The Physical Placement Problem in Integrated Circuits

Prof. Kurt Keutzer
EECS
University of California
Berkeley, CA
Thanks to Prof. A. Kahng

Class News

General theme – students are responsible for reading articles
Class time will be spent on key points and providing more problem context
**Class Quiz**

Take 10 minutes and:

- 1) What’s the principal advantage of Fiduccia-Matheyses partitioning over Kernighan-Lin?
- 2) Briefly, what key elements does the Fiduccia-Matheyses approach use to accomplish this?

---

**Two-Way Partitioning: Fiduccia & Mattheyses**

If $p(i) =$ no. of pins on cell $i$:

$$-p(i) < g_i < p(i)$$

Bin-sort cells on $g_i$

- $p_{max}$
- $-p_{max}$

Time required to maintain each bucket array $O(P)/\text{pass}$
**RTL Design Flow**

- Library/module generators
  - HDL
  - RTL Synthesis
  - Logic optimization
  - Physical design
  - Layout

**Physical Design: Overall Conceptual Flow**

- Input
  - Read Netlist
  - Floorplanning
    - Initial Placement
    - Routing Region Definition
    - Global Routing
    - Cost Estimation
  - Placement Improvement

- Placement
  - Routing Region Ordering
  - Detailed Routing
  - Cost Estimation
  - Placement Improvement

- Output
  - Compaction/clean-up
  - Write Layout Database
**“Industrial” Flow**

**Definitions:**
- **Cell**: a circuit component to be placed on the chip area. In placement, the functionality of the component is ignored.
- **Net**: specifying a subset of terminals, to connect several cells.
- **Netlist**: a set of nets which contains the connectivity information of the circuit.

**Formulation of the Placement Problem**

**Given:**
- A netlist of cells from a pre-defined semiconductor library
- A mathematical expression of that netlist as a vertex-, edge-weighted graph
- Constraints on pin-locations expressed as constraints on vertex locations / aspect ratio that the placement needs to fit into
- One or more of the following: chip-level timing constraints, a list of critical nets, chip-level power constraints

**Find:**
- Cell/vertex locations to minimize placement objective subject to constraints

**Objectives:**
- minimal delay (fastest cycle time)
- minimal area (least die area/cost)
- other niceties: e.g. power
Results of Placement

A bad placement

A good placement

Global and Detailed Placement

In global placement, we decide the approximate locations for cells by placing cells in global bins.

In detailed placement, we make some local adjustment to obtain the final non-overlapping placement.
Placement Footprints:

Standard Cell:

Data Path:

IP block - Floorplanning

Placement Footprints:

Reserved areas

Core

IO

Control

Mixed Data Path & sea of gates:

A. Kahng

Kurt Keutzer
Placement Footprints:

Perimeter IO

Area IO – ball grid array

Complexity
space: $O(m)$ time: $Q(m^{1.5} \log^2 m)$

Final placement
*standard cell  *macro-cell & SOG
**Gordian: A Quadratic Placement Approach**

- Global optimization: solves a sequence of quadratic programming problems
- Partitioning: enforces the non-overlap constraints

**GORDIAN (quadratic + partitioning)**

Initial Placement

Partition and Replace

A. Kahng
Gordian Placement Flow

J. Kleinhaus, G. Sigl, F. Johannes, K. Antreich,
GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization,

Library

Contains for each cell:
- Functional information: cell = a * b * c
- Timing information: function of
  - input slew
  - intrinsic delay
  - output capacitance
  - non-linear models used in tabular approach
- Physical footprint (area)
- Power characteristics

Wire-load models - function of
- Block size
- Wiring
Library, Netlist, and Aspect Ratio

Netlist - >100K -> 10M cells from library

Size and aspect ratio of core die

Setting up Global Optimization

Fig. 1. Data flow in the placement procedure GORDIAN.
**GORDIAN: Global Placement**

We want to optimize the delay of critical paths
Instead we:
- optimize the sum of squares of net-lengths times a static weight

Global placement by quadratic wire-length optimization
- Problem is computationally tractable and well behaved
- Global connectivity is considered at all stages
  - An increasing number of constraints is imposed
  - Global placement of modules is obtained simultaneously for all sub-problems
  - No dependence on processing sequence

Quadratic placement clumps cells in center
Partitioning spreads cells and imposes new constraints on further optimization

**Intuitive formulation**

Given a series of points $x_1, x_2, x_3, \ldots x_n$
and a connectivity matrix $C$ describing the connections between them
(If $c_{ij} = 1$ there is a connection between $x_i$ and $x_j$)
Find a location for each $x_j$ that minimizes the total sum of all spring tensions between each pair $<x_i, x_j>$

Problem has an obvious (trivial) solution – what is it?
**Improving the intuitive formulation**

To avoid the trivial solution add constraints: \( Hx = b \)

- These may be very natural - e.g. endpoints (pads)

\[ x_1 \quad \ldots \quad x_n \]

To integrate the notion of "critical nets"

- Add weights \( w_{ij} \) to nets

\[ w_{ij} \]

\[ w_{ij} \] - some springs have more tension should pull associated vertices closer

**Modeling the Net’s Wire Length**

The length \( L_v \) of a net \( v \) is measured by the squared distances from its points to the net’s center

\[ L_v = \sum_{u \in M_v} [(x_{uv} - x_v)^2 + (y_{uv} - y_v)^2] \]

\[ (x_{uv} = x_u + \xi_{uv} ; \quad y_{uv} = y_u + \eta_{vu}) \]
What do we really want to optimize?

For high-performance circuits, we want to minimize longest path through network

- In addition to total wire length

Approximate delay minimization by minimization of squared wire length

- Penalizes long wires

Quadratic Objective Function

Objective function: weighted sum of the squared net lengths

\[ \Phi = \sum_{v \in N} L_v w_v \]

Where \( N \) is nets set, \( w_v \) is net weight, and \( L_v \) is the measure of net length

Can re-formulate it in terms of block coordinates (in one 1-D)

\[ \Phi(x) = \sum w_{ij} (x_i - x_j)^2 = 0.5x^T C x \]

- \( x_i \) - locations of vertices (modules)
- \( w_{ij} \) - edge weights (net weights)
- \( C \) - the system matrix

Generalize to 2-D

\[ \Phi(x,y) = \sum w_{ij} [(x_i - x_j)^2 + (y_i - y_j)^2] = 0.5x^T C x + 0.5y^T C y \]
**Formulating Optimization Problem**

For simplicity, we’ll consider a 1-D formulation

Need to add a term to account for fixed modules

\[ \Phi(x) = 0.5 x^T C x + d^T x \]

The vector \( d \) is constructed based on coordinates of the fixed modules and pin coordinates of all modules

Also need to constraint placement s.t. center-of-gravity (the area weighted mean of all coordinates) corresponds to the center of placement region:

\[ Ax = u \]

The vector \( u \) contains geometric centers of placement regions, the matrix \( A \) reflects the assignment of modules to placement regions

---

**Quadratic Optimization Problem**

- Linearly constrained quadratic programming problem

\[ \min_{x \in \mathbb{R}^m} \{ \Phi(x) = x^T C x + d^T x \} \]

\[ \text{s.t. } A^{(l)} x = u^{(l)} \]

\[ A^{(l)} = \begin{bmatrix} \rho & * & * & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & * & * & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots \end{bmatrix} \]

- Accounts for fixed modules
- Wire-length for movable modules
- Center-of-gravity constraints

Problem is computationally tractable, and well behaved
Global Optimization Using Quadratic Placement

Quadratic placement clumps cells in center

Partitioning divides cells into two regions
- Placement region is also divided into two regions

New center-of-gravity constraints are added to the constraint matrix to be used on the next level of global optimization
- Global connectivity is still conserved

What do I need to know about QP?

- Get an intuitive sense of what quadratic programming is trying to solve
  - Create a good placement of the netlist by placing cells so as to minimize the squares of the wirelengths in the netlist
- Understand strengths and weaknesses of the formulation
  - Strength - optimal solution!
  - Weakness –
    - quadratic vs. linear wirelengths
    - Never knows (or optimizes) the length of a path - only the individual 2-pin nets
- Don't need to understand the mathematical solutions
Setting up Global Optimization

Fig. 1. Data flow in the placement procedure GORDIAN.

Layout After Global Optimization

A. Kahng
In GORDIAN, partitioning is used to constraint the movement of modules rather than reduce problem size.

By performing partitioning, we can iteratively impose a new set of constraints on the global optimization problem:
- Assign modules to a particular block

Partitioning is determined by:
- Results of global placement
  - Spatial (x,y) distribution of modules
- Partitioning cost
  - Want a min-cut partition
**Partitioning due to Global Optimization**

Sort the modules by their x coordinate (for a vertical cut)

Choose a cut line such that area is approximately equal between two sides of cut

\[ M_p \rightarrow M_p', M_p'' \]
\[ x_u \leq x_u'' \quad u' \in M_p', u'' \in M_p'' \]
\[ \alpha = \frac{\sum F_u}{u'TM_p} / \frac{\sum F_u}{u \in M_p} \approx 0.5 \]

---

**Partitioning Improvement - I**

- The cost of initial partition may be too high
- Can change position of the cut to reduce the cost
- Plot the cost function, choose “best” position

\[ M_p \rightarrow M_p', M_p'' \]
\[ x_u \leq x_u'' \quad u' \in M_p', u'' \in M_p'' \]
\[ \alpha = \frac{\sum F_u}{u' \in M_p'} / \frac{\sum F_u}{u \in M_p} \approx 0.5 \]

cut value: \[ C_p(\alpha) = \sum_{v \in N \forall} \]

\[ \begin{array}{c}
\text{0.0} & \text{0.25} & \text{0.5} & \text{0.75} & \text{1.0} \\
\text{0} & \text{10} & \text{20} & \text{30} & \text{40}
\end{array} \]
**Adding Positioning Constraints**

- Partitioning gives us two new "center of gravity" constraints
- Simply update constraint matrix
- Still a single global optimization problem
- Partitioning is not "absolute"
  - modules can migrate back during optimization
  - may need to re-partition

\[
A^{(i)} = \begin{bmatrix}
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\ast & \ast & \ast & 0 & 0 & 0 \\
0 & 0 & 0 & \ast & \ast & \ast \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots 
\end{bmatrix}
\]

Fig. 4. The constraints for global placement.
**Continue to Iterate**

**GORDIAN**

**Procedure** Gordion

\[
l := 1; 
\]

\[
global-optimize(l); 
\]

\[
while(\|M_l\| > k) 
\]

\[
for each \rho \in R(l) 
\]

\[
partition(\rho, \rho', \rho''); 
\]

\[
endfor 
\]

\[
l := l+1; 
\]

\[
setup-constraints(l); 
\]

\[
global-optimize(l); 
\]

\[
/* extras 
\]

\[
repartition(l); */ 
\]

\[
endwhile 
\]

\[
final-placement(l); 
\]

\[
endprocedure 
\]

**Fig. 1.** Data flow in the placement procedure GORDIAN.
First Iteration

Second Iteration
**GORDIAN (quadratic + partitioning)**

**Initial Placement**

**Partition and Replace**

**Final Placement**

Fig. 1. Data flow in the placement procedure GORDIAN.
**Final Placement - 1**

Earlier steps have broken down the problem into a manageable number of objects.

Two approaches:

- Final placement for standard cells/gate array – row assignment
- Final placement for large, irregularly sized macro-blocks – slicing – look over in “Extra” slides at the end

---

**Final Placement – Standard Cell Designs**

This process continues until there are only a few cells in each group ($\approx 6$)

Channeled Gate Array

Standard Cell Layout
Final Placement – Creating Rows

Partitioning of circuit into 32 groups. Each group is either assigned to a single row or divided into 2 rows.

Final Placement – Creating Rows

Partitioning must be done breadth-first not depth first.

Creating Rows

Choose $\alpha$ and $\beta$ preferably to balance row to balance row length (During re-arrangement.)
**Final Placement**

Earlier steps have broken down the problem into a manageable number of objects.

Two approaches:

- Final placement for standard cells – row assignment
- Final placement for large, irregularly sized macro-blocks – slicing – see Extras

---

**Generating Final Placement**

![Diagram](image)

Fig. 1. Data flow in the placement procedure GORDIAN.
**Example Final Placement**

![Macrocell design with standard cell blocks scb8 and scb9](image)

**Summary of status on placement**

Most place and route tools use a series of “half-truths” regarding timing modeling and delay constraints.

Not uncharacteristically, the Gordian approach uses three different approaches to attack the problem - each approach makes a number of different simplifications.

Current gate-level placement tools (e.g. Apollo) are able to place and route hundreds of thousands to millions of cells flat (without hierarchy).

Despite the lack of direct correlation between the internal timing model and the actual integrated circuit timing, timing closure between synthesis and physical design is improving.

Principally due to:

- Better delay *estimation* in synthesis
- Better delay *calculation* in physical design
- Automated buffer insertion in routing
**Extras**

- Brief survey of approaches
- More on quadratic placement
- Using slicing to handle macro-blocks

**Traditional Approaches**

- Quadratic Placement
- Simulated Annealing
- Bi-Partitioning
- Quadrisection
- Force Directed Placement
- Hybrid
Overview of Gordian Package

GORDIAN with repartitioning procedure

Procedure Gordian

\[ l := 1; \]
\[ \text{global-optimize}(l); \]
\[ \text{while}(\exists |M_i| > k) \]
\[ \text{for each } \rho \in R(l) \]
\[ \text{partition}(\rho, \rho', \rho''); \]
\[ \text{endfor} \]
\[ l := l + 1; \]
\[ \text{setup-constraints}(l); \]
\[ \text{global-optimize}(l); \]
\[ \text{repartition}(l); \]
\[ \text{endwhile} \]
\[ \text{final-placement}(l); \]
\[ \text{endprocedure} \]

Cost Function

Overall objective

\[ \phi = \frac{1}{2} \sum_{v \in V} L_v w_v \]

\[ \phi (x, y) = X^T C X + d_x^T X + Y^T C Y + d_y^T Y \]

\[ \xi_{uv} \]

\[ \phi (x) = X^T C X + d^T X \]
Global Placement and Constraints

The center of gravity constraints
At level l, chip is divided into \( q(\leq 2^l) \) regions
For region \( p \), the center coordinates: \((u_p, v_p)\)
constraints: \( \sum_{u \in M_p} F_u x_u = u_p \sum_{u \in M_p} F_u \)
\((M_p: \text{set of modules in region } p)\)
Matrix form for all regions
\[ A^{(l)} X = t^{(l)}, \quad a_{pu} = \begin{cases} \frac{F_i}{\sum_{i \in M_p} F_i} & \text{if } i \in M_p \\ 0 & \text{otherwise} \end{cases} \]

Problem Formulation

\[ E = (u_p, v_p) \]

\[ A^{(l)} = \begin{bmatrix} A & B & C & D & E & F & G \\ : & : & : & : & : & : & : \\ : & : & : & : & : & : & : \\ : & : & : & : & : & : & : \\ \rho & * & * & 0 & 0 & 0 & \ldots \\ \rho & 0 & 0 & * & * & \ast & \ldots \\ : & : & : & : & : & : & : \end{bmatrix} \]

\( A^{(l)} X = t^{(l)} \)

\( LQP: \min_{x \in \mathbb{R}^m} \{ \Phi(x) = X^T C X + d^T X \} \) s.t. \( A^{(l)} X = t^{(l)} \)
Solution Method

\[ A_{q \times m} = [D_{pq} B_{q \times (m-q)}] \]

\[ \begin{bmatrix} X_d \\ X_i \end{bmatrix} = u \]

\[ X_d = -D^T B X_i + D^T u \]

\[ X = \begin{bmatrix} -D^T B & I \\ I & 0 \end{bmatrix} X_i + \begin{bmatrix} D^T u \\ 0 \end{bmatrix} = Z X + X_0 \]

unconstrained quadratic programming problem

\[ \text{UQP: } \min_{x_i \in R^{m-q}} \{ \psi(x_i) = X_i^T Z^T C X + C^T X_i \} \quad (C^T = C X_0 + d) \]

Solved by conjugate --gradient method

Terminal Propagation

- We should use the fact that \( s \) is in \( L_1 \)!

- Assuming located at center

  Fictitious cell of net \( s \)

- Prefer to have all of them in \( R_1 \)

- \( p \) will stay in \( R_1 \) for the rest of partitioning
Macro Placement by Slicing

There are different ways of placing cells in each region.

Want to choose placements that minimize total chip area.

Constraining Placement of Macros

Global optimization and partitioning assigns <=k cells to each physical region.

There are different ways of placing cells in each region.

Want to choose placements that minimize total chip area.
Macro-blocks: Exhaustive Slicing Optimization

L. van Ginnekin, R. H. Otten, Optimal Slicing of Point Placements, EDAC, 1990, pp. 322-326

procedure ESO

for all regions \( \varrho \) with \( |\mathcal{M}_\varrho| \leq k \) modules

Determine the shape function of region \( \varrho \) by enumeration of all slicing structures that can be derived from the global placement coordinates of the modules;

endfor

Recursively compute the shape function of the root region bottom up from the shape functions of all ESO regions;

Select one shape for the root region;

Traverse the slicing tree top down to determine the module coordinates and shapes;

endprocedure

Fig. 8. Exhaustive slicing optimization procedure.

Slicing vs. Non-Slicing Floorplans

Slicing Floorplan (slicing structure): A rectangle dissection which can be obtained by recursively dissecting the base rectangle into smaller rectangles by vertical and horizontal slicing lines.

Slicing structures are good for routing

Slicing Floorplan  Non-Slicing Floorplan (wheel)
The Shape Algorithm

Input:  
- Slicing Tree
- Shape Constraints for modules
- Cost Function (non-decreasing in w, h)

Output:  
- Shapes/Implementation for each module

Algorithm

- Compose shape constraints bottom-up in the slicing tree
- Apply cost function to compute boundary point on shape constraint of base block (root)
- Propagate boundary point top-down in slicing tree to obtain implementation for each module
**Shape Constraints**

Given a rectangular module, the shape constraint relation $R$ is the set of $y$-$x$ pairs so that a rectangle with width equal to $x$ and height equal to $y$ contains at least a shape/orientation realization of the module.

**Composing Shape Constraints**
**Slicing Tree**

Traverse tree bottom-up deriving composed shape functions

---

**The Shape Algorithm**

**Input:**
- Slicing Tree
- Shape Constraints for modules
- Cost Function (non-decreasing in w, h)

**Output:** Shapes/Implementation for each module

**Algorithm**

- Compose shape constraints bottom-up in the slicing tree
- Apply cost function to compute boundary point on shape constraint of base block (root)
- Propagate boundary point top-down in slicing tree to obtain implementation for each module
**Constraining Placement of Macros**

At top level can have complex shape functions

Apply a cost function and select lowest score point, e.g.

- \( \text{cost}(w,h) = wh \)
- \( \text{Cost}(w,h) = 2(w+h) \)

**Slicing Tree**

```
  V
 / \  
H   H
 / \ / \  
A   V  E  H
 / \ / \ / \  
B   C   F   G
```

30 X 50

Insert your logic here

30 X 50
The Shape Algorithm

Input:  
- Slicing Tree
- Shape Constraints for modules
- Cost Function (non-decreasing in w, h)

Output:  
- Shapes/Implementation for each module

Algorithm
- Compose shape constraints bottom-up in the slicing tree
- Apply cost function to compute boundary point on shape constraint of base block (root)
- Propagate boundary point top-down in slicing tree to obtain implementation for each module

Slicing Tree

Propagate optimal choices top-down to all the leaves
Min-Cut Based Placement (Cont’d)