

**EE 6375**

**DESIGN AUTOMATION OF VLSI SYSTEMS**

|  |
| --- |
| **PROJECT #3**  **PLACEMENT ALGORITHM BASED ON TIMBERWOLF** |
|  |

***Abstract – Placement is the process of assigning geometrical locations to standard cells, whereas routing is the process of performing net connections on cells when a netlist is given. The primary goal of routing is to minimize the total wire length and to minimize the total number of bends. The output of placement is used as an input to routing, which makes the two processes interdependent, and hence wire length is an important consideration for placement algorithms. Simulated annealing is one of the most popular algorithms for performing placement. Several flavors of this algorithm have been developed with the aim of optimizing different constraints such as power, wire length and timing. Our algorithm implements Simulated annealing which is aimed at minimizing wire length. The algorithm is developed in C++ and tested on several benchmarks. The results are tabulated and plotted using Matlab.***

**INTRODUCTION:**

Placement is the process of arranging individual cells in a manner that would minimize the given objective function subject to some constraints. The objective function usually used is the wire length since in contemporary designs the total wire length when un-optimized is a very large value. Timberwolf is a standard cell placement and routing package. It first places the cells according to the simulated annealing algorithm. Once this is done, it routes the placed cells and then makes changes in order to get the lowest wire length possible. The Timberwolf placement and routing package provides substantial area and wire length reduction in existing cell layouts. In this project we focus on the placement algorithm.

Simulated annealing based on Timberwolf focusses mainly on reducing the wire length, while maintaining the aspect ratio of the chip. The cost function is the total wire length of all nets computed using bounding box model. Constraint is placed on the maximum chip area, which is chosen to be roughly two times the logic area. Another constraint is the aspect ratio which is kept as close to 1 as possible. Although pads are placed around the periphery of the chip, we considered these to be similar to other cells rather than separate entities. The project itself is written in C++.

**ALGORITHM:**

This algorithm tries to place standard cells by simulating the annealing process. A sufficiently high start temperature is chosen, and allowed to cool very slowly. At every temperature there are perturbations done on the solution set, which are accepted or rejected based on a function of cost and temperature. The solution is accepted or rejected after comparing the function against a random probability (ranged between 0 and 1). Good solutions are automatically accepted .For bad solutions , the temperature acts like a gatekeeper which initially accepts any solution, but only accepts good solutions as it cools down. The initial solution accepted is iteratively improved by local changes until no better solution is found. Simulated annealing allows for accepting bad solutions so as to not get stuck at locally good but globally bad solutions.

The influence for this algorithm is the actual annealing process in metals where the metal is heated to a high temperature and then cooled down very slowly. If the cooling process is slow enough to ensure thermal equilibrium and the heating temperature is sufficiently high to ensure random state, the atoms of the metal place themselves in a pattern that very closely resembles a perfect crystal. The algorithm can be summarized by the following steps:

**Initialize** - Random initial placement is created. The area of the chip is chosen to be 2X the area of the standard cells. There are empty rows above and below the standard cell rows.

**Disturb/Move** - Disturb the placement by swapping cells within a window range

**Calculate cost**- Due to the move made in the previous step, calculate the cost factor.

**Select** - Based on the cost factor or function, accept or reject the move made in step. There is a probability of acceptance here.

**Update** - The temperature is lowered and the window size is reduced appropriately. The algorithm is repeated from the Disturb/Move step. This is done until lowest temperature is reached in a manner similar to that of the simulated annealing process in metals.

We have considered the following parameters in designing algorithm:

**Start temperature** –

After the initial random placement, all the cells are allowed to swap. The wire length is computed at each step using the bounding box model.. The start temperature is such that it accepts the worst case change in wire length with a probability of 0.8

*Tstart=(wire\_min-wire\_max)/log(0.8);*

The value of 0.8 was arrived at after performing several iterations on one benchmark and checking what value gave a good start temperature.

**Cooling schedule:**

The temperature is allowed to cool very slowly. Every new temperature is 0.94 times the old temperature

**Type of moves:**

Our algorithm is designed to make only one kind of move ie swapping. Moving the cell to new location is skipped so as to reduce an extra overhead of removing overlaps. Reflection of cell about its axis is also skipped since the pin locations inside the cells are not known.

**Cost:**

The cost function is the wire length computed using the bounding box model. Since swapping two cells of same size does not cause any overlap, the computation of overlap penalty is not included.

**No of cells to swap within each temperature**

Within a given temperature, most of the cells are allowed to swap with each other. Our algorithm allows for swap of 80% of cells.

**Range delimiter:**

Each swap is performed only if the second cell falls within a window defined around the first cell. The window is initially defined to be 2X the size of the entire chip to allow any two cells within the chip to swap with each other.

As temperature is updated, height and width of the window are updated as follows:

*width= [width\_initial / log(Tstart) ]\*log(Temp)*

*height= [height\_initial / log(Tstart) ]\*log(Temp)*

where

*width\_initial ,height\_initial -> initial width and height*

*Tstart -> Start temperature*

*Temp-> Updated Temperature*

**Ending the algorithm:**

The simulated annealing process ends when one of the following happens:

* Width and height of the window are equal to width and height of the smallest standard cell
* The cost value remains unchanged for “500n” iterations.

n-> number of cells

500 was arrived at after experimenting and checking what value gave best results over all benchmarks

**Challenges faced :**

A major challenge faced in the algorithm was the computation complexity. Initially the algorithm calculated the complete wire length after swapping cells. For every swap, the wire length calculation was an order of complexity = number of nets .This increased the runtime for huge benchmarks. This was overcome by calculating only the change in wire length after every swap .With this method, the computation complexity for every swap = number of nets connected to swapped cells. For the biggest benchmarks (ibm16 and ibm18), this reduced the computation time from several hours to under 5 minutes.

Another challenge faced was deciding on when to end the algorithm. For huge benchmarks, as the window size becomes smaller, the random generator function has to be executed several times to find a suitable cell for swap. If the cost function is unchanged for several thousand iterations (based on size of the circuit), a Boolean variable is set to 1. This is used in the conditional part of the “*for loop”* to end the algorithm.

**IMPLEMENTATION DETAILS:**

*Cell:*

The data structure for a cell contains the following elements:

* Cell\_name - Name of cell
* Cell\_area- Area of cell
* “x,y” – stores x and y co-ordinates of lower left corner
* Head – stores the address of a data structure called net\_node

*Net\_node:*

Net\_node is a member of the netlist implemented as a linked list. It contains two elements:

* NP-Contains the address of the net
* next- contains the address of the next node in the linked list

*Net:*

The data structure for net contains the following elements:

* Net\_number
* Wire length- Wire length computed for the net using bounding box model
* Head – stores the address of a data structure called cell\_node

*Cell\_node:*

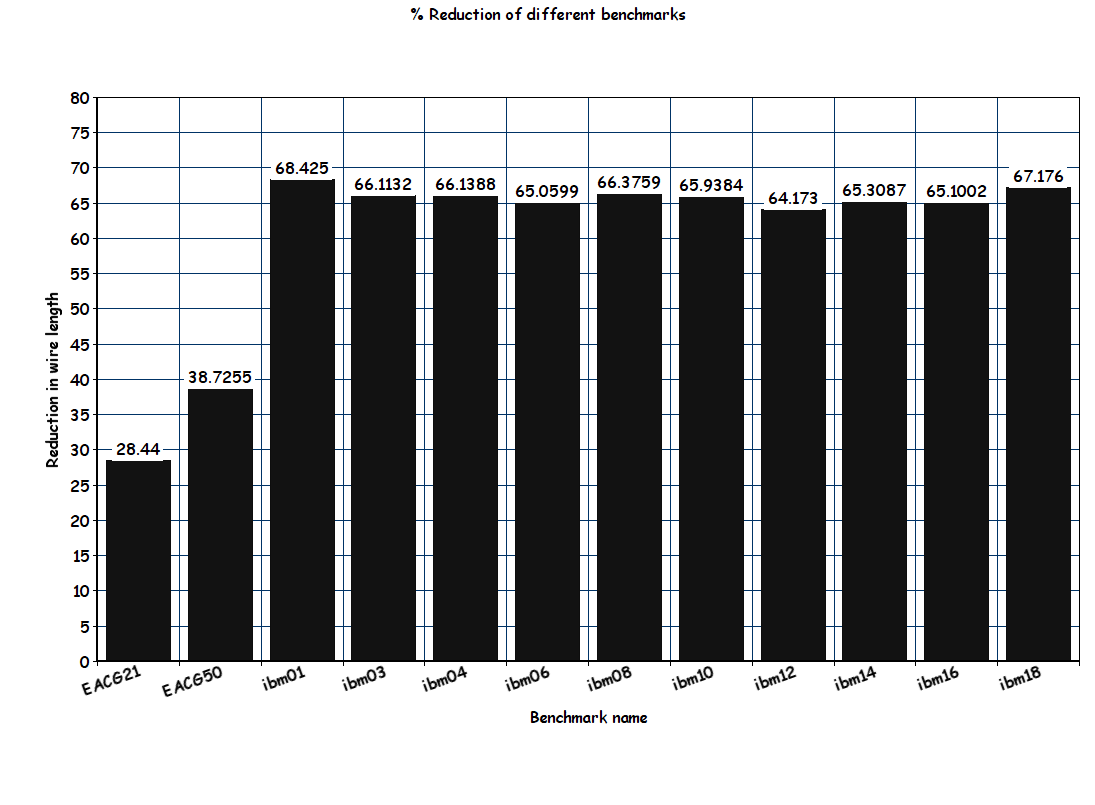
Net\_node is a member of the cell list implemented as a linked list. It contains two elements:

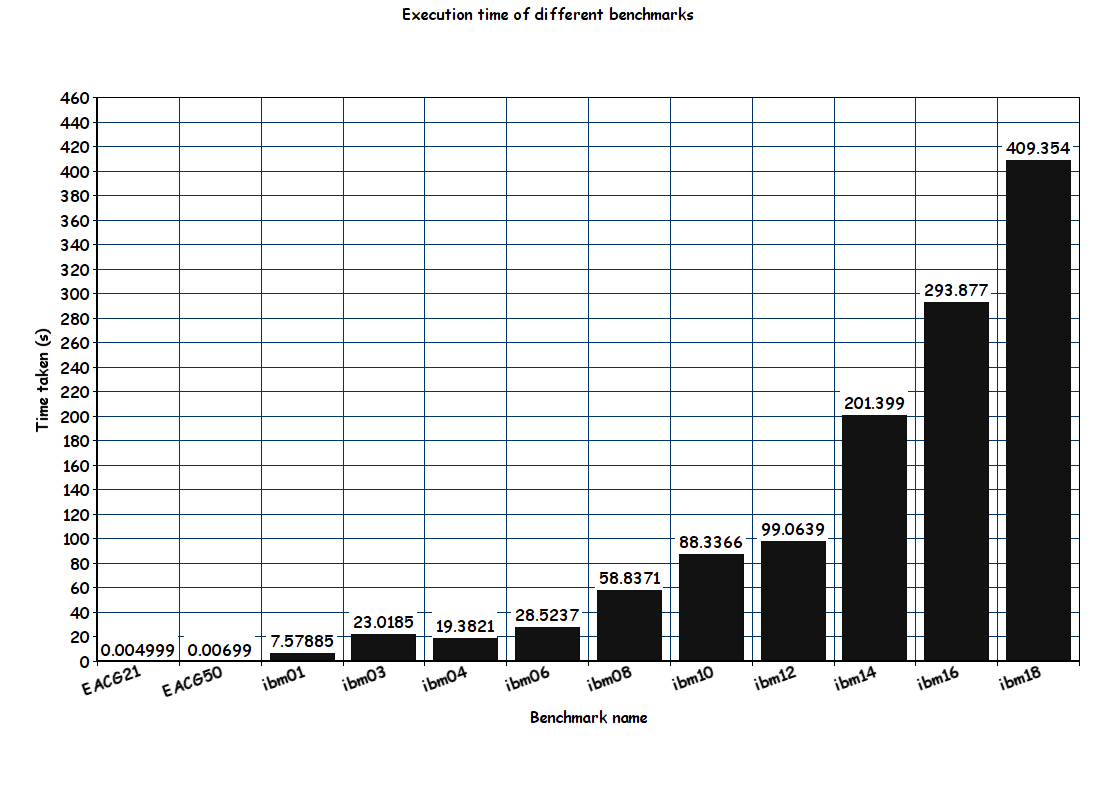
* CP-Contains the address of the cell
* next- contains the address of the next node in the linked list

**RESULTS:**

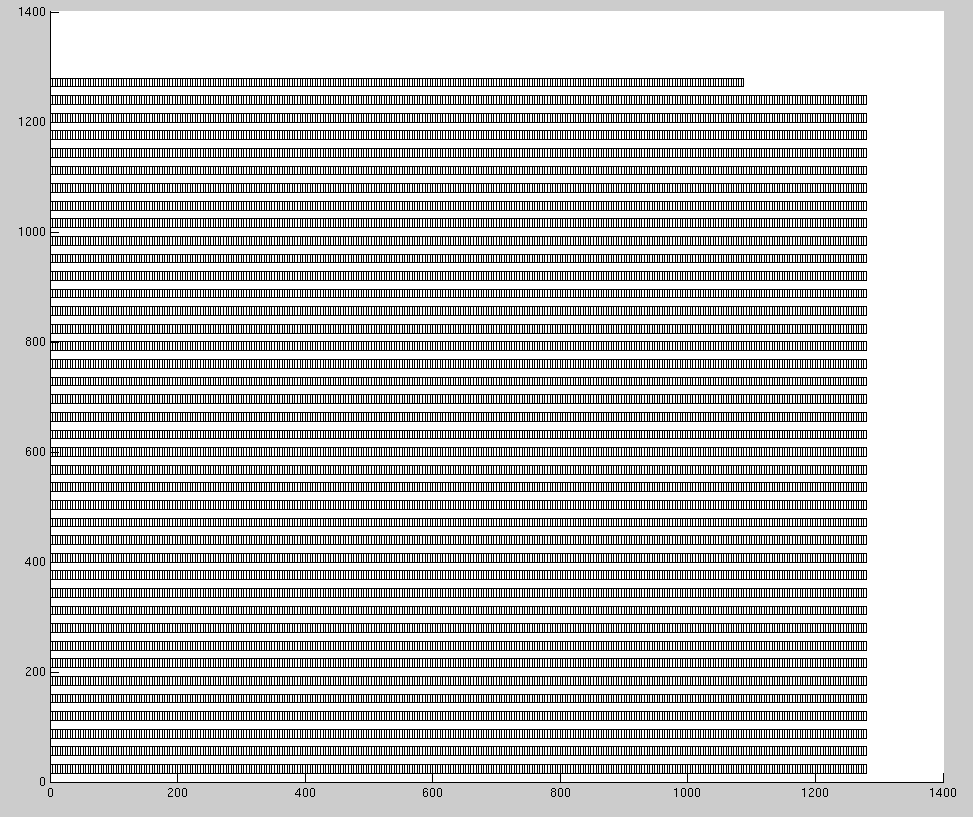
The following values are obtained when all the benchmarks are tested on the UTD engnx servers. The values are recorded and plotted. We observe a constant reduction in cutset by about 64% on average for all the benchmarks. The time taken for the execution of the benchmarks is observed to be reasonable.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Benchmark name** | **Execution time (s)** | **Initial wire length** | **Final Wire length** | **%age reduction** | **Aspect Ratio** |
| EACG21 | 0.004999 | 872 | 624 | 28.44 | 1.42857 |
| EACG50 | 0.00699 | 3264 | 2000 | 38.7255 | 1.33333 |
| ibm01 | 7.57885 | 1.66E+07 | 5.25E+06 | 68.425 | 1.0125 |
| ibm03 | 23.0185 | 4.22E+07 | 1.43E+07 | 66.1132 | 1.0116 |
| ibm04 | 19.3821 | 5.41E+07 | 1.83E+07 | 66.1388 | 1.01277 |
| ibm06 | 28.5237 | 6.63E+07 | 2.32E+07 | 65.0599 | 1.00781 |
| ibm08 | 58.8371 | 1.22E+08 | 2.32E+07 | 66.3759 | 1.00312 |
| ibm10 | 88.3366 | 2.09E+08 | 7.11E+07 | 65.9384 | 1.01477 |
| ibm12 | 99.0639 | 2.20E+08 | 7.88E+07 | 64.173 | 1.01326 |
| ibm14 | 201.399 | 5.97E+08 | 2.07E+08 | 65.3087 | 1.00368 |
| ibm16 | 293.877 | 8.70E+08 | 3.04E+08 | 65.1002 | 1.00577 |
| ibm18 | 409.354 | 1.01E+09 | 3.30E+08 | 67.176 | 0.999231 |





The placement co-ordinates are fed to a matlab script to verify that cells are placed in a row. A sample placement plot for ibm01 benchmark is shown below:



**CONCLUSIONS:**

Observations from executions:

* Initially even very bad solutions are accepted
* As the temperature cools down, the acceptance rate decreases drastically, and only good or relatively good solutions are accepted.
* Allowing cells to initially swap freely without much restriction ultimately contributes to a good placement

We see two possible improvements over our implementation:

1. Instead of using only one kind of movement, use both displacement and swapping initially. After a certain temperature, the row lengths can be adjusted and only swapping can be performed. If the absolute value of pin positions are known, reflection can be used in order to refine the placement.
2. The temperature updation is done with a fixed value. This could be made dynamic to reduce execution time.

**REFERENCES:**

[1] C. Sechen, A.S Vincentelli, "The Timberwolf Placement and Routing package", in IEEE journal of solid-state circuits, pp 510-522, April 1985

[2] Dr. Dinesh Bhatia , Class notes and lectures of fall 2015,UT Dallas

[3] B. Stroustrup, "Programming: Principles and practice using C++ "