18760: Project 2

ANALYTICAL PLACER

May 6, 2015

Andrew Mort (amort@andrew.cmu.edu)
Chad Cole (cjcole@cmu.edu)

Introduction

The purpose of this project is to create a tool to automate gate placement in ASIC design. Our solution is analytical, which is to say it starts with a placement that is random and that does not consider overlap between gates. The tool iteratively works towards an optimal solution for the locations of all of the gates that focuses on the length of the nets and the density of the gate placement while staying inside a predetermined boundary. The solution is found using smooth cost functions that can be minimized using numerical techniques.

Goals

- **75% Area Utilization**
- **Efficient Usage of Memory and Operational Overhead**
- **Efficient Gradient and Cost Calculation**

Cost Function Alorithms

In order to minimize the cost function and calculate the gradient, smooth, differentiable functions are needed to calculate the cost.

Wirelength

The length of a net is calculated using the half-perimeter method. That is, for all of the gates connected to a certain node, that nodes cost is:

$$C_{\text{wirelength}} = \max(x_1, x_2, ..., x_n) - \min(x_1, x_2, ..., x_n) + \max(y_1, y_2, ..., y_n) - \min(y_1, y_2, ..., y_3)$$

This however, is not differentiable so we use the Logarithmic-Exponential analytical form for max:

$$\max(x_1, x_2, ..., x_n) = \alpha \log\left(e^{x_1/\alpha} + e^{x_2/\alpha} + ... + e^{x_n/\alpha}\right)$$
$$\min(x_1, x_2, ..., x_n) = -\alpha \log\left(e^{-x_1/\alpha} + e^{-x_2/\alpha} + ... + e^{-x_n/\alpha}\right)$$

Substituting these functions into the cost function above will yield a differentiable function and thus allow the computation of the gradient.

Density

The density cost is determined by imposing a grid over the circuit and assigning a capacity to each grid point then counting how many gates are touching that grid point. The function takes the form:

$$\sum_{\text{gridpoints } g} (C_g - \#\text{ of gates touching grid g})^2$$

Here, again, we run into the non-differentiable problem because appurtenance is discontinuous. However, if we adjust our definition of belonging from an "in" or "out"

decision, then this function becomes smooth. This function would need to account for the area of the cell. Below is a smooth, differentiable function that represents the degree of appurtenance of each cell to each grid, and then the new cost function is shown.

$$p(d) = \begin{cases} 1 - 2d^2/r^2 & (0 \le d \le r/2) \\ 2(d-r)^2/r^2 & (r/2 \le d \le r) \end{cases}$$

$$\text{Potential(c, g)} = \text{area}(c)/r^2 \cdot p(|x_g - x_c|) \cdot p(|y_g - y_c|)$$

$$\text{Penalty} = \sum_{\text{Grid } g} \left[\left(\sum_{\text{Cell } c} \text{Potential}(c, g) \right) - \text{Capacity}(g) \right]^2$$

Boundary

The boundary penalty is more simple than the last two cost functions. It has two components: one for the left and one for the right sides of the boundary. Shown below is the penalty for the x direction, it is the same for the y direction.

$$\text{Penalty-Left(x)} = \left[x < x_{\text{left}} \implies \left(\frac{x - x_{\text{left}}}{\alpha} \right)^2 \right]$$

$$\text{Penalty-Right(x)} = \left[x < x_{\text{right}} \implies \left(\frac{x - x_{\text{right}}}{\alpha} \right)^2 \right]$$

Milestone Placer

Data Structures

Locations for the gates are stored in an array of 2-tuples containing the x and y coordinates. The array is of length $2N$, where N is the number of gates in the circuit.

Gates are stored in an array of arrays containing that gate's nets. The total length of this array is $C \cdot N$ where $C \ll N$.

Nets are stored in an inverse structure to the gates — the net structure is a vector of the 2-tuples that contain both the pin, if any is connected to this net, and a vector of the gates connected to the net. The total space required for this structure is $2 \cdot C \cdot M$, where M is the number of nets and C is some constant, $C \ll M$.

Pins are stored in an array of 3-tuples that contain the attached net identifier as well as the x and y coordinates of the pin. The total space used for this structure is $3 \cdot P$, where P is the number of pins.

This redundant data is kept because we prioritized a lower operational-overhead over a lower memory overhead. These structures allow for a constant lookup time given any identifier (the gate name, net name, or pin name). The total spacial overhead is $2N + C(N + 2M) + 3P$.

Milestone Placer continued on next page...

Gradient Calculation

In the gradient calculation, we are looking to see how much the cost changes when you move a gate in either the x or y direction. This must be done for all of the gates. We broke up the code into several functions for readability. The gradient calculation is shown below:

$$\frac{\partial f}{\partial x_i} = \frac{F(x_i + h) - F(x_i)}{h}$$
$$\frac{\partial f}{\partial y_i} = \frac{F(y_i + h) - F(y_i)}{h}$$

The gradient calculates this derivative for both coordinates of every cell in the circuit. The functions below are summed together to create an expression for the components representing each of our penalties.

Wirelength: For a specific gate moving in a specific direction, you will go through all of the nets connected to that gate and recalculate the smooth wire length for each of those nets with the initial and with the new location. The difference between these two values is returned and used to find the derivative for that specific configuration. This results in a complexity, for a single gate, of O(1) if the number of the nets connecting to a gate is much smaller than the number of gates.

Density: We implemented this function in a rather inefficient way. The density was fully calculated twice, once for the original density and again for the new density. The complexity of this function was $O(M \cdot N)$, where N is the number of gates and M is the number of grid points. This is due to the fact that we must visit every grid point, and, for each grid point visited, we must visit every cell.

Boundary: We calculated the old and the new boundary cost for the placement of that gate and take the difference. The complexity of this function is O(1)

The overall complexity of the gradient function is $O(N^2 \cdot M)$ because the gradient has to be recalculated for every gate.

Difficulties

Although our cost functions were accurate, we didn't know how to correctly call the gradient descent and therefore our placement wasn't valid.

Final Placer

Data Structures

The only change in the Final Placer from the Milestone was the addition of a Grid-points data structure, made with an array of doubles to store the value of the density of the grid-points to make the density calculation faster.

Final Placer continued on next page...

Density Calculation

This function was updated from the milestone by using the grid-points structure. It goes through every cell and for each cell, it'll find all the grid-points within the radius of the cell and updates the potential for just those grid-points. After going through all of the cells, the density penalty is formulated by visiting every grid-point and take the square of the difference between the new density and the capacity of the grid-point. The complexity of this function drops to $O(N + M)$, where N is the number of gate and M is the number of grid-points.

Density Gradient

The grid-point matrix used in the density cost calculation is also used to calculate the gradient. Before the gradient is calculated, the array is populated in the same way as before — by going through every cell and calculating the potential. Then, when calculating the gradient for a specific cell, we must find how the potential for all of the grid-points around the cells change, and then calculate the difference following the density function given above. This results in a final complexity for the whole gradient function of $O(N)$.

Test Results

Circuit	Wirelength	Utilization
toy1	749.78	29%
toy2	1371.10	31%
fract	5707.42	52%
industry1	74669.05	53%
primary1	35526.03	68%
struct	61857.62	57%

Final Notes

After playing around with the loop that runs the gradient descent function, we found that two iterations were enough to generate a good placement. We start with a course grid, a small radius, and a higher weight on the wirelength. In the next iteration, we have a much finer grid, a larger radius, and the weight of the wirelength is lower than the weight of the weight density.