edge connecting cell i and cell j,
- $Wp_{i,j}$ 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.

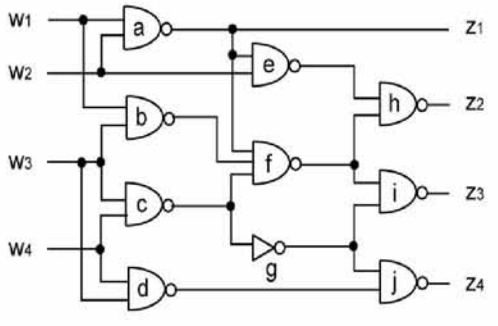
The objective function of the global optimization step is now formulated in the following quadratic programming problem:

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

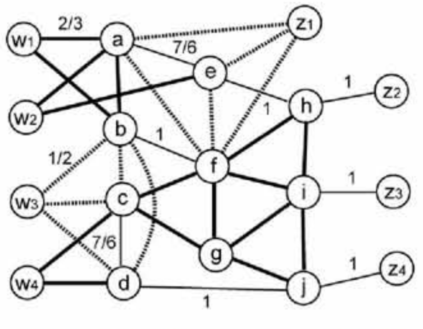
and:

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

where d_x and d_y are the *Fixed Pin* vectors and C is the *Laplacian* matrix.



Picture 3: Input circuit



Picture 4: Undirected weighted graph

For minimizing the above convex functions the following linear systems must be solved:

$$Cx = -d_x$$

and

$$Cy = -d_y$$

The resultant x and y vectors contain the x- and y-positions of all the cells. At this point all the cells will fit inside the rectangle core area, but they be highly overlapping as shown on Picture 5.

Top-down partitioning

During the first step of the top-down partitioning procedure the rectangular core area is divided into two regions, each containing a subset of the movable cells. On the following steps every region is recursively divided in two sub regions, creating a total of $q \leq 2^i$ partitions, where i is the number of partitioning step. The centers u of those regions impose the following constraints on the quadratic programming formula:

$$A^i x = u$$

and

$$A^i y = u$$

such that the weighted mean value of the cells assigned to a region corresponds to the center of that region.

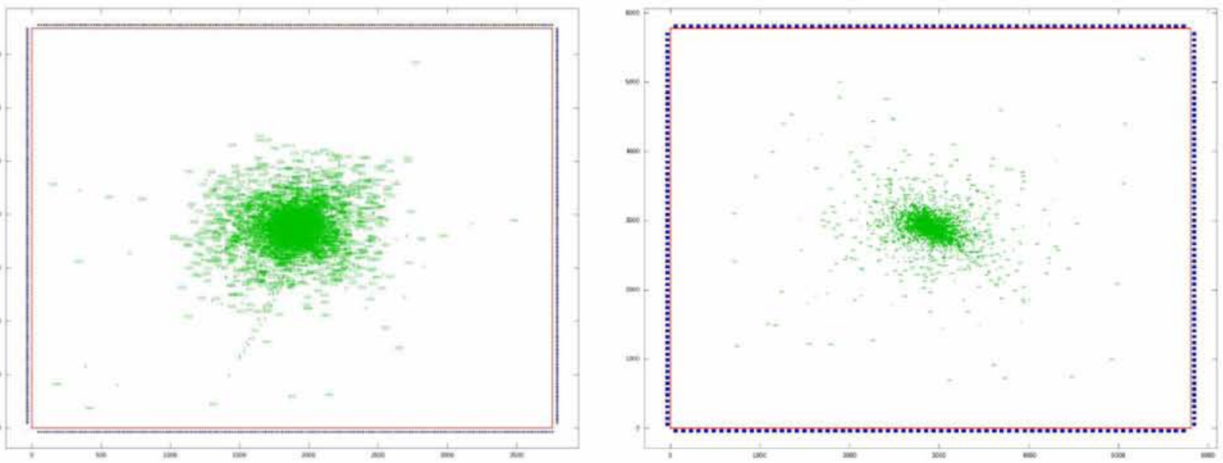
The contents of the constraints matrix A_{qxm} are $A_{i,j} = \begin{cases} 1, & cell\ j\ belongs\ to\ partition\ i \\ 0, & otherwise \end{cases}$

Combining the objective functions with the above constraints, the following linearly constrained quadratic programming problems are obtained:

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

and

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



Picture 5: Distribution of cells after Global optimization step.

For minimizing the above convex functions the following linear systems must be solved:

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

and

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

where u_x and u_y are the x and y coordinates of the partitions respectively and the resultant x and y vectors contain the x- and y-positions of all the cells.

Final placement

The goal of the GORDIAN legalization is to meet the following technological constraints:

- all cells aligned to rows
- no overlapping cells
- all cells fit inside the core area

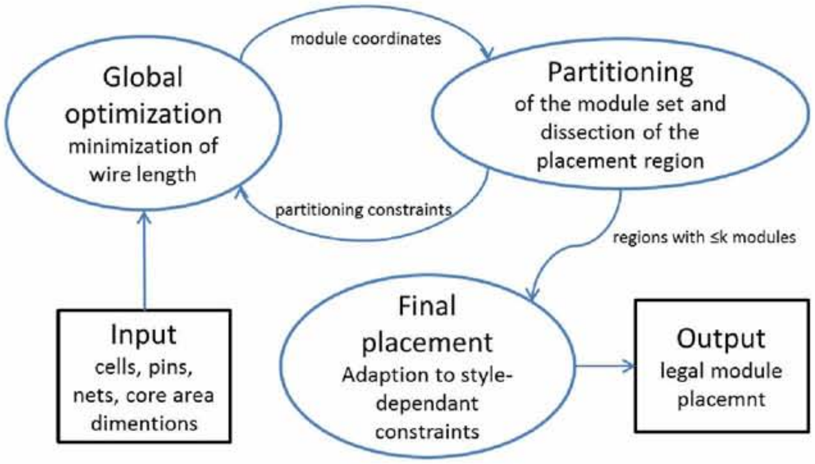
while affecting the calculated positions of the modules as little as possible.

To achieve that, the sorting of the cells based on their position's y-coordinate is required. The rectangular core area is then divided into r subsets using $r-1$ horizontal cuts such that

$$y_{\mu_1} \leq \cdots \leq y_{\mu_i} \leq \cdots \leq y_{\mu_r}$$

where the module μ_i belongs to the ith row. The sequence of the modules within the rows is determined by their x-coordinate. The outboard modules of a row based on their y-coordinate can be moved to close by rows, if the sum of the modules' widths exceeds the maximum row length.

Due to the simplicity of the legalization algorithm, the final result is not absolutely refined. Our experimental measurements show an average increase of 25% in the wire length after legalization. More sophisticated legalizers can drop that percentage down to 10%. Furthermore, the addition of a detailed placement stage can reduce to total wire length by a factor of 10-15%.


module coordinates
Global optimization minimization of wire length
Partitioning of the module set and dissection of the placement region
partitioning constraints
regions with ≤k modules
Input cells, pins, nets, core area dimentions
Final placement Adaption to style-dependant constraints
Output legal module placemnt

Picture 6: Data flow of the GORDIAN placement procedure

4. Implementation of GORDIAN

4.1. Algorithm implementation

Our motivation for implementing GODRIAN derives from the fact the GORDIAN is a fast placement algorithm supported by a strong mathematical model and has the ability to place modern sized circuits within reasonable timeframes. Moreover, GORDIAN is highly configurable offering room for interesting optimizations.

Our goal was to develop and optimize the GORDIAN method so that:

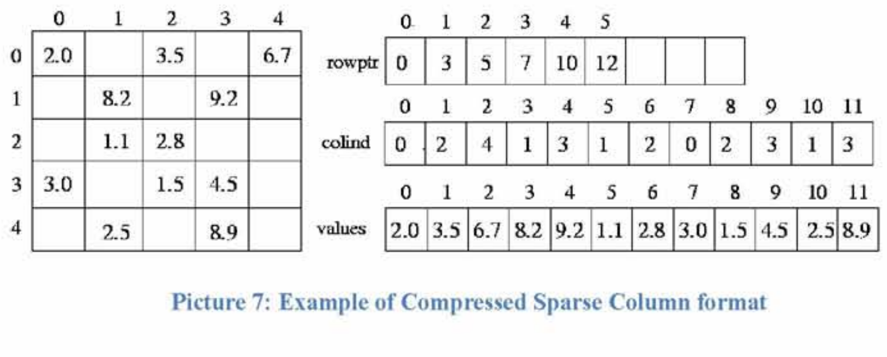
1. Modern sized circuits can be placed in a few minutes time, without the need of a massive computing infrastructure.
2. The quality of placement solution would be comparable to that of a state-of-the-art placer.

Matrix compression

Since all matrices used by GORDIAN (*Adjacency*, *Laplacian*, etc.) derive from sparse graphs, the matrices themselves will be sparse. In order to accommodate the matrices created by a modern sized circuit inside the RAM memory of a single computer, the use of sparse data structures and compression techniques are substantial. On the other hand all vectors used by GORDIAN are dense, thus the use of sparse vector structures is redundant.

In our implementation we exploit the csparse library [27], a C library for managing sparse matrices. We use the Compressed Sparse Column format for our matrices. The amount of occupied memory is limited to $2N + M$ for a NxN sized sparse matrix with M non-zero elements, instead of N^2 required by a dense matrix.

The calculation and conservation of the *Adjacency*, *Pin Connections* and *Degree* matrices is dispensable, since the elements of the *Laplacian* matrix that derive from the above matrices can be calculated in a single step.



	0	1	2	3	4
0	2.0		3.5		6.7
1		8.2		9.2	
2		1.1	2.8		
3	3.0		1.5	4.5	
4		2.5		8.9	

	0	1	2	3	4	5			
rowptr	0	3	5	7	10	12			

	0	1	2	3	4	5	6	7	8	9	10	11
colind	0	2	4	1	3	1	2	0	2	3	1	3

	0	1	2	3	4	5	6	7	8	9	10	11
values	2.0	3.5	6.7	8.2	9.2	1.1	2.8	3.0	1.5	4.5	2.5	8.9

Picture 7: Example of Compressed Sparse Column format

Linear system solvers

Our implementation uses iterative methods for solving the generated linear systems, over direct methods, for reasons of time complexity. Direct solvers usually have a time complexity of $O(N^2)$, compared to iterative solvers with nearly linear to the size of the system time complexity.

More specifically the methods used are:

- The *Conjugate Gradient (CG)* method [32] for solving the linear system generated during the global optimization step.
- The *Bi-Conjugate Gradient (Bi-CG)* method [32] for solving the linear systems generated during the top-down partitioning procedure.

Unlike *Bi-CG*, the *CG* method can only handle *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 A.

Input and Output format

Out GORDIAN implementation supports the *Bookshelf GSRC* input format [24]. However, the conversion to a more common or industrial input format - such as LEF/DEF - is possible using various conversion software available online [26].

The *Bookshelf GSRC* format is designed to be simple and easily human readable, as well as conveniently parsable. For this purpose, details such as inter-node geometries or hierarchies are not included. If such level of detail is required, existing industry standard file formats (LEF/DEF, TW etc.) can be used.

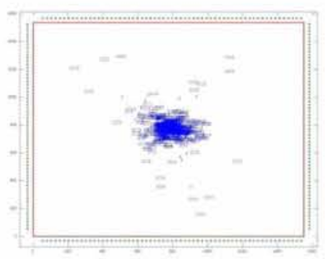
All placement information is distributed over several component files. To assemble multiple component files into problem instances, an *.aux* a file is used that typically includes one line that consists of the paths of the individual component files. The component files may be of the following format:

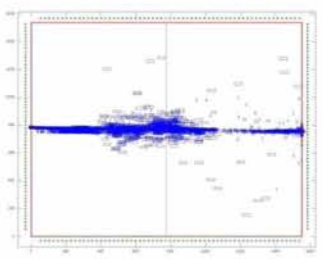
- A *.nodes* file consisting of a list of points. These points typically represent the lower-left corner or the center of a cell.
- A *.nets* file consisting of a list of nets.
- A *.wts* file consisting of weighting information about the nets.
- A *.scl* file containing information regarding the standard cell layout.
- A *.pl* file containing information about the position of cells and pins/pads.

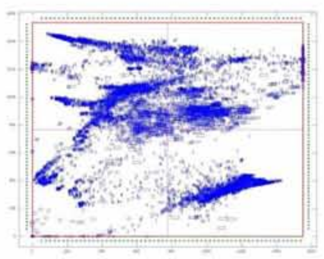
The format also supports additional file types used by later stages of the placement process and the routing process (routing info etc.).

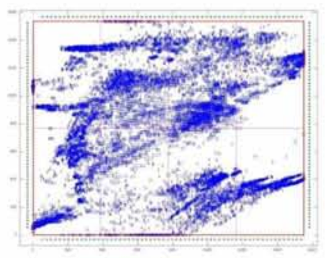
Solution plotting

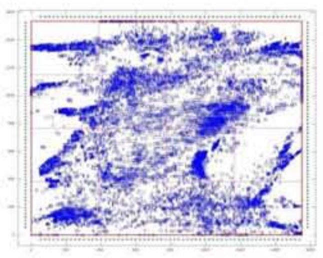
Using the *gnuplot* graphical utility for Linux [28][29], we incorporated plotting abilities in out implementation. The output of the plotter is demonstrated in Picture 8.

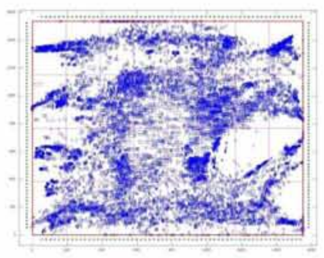

Global optimization step

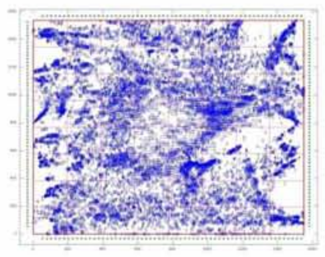

Partitioning iteration 1

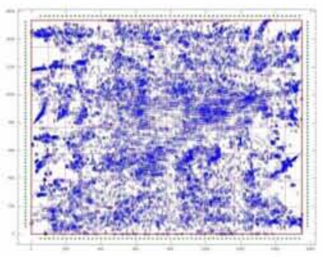

Partitioning iteration 2

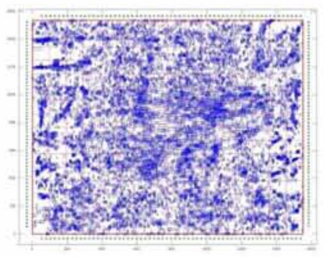

Partitioning iteration 3

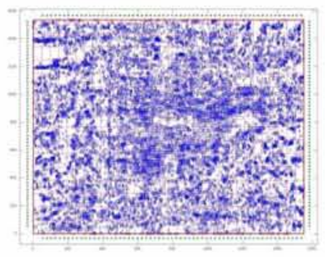

Partitioning iteration 4

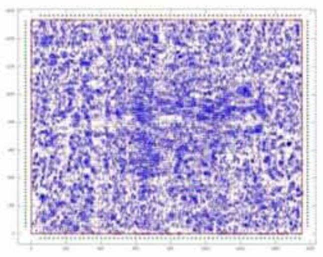

Partitioning iteration 5

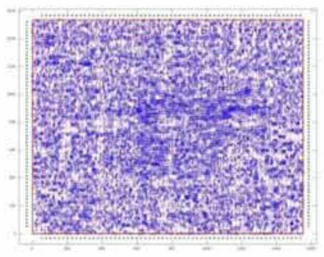

Partitioning iteration 6

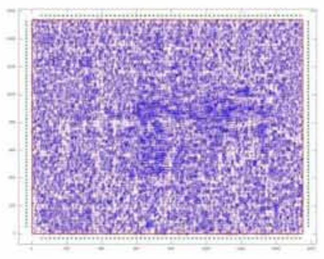

Partitioning iteration 7

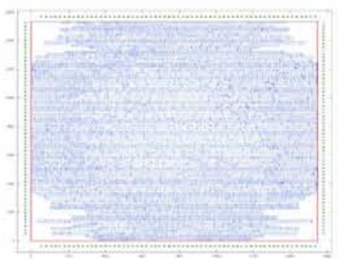

Partitioning iteration 8


Partitioning iteration 9


Partitioning iteration 10


Partitioning iteration 11

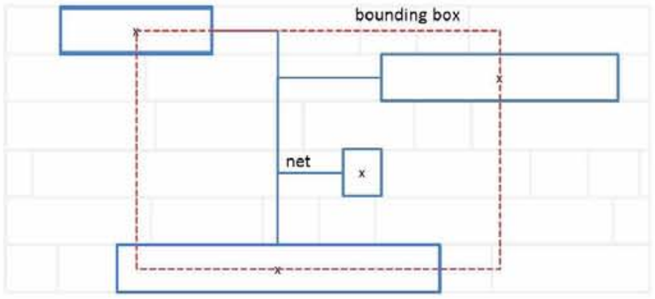

Partitioning iteration 12


Final placement

Picture 8: Distribution of cells after every step of GORDIAN
Benchmark: ibm01

Wire length model

In order to evaluate the placement solution of out implementation, we use the half perimeter wire length model (HPWL). The HPWL is defined as the sum of the $\frac{1}{2}$ perimeter of the bounding box of every net, where bounding box is the minimum rectangular area that contains all modules connected by the net (Picture 9).

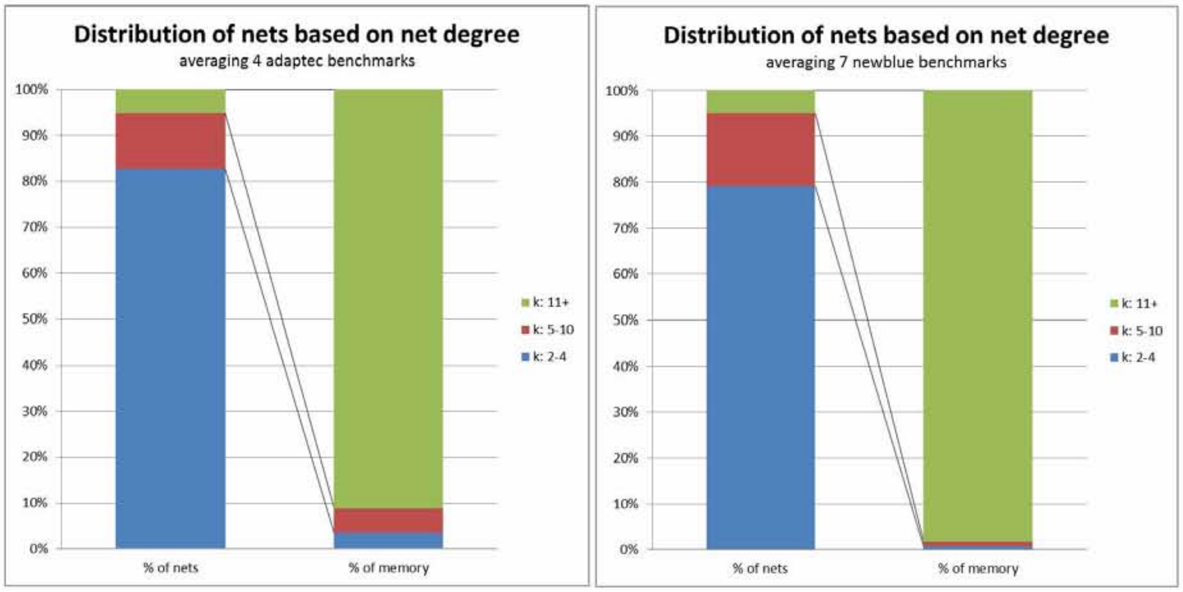


Picture 9: Net's bounding box

4.2. Algorithmic optimizations

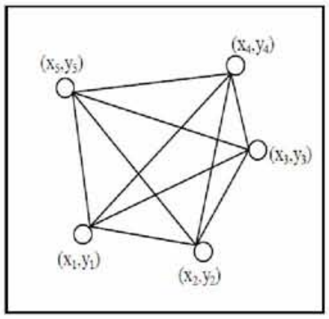
Star nets model

As mentioned before, nets are treated as k-cliques so that a weight is assigned to every edge of the clique. However, the number of edges in a k-clique are $\frac{k(k-1)}{2}$. For large nets this leads to a significant computational and memory cost even if the total count of those nets in the circuit is low. The percentage of large- and medium-degree nets of the tested benchmarks and the corresponding memory they occupy is shown in Picture 10.

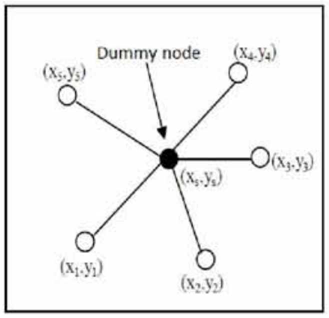


Picture 10: Distribution of nets based on net degree

To resolve the above issue, we treat all large- and medium-degree nets as star nets, by adding a dummy-node in the center of the net. The edge count of the k-cluque is redused to k. We achieve a minimum of 80% reduction to the size of required memory and a 2.0x speedup. Concurrently, there is no significant change of the solution's quality.

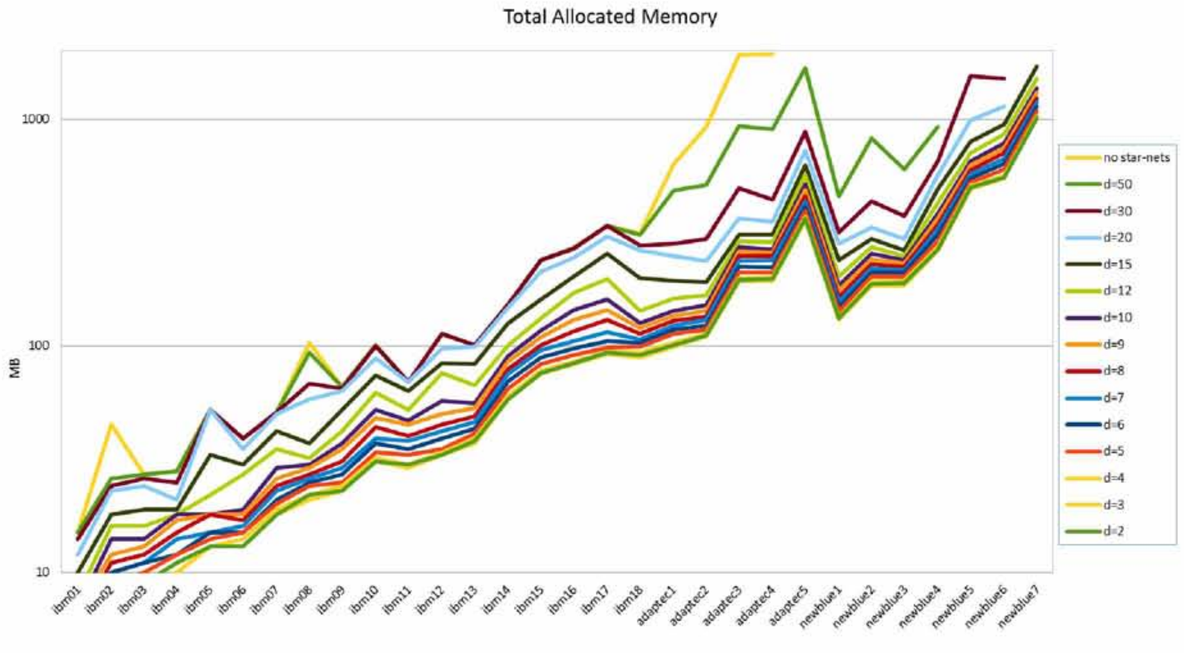


Picture 11: k-clique model with $\frac{k(k-1)}{2}$ edges



Picture 12: star model with k edges

The exact selection of the d constant, where all k-degree nets with $k > d$ will be treated according to the star model, affects the total allocated memory. Picture 13 shows the total required memory for all benchmarks for various values of d. Consequently, lower values for d are preferable.



Picture 13: Memory needs based on selected d

Cell sorting

During the top-down partitioning process the repetitive sorting of cells based on their x- or y-coordinate is required. Our implementation exploits the *quicksort* recursive algorithm of $O(n \log n)$ time complexity, opposed to the *insertion* or *selection sorting* algorithms of quadratic time complexity.

Norm selection

A norm function is required in *CG* and *Bi-CG* methods, in order to determine the sufficiency of the convergence. The most adequate norm functions are the *Euclidian* and the *Infinite norm* which correspond to the following formulas:

$$\|x\|_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2}$$

and

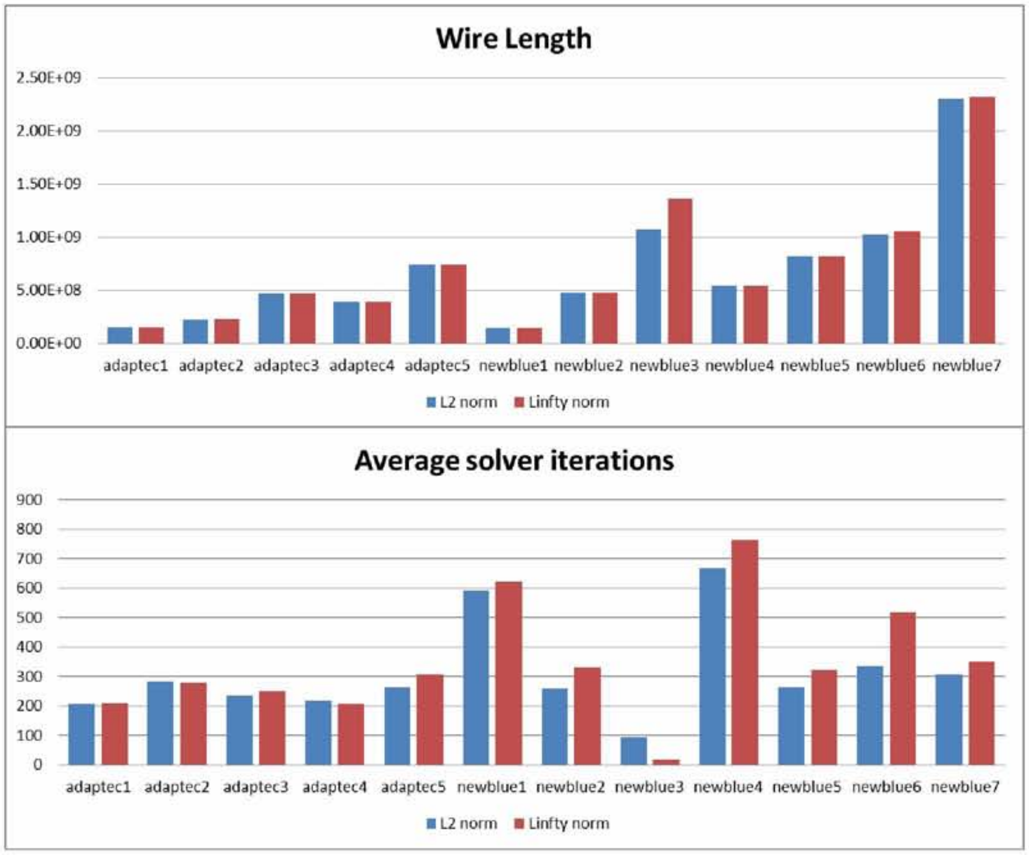
$$\|x\|_\infty = \max_i |x_i|$$

The sufficient convergence criterion is:

$$\frac{\|r\|}{\|b\|} \leq t$$

where b is the initial solution vector (typicaly the linear system's right-hand side vector), r is the current iteration's solution vector and t is a predefined convergence threshold.

The *Euclidean norm* results a faster convergence and less HPWL in most cases as illustrated Picture 14.



Picture 14: Norm comparison

Improved partitioning

During a partitioning step each region r is divided into two subregions r_1 and r_2. The corresponding module set m is also divided into two subsets m_1 and m_2. The sums of the module areas of both subsets determine the bisection of the rectangular area of region r such that

$$\frac{\sum_{i\in m_1} area(i)}{\sum_{i\in m_2} area(i)} = a$$

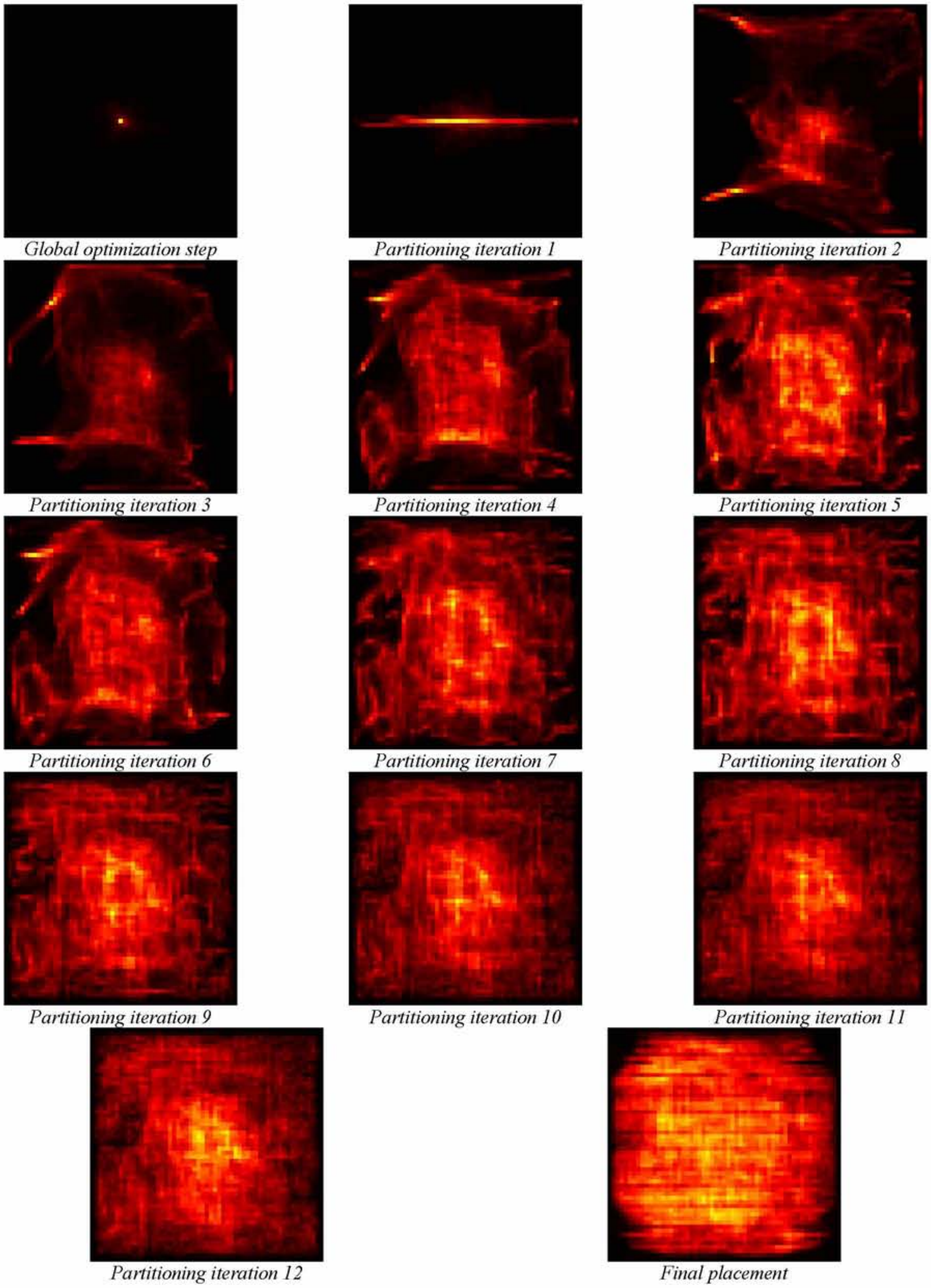
where a is the desired area ratio. The most obvious way of partitioning is to predefine $a = 0.5$ and to alternate the direction of the cut on each level. This leads to regions with approximately the same area and aspect ratio.

An improved partitioning decision should be based on the number of nets crossing the new cut line. This number can be minimized by variation of the cut position and thus the desired area ratio a. However, extreme values of a such as $a < 0.4$ or $a > 0.6$ should be avoided, as they can result in uneven partitions and increased wasted area. Experimental results show an average of 1% improvement in the final solution, when the optimal cut line is used. Yet, due to the dissimilar resulting partitions, this technique shows slight solution deterioration on some benchmarks.

In order to decrease the total number of linear systems requiring solving, multiple partitionings can be applied before the next constrained quadratic programming problem must be solved. Since all modules are assigned to each partition based solely on their position, the simultaneous dissection of the placement area in more than two partitions harms the solution's quality by increasing the HPWL. On the other hand, by reducing the number of costly linear system solutions, the total execution time is significantly reduced. Experimental measurements indicate a 7% increase in HPWL and 33% faster execution when 2 partitionings are applied before solving and a 27% increase in HPWL and 64% faster execution when 4 partitionings are applied before solving.

Congestion plotting

Using the *CongestionMaps* plotter tool [26], we demonstrate the chip congestion after the global optimization step as well as the congestion's spreading throughout the partitioning process in Picture 15. Notice the gradual elimination of the highly-congested spots (yellow) and the proliferation of the low-congested regions (red).



Picture 15: Core area congestion after every step of GORDIAN
Benchmark: ibm01

Legalization

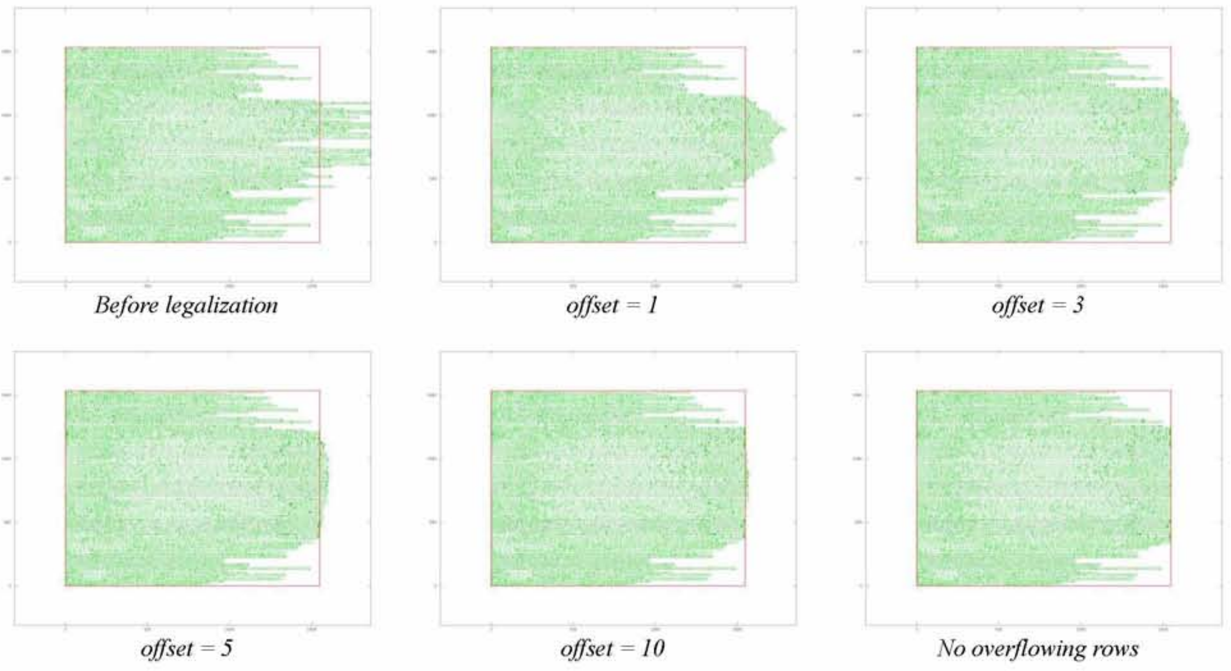
After aligning all modules to rows, our implementation uses a recursive sweeping technique in order to relieve the rows of their overflowing modules (Picture 16). Modules are exchanged between *offset* adjacent rows. If no exchanges are possible during a sweep of all rows, the *offset* parameter is gradually increased so overflowing cells are able move towards farther rows. The procedure is terminated when all overflows are eliminated.

```
procedure Legalization (Cells, Rows)
    offset:= 1;
    do
        changes:= false;
        for every $R_i \in Rows$
            if $R_i$ overflows
                if first cell in $R_i$ fits in $R_{i-offset}$ then
                    move cell to $R_{i-offset}$
                    changes:= true;
                    offset:= 1;
                endif
                if last cell in $R_i$ fits in $R_{i+offset}$ then
                move cell to $R_{i+offset}$
                    changes:= true;
                    offset:= 1;
                endif
            endif
        endfor

        if changes = false then
            offset:= offset+1;
        endif

    until $\nexists R_i \in Rows \mid R_i$ overflows
endporcedure
```

Picture 16: Legalization procedure

Since the described legalization process is highly affected by the structure of the overflowing modules, an inconsistent time complexity is noted when applying to different benchmark circuits. Our experimental measurements indicate that several tenths or hundreds of row sweeps are generally required.

Picture 17 illustrates the gradual smoothing of the overflowing rows as the offset parameter increases and overflowing modules are allowed to migrate to farther rows.



Picture 17: Gradual elimination of overflowing modules

Preconditioning

The iterative solver's convergence rate can be improved by the addition of enhanced preconditioners. The most obvious option is the *Jacobean* preconditioner, which leads the *CG* and *Bi-CG* methods to converge in a few hundreds iterations. More efficient preconditioners such as Vayda's graph based preconditioners [17] can result a significantly faster convergence. Our implementations exploits the CMG [18] preconditioner for solving SPD linear systems. Due to the posed constraint, the above preconditioner can only be applied to the SPD linear system formed during the global optimization step.

Experimental results show an average speedup of 2%. However, due to the preconditioner's vast memory requirements, larger circuit benchmarks were impossible to place.

Alternative solution methods

An alternative method to solve the equality constrained quadratic programming problem created during the top-down partitioning process is presented in [15]. This approach is based implicitly on a reduced linear system and generates iterates in the null space of the constraints. More specifically, this method eliminates the constraints and solves the following reduced problem:

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

where A is the constraints matrix, x_A^* is calculated using:

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

Z is a matrix spanning the null space of A and x_z^* is calculated using:

$$C_{zz} x_z = -c_z,$$

where

$$C_{zz} = Z^T C Z,$$

and

$$c_z = Z^T (C A^T x_A^* + c).$$

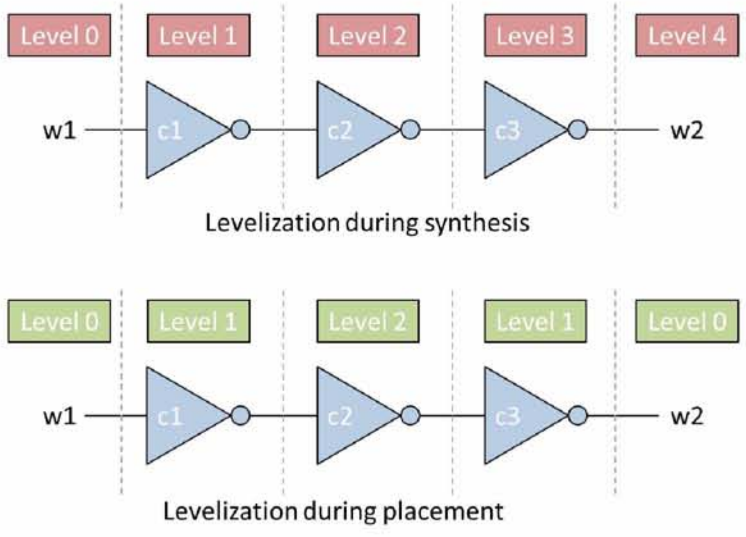
Unfortunately, the C_{zz} matrix formulated during the above process is dense, resulting high execution time and prohibitively great memory requirements.

Levelization

In order to achieve the desired cell distribution earlier in the partitioning process, we apply a levelization algorithm and modify the weight assigning process accordingly. Smaller weights are assigned to nets that connect cells of lower level, i.e. cells closer to the I/O pins/pads. This results in an increased degree of freedom for those cells, allowing them to easier move away from the core area center and towards the I/O pins/pads.

Notice that our levelization process determines a cell's level based solely on the distance from either an input or an output pin. It differs from the levelization applied during the EDA Synthesis process, where the logical level of the cells is calculated. The difference is demonstrated in Picture 18.

The result of the levelization process on the placement solution is an average increase of HPWL of 4%. However, the target placement can be achieved using 1-2 less iterations of the partitioning process.



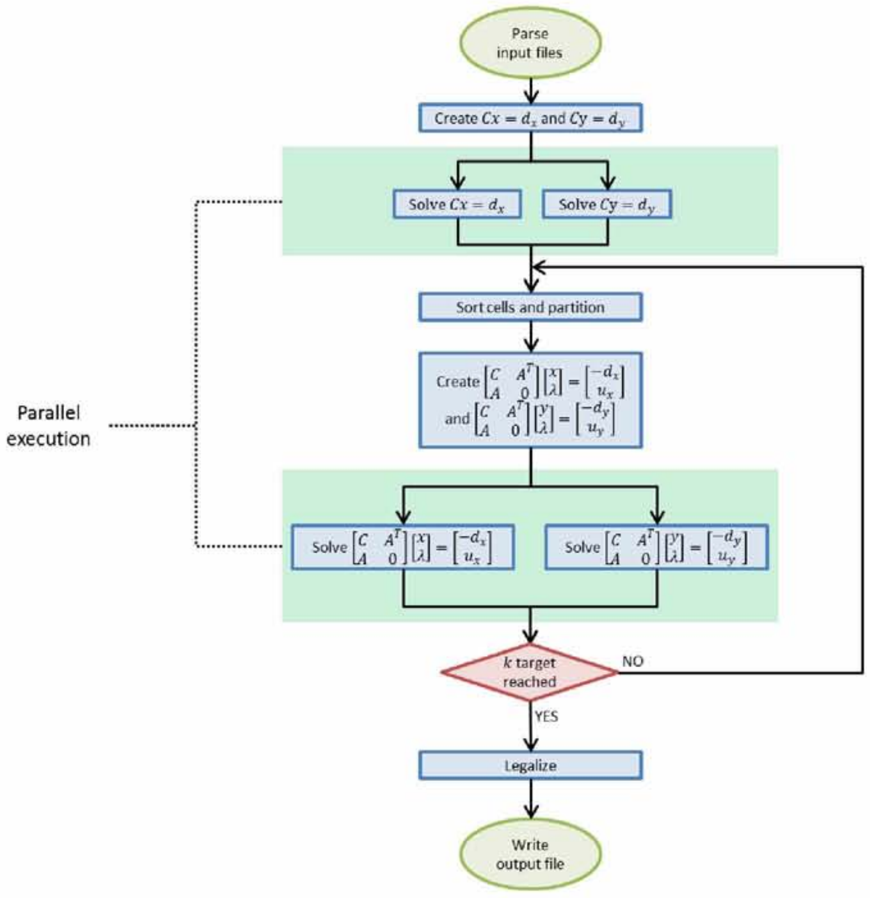
Picture 18: Difference of Synthesis and Placement levelization process

4.3. Parallelization

Out experimental measurements show that the vast majority of the execution time is devoted to the top-down partitioning process. Consequently, any parallelization efforts focus on that process. However, since the iterations of the top-down partitioning process are data dependent, any form of parallelization between consecutive iterations is unlikely.

Notice that the purpose of the partitioning process is to iteratively refine the placement solution and does not reduce the problem size, or offer any type of independent data and/or calculation sets that can be treated in parallel. The sole requirement of the above process is the solution of two solid linear systems, for the x- and y-coordinate respectively. Therefore, GORDIAN's capacity to exploit potential parallelism is limited to parallelizing a single iteration of the top-down partitioning process.

Our implementation uses *OpenMP* [30] in order to concurrently solve the two independent linear systems formulated during every iteration of the top-down partitioning process as well as the unconstrained linear systems of the global optimization step (Picture 19). A 1.6x speedup is obtained. Other techniques involving parallelization of the *CG* and *Bi-CG* methods using *OpenMP* [30] and *CUDA* [31] have also been tested, but rejected due to parallelization time overhead.


Parse input files
Create $Cx = d_x$ and $Cy = d_y$
Solve $Cx = d_x$
Solve $Cy = d_y$
Sort cells and partition
Create $\begin{bmatrix} C & A^T \\ A & 0 \end{bmatrix}\begin{bmatrix} x \\ \lambda \end{bmatrix} = \begin{bmatrix} -d_x \\ u_x \end{bmatrix}$ and $\begin{bmatrix} C & A^T \\ A & 0 \end{bmatrix}\begin{bmatrix} y \\ \lambda \end{bmatrix} = \begin{bmatrix} -d_y \\ u_y \end{bmatrix}$
Solve $\begin{bmatrix} C & A^T \\ A & 0 \end{bmatrix}\begin{bmatrix} x \\ \lambda \end{bmatrix} = \begin{bmatrix} -d_x \\ u_x \end{bmatrix}$
Solve $\begin{bmatrix} C & A^T \\ A & 0 \end{bmatrix}\begin{bmatrix} y \\ \lambda \end{bmatrix} = \begin{bmatrix} -d_y \\ u_y \end{bmatrix}$
k target reached
NO
YES
Legalize
Write output file
Parallel execution

Picture 19: GORDIAN flow chart with parallel regions

5. Experimental results

5.1. Benchmark suites

Our GORDIAN implementation was tested on the *ISPD-04 Placement Benchmarks* [20], the *ISPD-05 Placement Benchmarks* [21] and the *ISPD-06 Placement Benchmarks* [22]. These benchmarks have been derived from industrial ASIC designs with circuit sizes ranging from 12K to 2M. They consist of standard cells and pre-placed I/O pins.

Benchmark	# Cells	# I/O Pins	# Nets	# Rows
ibm01	12506	246	14111	96
ibm02	19342	259	19584	109
ibm03	22853	283	27401	121
ibm04	27220	287	31970	136
ibm05	28146	1201	28446	139
ibm06	32332	166	34826	126
ibm07	45639	287	48117	166
ibm08	51023	286	50513	170
ibm09	53110	285	60902	183
ibm10	68685	744	75196	234
ibm11	70152	406	81454	208
ibm12	70439	637	77240	242
ibm13	83709	490	99666	224
ibm14	147088	517	152772	305
ibm15	161187	383	186608	303
ibm16	182980	504	190048	347
ibm17	184752	743	189581	379
ibm18	210341	272	201920	361

Table 1: ISPD-04 Placement Benchmarks

Benchmark	# Cells	# I/O Pins	# Nets	# Rows
adaptec1	210904	543	221142	928
adaptec2	254457	566	266009	1221
adaptec3	450927	723	466758	1948
adaptec4	494716	1329	515951	1948
bigblue1	277604	560	284479	928
bigblue2	534782	23084	577235	1572
bigblue3	1095519	1293	1123170	2322
bigblue4	2169183	8170	2229886	2698

Table 2: ISPD-05 Placement Benchmarks

5.2. Experimental measurements

Solution quality

In Table 5 we present the HPWL and Runtime measurements of our GORDIAN implementation, on the *ISPD-04 Placement Benchmarks*. We compare our results with state-of-the-art academic placers *FastPlace 2.0* [3][4], *Capo 9.1* [6] and *FengShui 5.0* [11]. All the placers were run in their default mode and all experiments were run on a 2.5 GHz Intel i5 core machine with 8 GB RAM.

The default mode for GORDIAN is

- $k = 4$: Partitioning proccess stops when all partitions contain 4 cells at most.
- $d > 4$: All nets of degree greater than 4, are treated as star nets.
- $a = 0.5$: Improved partitioning via variation of the cut line is disabled.

We present the results of GORDIAN before and after legalization. That being because GORDIAN's legalization technique produces a rough final placement, increasing the HPWL by approximately 25%. Moreover, GODRIAN results are not refined by a detailed placer. Our experience shows that state-of-the-art legalizers cause an average increase of 10% to the HPWL. This loss is earned back when a detailed placer is applied. Thus, it is logical to assume that a combination of GORDIAN global placer with a state-of-the-art legalizer and detailed placer can produce results similar to that of GORDIAN before the legalization step.

In Table 6Table 5 we present the HPWL and Runtime measurements of out GORDIAN implementation, on the *ISPD-05 Placement Benchmarks*. We compare our results with placers reported during the ISPD-05 contest *Aplace 2.0* [13], *Capo 10.5* [6], *FastPlace 3.0* [3][5], *mPL6* [12] and *SimPL* [2].

The HPWL and Runtime improvement ratios are shown in Table 3 and Table 4 respectively. GORDIAN produces an unlegalized solution with an average of 65% more wire length, compared to the other placers but runs 5 to 257 times faster.

HPWL Improvement	FastPlace 2.0	FastPlace 3.0	Capo 9.1	Capo 10.5	Aplace 2.0	FengShui 5.0	mPL6	SimPL
GORDIAN Unlegal	0.73	0.47	0.82	0.54	0.49	0.76	0.47	0.46
GORDIAN Final	0.58	0.43	0.66	0.49	0.44	0.61	0.42	0.41

Table 3: GORDIAN HPWL improvement

Runtime Improvement	FastPlace 2.0	FastPlace 3.0	Capo 9.1	Capo 10.5	Aplace 2.0	FengShui 5.0	mPL6	SimPL
GORDIAN Unlegal	15.13	7.00	257.04	70.47	91.62	119.05	42.22	5.86
GORDIAN Final	12.80	6.44	210.37	64.72	83.91	97.61	38.49	5.34

Table 4: GORDIAN Runtime improvement

	HPWL (e6)					Runtime (sec)				
Benchmark	GORDIAN Unlegal	GORDIAN Final	FastPlace 2.0	Capo 9.1	FengShui 5.0	GORDIAN Unlegal	GORDIAN Final	FastPlace 2.0	Capo 9.1	FengShui 5.0
ibm01	2.68	3.63	2.45	2.57	2.47	0.6	0.8	10.0	219.0	142.0
ibm02	6.69	7.65	4.91	5.20	5.30	2.0	2.0	34.0	457.0	245.0
ibm03	8.74	10.35	7.32	8.78	8.49	1.7	3.2	27.0	735.0	284.0
ibm04	10.16	12.30	8.14	9.04	8.55	2.0	2.5	38.0	771.0	323.0
ibm05	18.29	19.36	10.24	10.24	9.83	2.2	2.2	34.0	684.0	372.0
ibm06	7.72	10.15	6.01	7.51	6.85	2.5	6.9	37.0	809.0	437.0
ibm07	15.96	19.57	10.99	12.20	11.54	4.1	4.0	105.0	1236.0	586.0
ibm08	14.62	18.29	12.38	13.99	12.88	6.6	6.8	121.0	1322.0	647.0
ibm09	15.76	22.26	13.79	15.31	13.79	5.4	7.5	94.0	1375.0	660.0
ibm10	32.68	40.39	31.65	37.35	35.13	10.5	10.5	162.0	2666.0	1085.0
ibm11	32.47	40.55	20.30	21.92	19.69	9.0	9.3	132.0	2172.0	891.0
ibm12	42.67	49.69	34.18	39.99	36.23	12.3	21.7	189.0	3413.0	1011.0
ibm13	29.59	41.42	25.21	29.24	24.71	11.1	11.2	170.0	4288.0	1189.0
ibm14	66.12	82.95	37.76	40.40	38.89	34.8	47.5	316.0	5091.0	2553.0
ibm15	82.70	102.87	52.56	59.39	50.98	44.7	49.2	415.0	6399.0	3171.0
ibm16	79.48	106.22	58.37	70.63	60.12	41.6	42.5	417.0	7211.0	3626.0
ibm17	166.30	190.19	69.89	75.48	69.19	51.8	55.6	624.0	6782.0	3935.0
ibm18	91.35	121.06	45.39	47.66	44.48	69.8	78.2	769.0	5163.0	3471.0

Table 5: Experimental measurements on ISPD-04 Placement Benchmarks

	HPWL (e6)							Runtime (sec)						
Benchmark	GORDIAN Unlegal	GORDIAN Final	Aplace 2.0	Capo 10.5	FastPlace 3.0	mPL6	SimPL	GORDIAN Unlegal	GORDIAN Final	Aplace 2.0	Capo 10.5	FastPlace 3.0	mPL6	SimPL
adaptec1	148.01	162.1	78.35	88.14	78.16	77.93	77.73	22.5	26.5	2101.2	1557	150	1101.6	136.2
adaptec2	227.33	203.5	95.70	100.25	93.56	92.04	90.36	34.6	39.1	3034.2	2163.6	219.6	1194.6	208.8
adaptec3	467.99	465.9	218.52	276.80	213.85	214.16	208.95	62.2	65.6	7171.8	4691.4	508.8	3535.2	422.4
adaptec4	388.73	413.4	209.28	231.30	198.17	193.89	187.40	53.1	57.9	7894.2	4759.2	426	3357	318
bigblue1	207.67	217.6	100.02	110.92	96.32	96.80	97.42	32.8	40.6	2694.6	2506.8	226.2	1369.2	240.6
bigblue2	300.83	451.2	153.75	162.81	154.91	152.34	145.78	126.6	139.6	6057.6	4833	577.2	3693	496.8
bigblue3	881.04	944.8	411.59	405.40	365.59	344.10	339.78	116.1	116.2	12554.4	10976.4	1295.4	5113.8	827.4
bigblue4	1727.5	2353.7	871.29	1016.19	834.19	829.44	808.22	575.9	586.8	29343	34029	2455.8	11389.8	2148

Table 6: Experimental measurements on ISPD-05 Placement Benchmarks

Benchmark	Required Memory (MB)	Benchmark	Required Memory (MB)
ibm01	4	ibm14	60
ibm02	8	ibm15	78
ibm03	9	ibm16	85
ibm04	11	ibm17	94
ibm05	13	ibm18	93
ibm06	14	adaptec1	103
ibm07	19	adaptec2	113
ibm08	22	adaptec3	199
ibm09	24	adaptec4	201
ibm10	32	bigblue1	123
ibm11	30	bigblue2	214
ibm12	34	bigblue3	392
ibm13	39	bigblue4	921

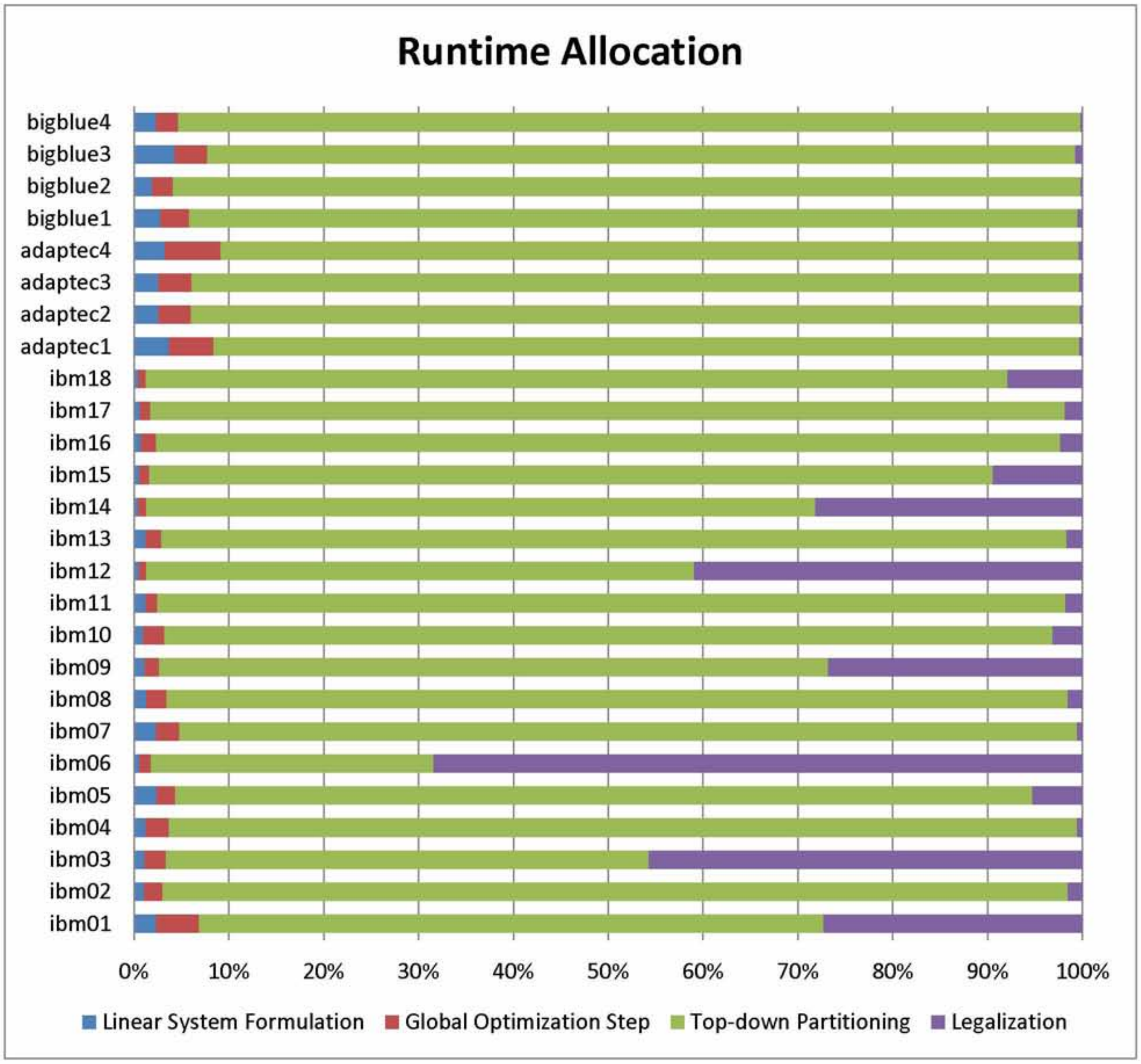
Table 7: Required Memory in MB

Memory footprint

By exploiting the star nets model, out GORDIAN implementation's memory footprint is significantly reduced. Table 7 show the memory required to place all benchmarks. Notice that the required memory is retained less than 1GB for all tested benchmarks.

Runtime allocation

More than 90% of the execution time is devoted to the top-down partitioning process and more specifically to the iterative solving method Bi-CG. The rest of the execution time is distributed among the initial linear system formulation, global optimization step and legalization as shown in Picture 20.



Picture 20: GORDIAN Runtime Allocation

6. Future work

Future placers based on our implementation have a considerable potential for further decreasing the total runtime. The iterative solver's convergence rate could benefit from graph based preconditioners. Solution quality could also be improved by optimizing the cutting process during the top-down partitioning steps, or even by comprising detailed placement techniques. Finally, congestion driven placement is possible using GORDIAN, by assigning lower net weights to the highly congested areas.

7. 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] Myung-Chul Kim, Dong-Jin Lee and Igor L. Marko, *"SimPL: An Effective Placement Algorithm", University of Michigan, Department of EECS, 2260 Hayward St., Ann Arbor, MI 48109-2121.*

[3] N. Viswanathan, C.ChongNuen Chu, *"FastPlace: Efficient Analytical Placement using Cell Shifting, Iterative Local Refinement and a Hybrid Net Model", Department of Electrical and Computer Engineering, Iowa State University, Ames, IA 500113060.*

[4] N. Viswanathan, M. Pan, C. Chu, *"FastPlace 2.0: an efficient analytical placer for mixed-mode designs.", Paper presented at the meeting of the ASP-DAC, 2006.*

[5] N. Viswanathan, M. Pan, C. Chu, *"FastPlace 3.0: A Fast Multilevel Quadratic Placement Algorithm with Placement Congestion Control.", Paper presented at the meeting of the ASP-DAC, 2007.*

[6] J. Roy, D. Papa, S. Adya, H. Chan, J. Lu, A. Ng, I. Markov, *"Capo: Robust and Scalable Open-Source Min-cut Floorplacer", ISPD, pp. 224-227, San Francisco, April 2005.*

[7] A. Caldwell, A. Kahng, I. Markov, *"Can Recursive Bisection Alone Produce Routable Placements?", DAC 2000, pp. 477 – 482.*

[8] X. Wang, M.Sarrafzadeh. *"Dragon2000: Standard-cell placement tool for large industry circuits.", ICCAD, pages 160–163, 2000.*

[9] X. Yang, Bo-Kyung Choi, M. Sarrafzadeh, *"A Standard-Cell Placement Tool for Designs with High Row Utilization.", Paper presented at the meeting of the ICCD, 2002.*

[10] M. Wang, X. Yang, M. Sarrafzadeh. *"DRAGON2000: Standard-Cell Placement Tool for Large Industry Circuits.", ICCAD, page 260-263. IEEE, (2000).*

[11] A. Agnihotri, S. Ono, P. Madden, *"Recursive Bisection Placement: Feng Shui 5.0 Implementation Details", International Symposium on Physical Design, April 2005.*

[12] Chan, T. Cong, J. Shinnerl, R. Joseph, S. Kenton, M. Xie, *"mPL6: enhanced multilevel mixed-size placement.", Paper presented at the meeting of the ISPD, 2006.*

[13] A. Kahng, S. Reda, Q. Wang, *"APlace: a general analytic placement framework.", Paper presented at the meeting of the ISPD, 2005.*

[14] C. Sechen, A. Sangiovanni-Vincentelli, *"The TimberWolf Placement and Routing Package.", IEEE Journal of Solid-State Circuits.*

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

[16] N. Gould, M. Hribar, J. Nocedal, *"On the Solution of Equality Constrained Quadratic Programming Problems Arising in Optimization.", SIAM Journal of Scientific Computing. 23 (4), April 2001, pp. 1376–1395.*

[17] D. Chen, S. Toledo. *"Implementation and evaluation of Vaidya's preconditioners.", Preconditioning 2001.*

[18] I. Koutis, G. Miller, A. Sinop, D. Tolliver, *"Combinatorial Preconditioners and Multilevel Solvers for Problems in Computer Vision and Image Processing", Technical Report CMU, 2009.*

[19] A. E. Caldwell, A. B. Kahng, I. L. Markov, *"VLSI CAD Bookshelf", http://vlsicad.eecs.umich.edu/BK*

[20] N. Viswanathan, C. Chu. *"ISPD04 IBM Standard Cell Benchmarks with Pads.", http://vlsicad.eecs.umich.edu/BK/Slots/cache/www.public.iastate.edu/~nataraj/ISPD04_Bench.html*

[21] G. Nam, C. Alpert, P. Villarrubia, B. Winter, M. Yildiz. *"The ISPD2005 placement contest and benchmark suite.", In Proc. ISPD, pages 216–220, 2005. http://archive.sigda.org/ispd2005/contest.htm*

[22] G Nam. *"ISPD 2006 placement contest: Benchmark suite and results.", In Proc. ISPD, pages 167–167, 2006. http://archive.sigda.org/ispd2006/contest.html*

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

[24] http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/Placement/plFormats.html

[25] https://www.semiwiki.com/forum/content/1955-rtl-design-flow-broken.html

[26] http://vlsicad.eecs.umich.edu/BK/PlaceUtils/

[27] https://www.cise.ufl.edu/research/sparse/CSparse/

[28] http://www.gnuplot.info/

[29] http://ndevilla.free.fr/gnuplot/

[30] http://openmp.org/wp/

[31] http://www.nvidia.com/object/cuda_home_new.html

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

8. Bibliography

[1] A. Caldwell, A. Kahng, I. Markov, *“Improved Algorithms for Hypergraph Bipartitioning.”, ASPDAC 2000, pp. 661-666.*

[2] A. Caldwell, A. Kahng, I. L. Markov, *“Can Recursive Bisection Alone Produce Routable Placements?”, DAC’00, p. 477.*

[3] S. N. Adya et al., *“Benchmarking for Large-Scale Placement and Beyond.”, IEEE Trans. on CAD 23(4), pp. 472-488, 2004.*

[4] A. Agnihotri et al., *“Fractional Cut: Improved recursive bisection placement.”, ICCAD, 2003, pp. 307-310.*

[5] U. Brenner and J. Vygen, *“Faster Optimal Single-Row Placement with Fixed Ordering.”, DATE 2000, pp. 117-121.*

[6] N. Selvakkumaran and G. Karypis, *“THETO: A Fast and High-Quality Partitioning Driven Global Placer.”, Technical Report 03-046, 2003, University of Minnesota.*

[7] A.E.Dunlop and B.W.Kernighan. *“A procedure for placement of standard-cell vlsi circuits.”, IEEE Trans. on CAD of Integrated Circuits and Systems, pages CAD–4(1):92–98, 1985.*