Deepak S
Department of Metallurgical and Materials Engineering,
IIT Madras,
Chennai, Tamil Nadu
email: mm22b011@smail.iitm.ac.in

Anurag Manoj V
Department of Electrical Engineering
IIT Madras,
Chennai, Tamil Nadu
email:ee22b00e@smail.iitm.ac.in

ID5130_Course_Project

The project develops a parallel global router for VLSI (Very Large Scale Integration) design, tackling the intricate task of establishing paths between chip pins based on a provided netlist, while abiding by grid-based routing constraints, navigating through paths delineated as sequences of squares on a coarse grid superimposed over the chip area. Each edge of a grid cell imposes a restricted capacity for wire crossings. The primary goal is to devise solutions that adhere to capacity limits while concurrently minimizing the count of bends and wire lengths.

Keywords: Global Routing, Bellman-Ford, VLSI,netlist,OpenMP,Parallelism

1 Introduction

In the subject of Very Large Scale Integration (VLSI) design, optimizing the routing of connections between pins on a chip is a critical task. Physical design is a stage in the VLSI design process where an abstract netlist, and logic level design is converted to a physical design that can be sent for fabrication to fouEditor mode.ndries and fabfs. A step in this process is the routing of metal interconnects between pins on the various components of the chip. These interconnects are of the form of straight metal wires and vertical connections called vias, which are arranged into multiple layers. For manufacturing simplicity, these wires all travel in only one directions, by convention, horizontally in odd-numbered layers and vertical direction in even numbered layers. The computational problem of successfully making these connections in a way that minimizes crowding and wastage of chip area is an NP-Hard problem of great industrial relevance. Software from leading EDA companies like Cadence and Synopsys can take 2-3 days to route an industrial chip, causing delays in time to market. HarnessingEditor mode.Editor mode. the power of parallel computing is a solution to this problem. Our course project worked on building a fast, good quality global router, focusing on the development of a parallel global router using the Bellman-Ford algorithm and OpenMP.

2 Problem Statement

The chip area is partitioned into a regular, coarse(relative to the wire spacing) grid, with multiple grid wires being able to pass through each grid cell boundary. Then the paths taken by the wires through these course grid cells, from the grid cell in which the source pin is located, to the grid cell in which the target pin is located is found. This is done by trying to minimize the wire length, congestion and other design rule constraints. This step is known as global routing. Then all connections inside a given grid cell are made, having fixed how many wires and vias flow into and out of a grid cell. Any violations which global routing could not remove are finally fixed. This step is called detailed routing. This project focuses specifically on the global routing stage.

2.1 Simplifying assumptions taken.

- The chip is assumed to have only 2 layers, in any case, many techniques for multi-layer routing relies on projecting the pins and nets onto a single layer
- All nets are assumed to be 2 pin nets, larger nets can be decomposed into 2 pin nets by adding a few extra branch points
- The capacity, or number of wires a grid cell can allow to pass, is assumed constant throughout the chip, and is taken to be above 5 atleast, else the grid is not course enough, and a cost minimization approach may not be as effective as an obstacle avoidance approach. It has been observed to happen with test cases with very low capacity, that the route quality deteriorates.
- The nets are assumed to be routable or nearly routable and not too dense or too sparse. For this, good placement of blocks within the chip is necessary in the earlier stages of physical design

Version 1.20, May 16, 2024

3 Approach

The problem of global routing is approached using an inherently sequential manner, from which 2 levels of parallelism can be extracted. One is routingmultiple nets in parallel, and te other is the performing the routing of a single net using multiple threads. The router begin with an initial guess of the routing solution, made using L shape connections. It then sequentially rips up nets one by one, and reroutes it using a variant of the Bellmann ford algorithm, as described in the paper "GAMER: GPU Accelerated Maze Routing". Each time a net is ripped up and rerouted, the effect of that net on an overall cost function is only to decrease it and improve the route quality. This is done till there is no more improvement in routing quality, in practice this is seen to happen with only 1-4 iterations. Also, as nets may overlap, this has to be done sequentially, but more parallelism can be extracted by assigning a "bounding box" or an area around a net, only inside which the net's wires may pass, and then partitioning the nets such that any set of nets with bounding boxes that do not mutually overlap, may be routed in parallel. We have chosen to do this using a minimum clique cover algorithm. Bellmann ford was chosen based on a paper titled "GAMER: GPU Accelerated Global Routing", and for its potential for significant parallelism. This technique finds a local minima, for the routing problem. A method of improving the solution quality, which this project skips over, is to use probabilistic techniques to find an optimum initial condition from where to begin the search for a loccal optimum routing. This process is called pattern routing. These algorithms, strategies and the flow of the routing steps is explained next.

4 Algorithms and Parallelization Strategy

Parallelism happens in 2 levels. First, nets are enclosed in a "bounding box" whose size is 1.5 times that of the rectangle whose corners are the two points to be connected by a path. Then the nets are split into multiple batches, each batch consisting of a set of nets whose bounding boxes don't overlap. This enables multiple nets to be routed in parallel. Partitioning nets into batches is also NP-Hard, it is done sequentially, and once at the beginning. It is done using a greedy "minimal clique cover" algorithm that approximates the best division of nets into batches or "cliques", for which the total number of batches is minimum.

Algorithm 1 Minimal Clique Cover Algorithm for VLSI Net Batching

```
procedure MinimalCliqueCover(nets, NUM_THREADS)
    N ← length of nets
    Initialize random seed
    Randomly shuffle nets
    clique ← array of zeros with length N
    batch_count ← empty list
    id ← 1
    for $i$ ← 0 to $N - 1$ do
        if clique[$i$] = 0 then
            clique[$i$] ← id
            count ← 1
            for $j$ ← $i + 1$ to $N - 1$ do
                if clique[$j$] = 0 then
                    clique[$j$] ← id
                    for $l$ ← $i$ to $j - 1$ do
                        if clique[$l$] = id and (nets[$l$], nets[$j$]) then checkRectangleIntersection
                            clique[$j$] ← 0
                            break
                        end if
                    end for
                end if
                if clique[$j$] ≠ 0 then
                    count ← count + 1
                end if
            end for
            batch_count.append(count)
            id ← id + 1
        end if
    end for
    Form batches based on cliques
    Perform further processing on batches
end procedure
```

Algorithm 2 Bellman-Ford Routing

```
for each net in nets do
    Initialize distance and direction for all grid points to infinity and null, respectively
    Set distance to source of net to 0
    repeat
        $flag$ ← false
        for each edge $(u, v)$ in grid G do
            if distance to $v$ > distance to $u$ + weight of $(u, v)$ then
                Update distance to $v$
                Update direction to $v$
                $flag$ ← true
            end if
        end for
        Perform via updates if applicable
    until not $flag$
    BacktrackAndUpdateRoutes(net)
end for

procedure BacktrackAndUpdateRoutes(net)
    Start from destination of net
    while not at source of net do
        Move to next cell in path based on direction array
        Update route for net
    end while
end procedure
```

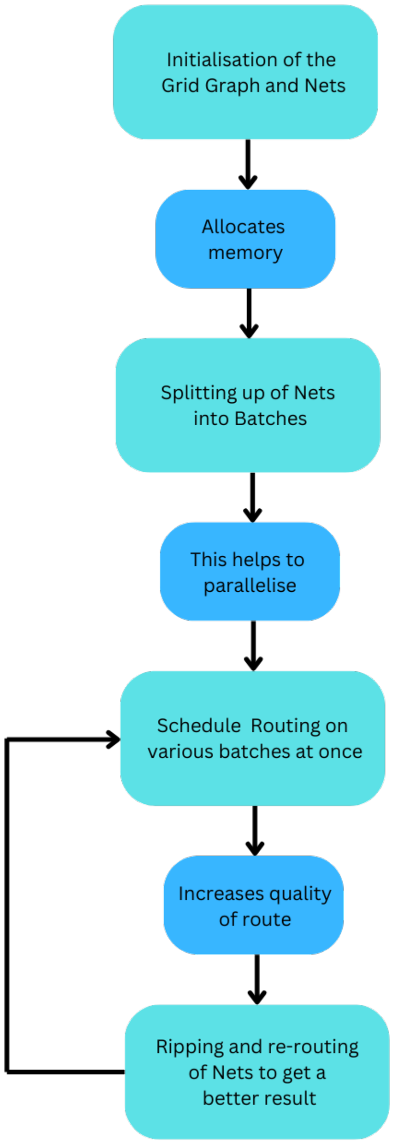
Initialisation of the Grid Graph and Nets
Allocates memory
Splitting up of Nets into Batches
This helps to parallelise
Schedule Routing on various batches at once
Increases quality of route
Ripping and re-routing of Nets to get a better result

Fig. 1 Flowchart explaining the Algorithm

This algorithm works as follows- The full netlist is traversed and each net is checked with all other batches, to search for compatability. Then it is either put into an existing batch if possible, else it is put into a new batch made from that net. Then each net is routed using multiple threads using a variant of Bellmann ford as outlined in the paper "GAMER: GPU Accelerated Maze Routing". The Bellman-Ford algorithm is a graph-based algorithm used for determining the shortest paths from a single source node to all other nodes within a weighted graph.

Application in Parallel Routing Parallel Routing involves the computation of routing paths for multiple nets in circuit designs, where each net must be routed without interference from others, efficiently utilizing the available space and resources. The Bellman-Ford algorithm, adapted for such use, allows simultaneous routing of multiple nets by segmenting the task across different processors or threads. This parallel execution is beneficial in speeding up the routing process, which is critical in the design of large-scale integrated circuits. First, for any point within the bounding box of a net, an initial guess is made for the distance of the best path connecting the source cell to that point, (the guess being infinity), as well as a guess for the direction from which the path will enter that point (can be d,u,n,s,l,r or x meaning no direction). Define a sweep operation as follows- traverse across a row or column, in either right or left direction, and update the estimate of shortest distance and direction of entry, if the entering the cell from the direction of traversal is less costly. This can be done in parallel using prefix scans, but it was decided that this is too fine grained to benefit from OpenMP parallelization. This sweep operation can be done in parallel across all rows first, and then all columns, from left to right, right to left, north to south and then south to north. Then, iterate over all the cells and check if entering that cell through a via is better option than entering from a planar direction. This can also be done in parallel across all vias. Now each sweep can be further parallelized, giving a 3rd level of parallelism, but this involves wite fine grained parallelism, and it was decided that OpenMP is not most suited for that. Finally, the optimum path is found by backtracking from destination to source. Along the way, the path is stored.

5 Analysis of Parallel Performance

Next, the analysis of parallel performance is done and performance parameters are plotted- A 700x700 grid cell example is taken, and finally, the circuit is routed so as to avoid a single electrical short or overflow. For this test case, full routing is possible, with a cost of 930000.

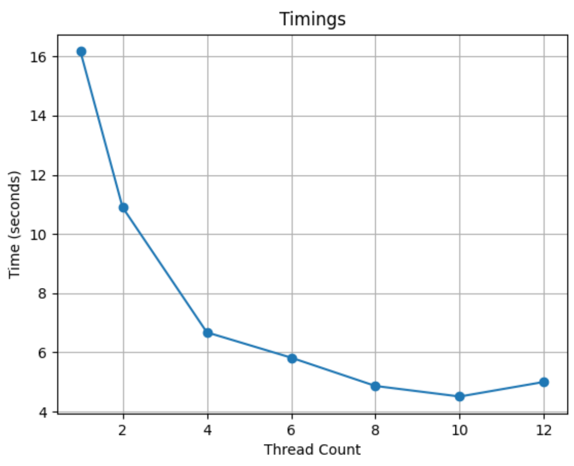

Fig. 2 55000 nets on 700x700 grid, Capacity = 10, Via Cost = 2

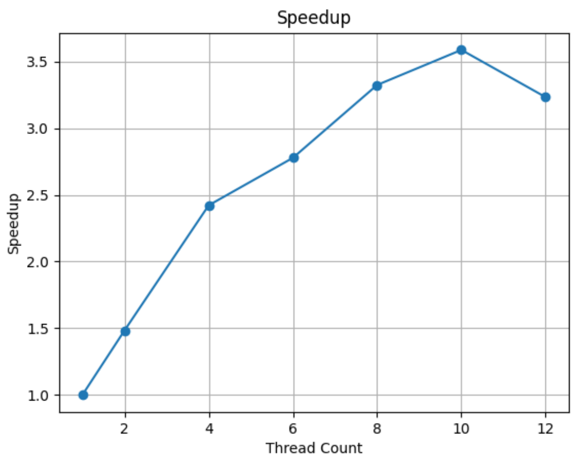

Fig. 3 55000 nets on 700x700 grid, Capacity = 10, Via Cost = 2

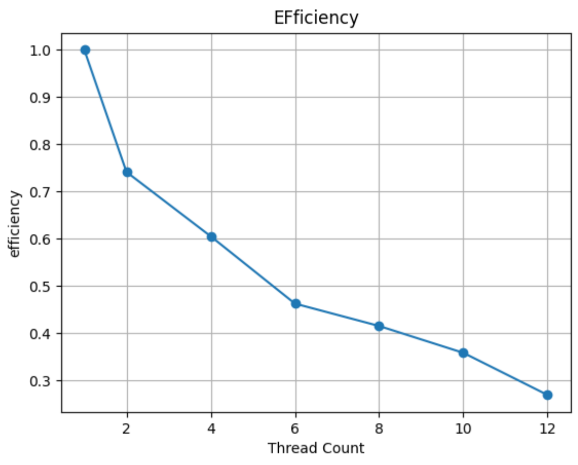

Fig. 4 55000 nets on 700x700 grid, Capacity = 10, Via Cost = 2

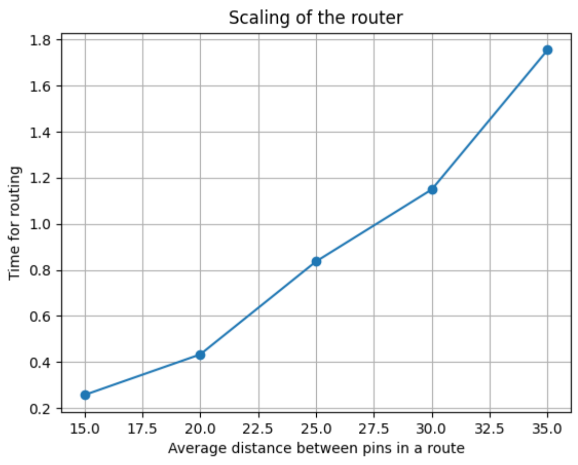

Fig. 5 100x100 grid, length of connections scales up

Few important points-

- In routing a net, there is overhead in the form of system calls to allocate memory for a net, this is due to the dynamically changing size and number of bends present in a net. The solution is to pre-allocate an estimate of the number of bends the net will make, as well as do 2 backtracking steps, one to calculate the size of the net, and then after allocating that much memory with a single system call, backtrack once more to store the list of points at which the path bends, which is the routing solution. This also resolves the issues that can arise due to multiple threads trying to allocate memory at the same time. Thus, the memory allocations are done in serial.
- The time complexity of the routing step s higher as compared to the step of backtracking and storing the output. For a 700x700 grid, a sample timing is 0.0012 seconds for a single succesfull bellman ford routing, while backtracking is around 10 microseconds.

6 More practical observations

3 functions were tried to measure the cost of adding a wire that crosses the edge of a grid cell, to evaluate the quality of routing produced with each, the logistic function performed the best. A test case of 50x50 grid with 1000 nets is summarized in the table.

Table 1 Effect of Different Cost Functions on Routing Quality

Cost Function	Cost	Shorts
Linear	63,553	47
Exponential	66,066	47
1 + Logistic	49,434	47

7 Time Complexity

The minimal clique cover algorithm presented has a time complexity of $O(N^2)$. The "maze routing" phase has a time complexity of roughly the number of routes, $y \cdot l^4$, where l is the average size of a net. Bellman-Ford has a time complexity of $|V|^2$, where V is the number of vertices. This increase in time complexity has to be weighed against the speedups that parallelism brings, and the fact that routing an individual net does not take many iterations in practice, and converges fast.

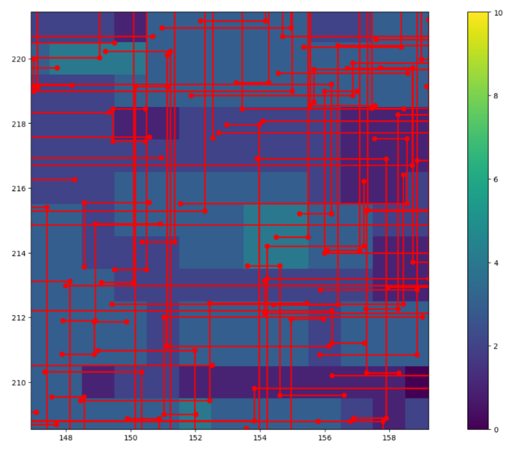

Fig. 6 400 nets in 20x20 grid

The quality of the 300x300 grid routing is evaluated as - Shorts = 0 cost = 424688 And the quality of the 20x20 route is- Shorts = 2 cost = 2428

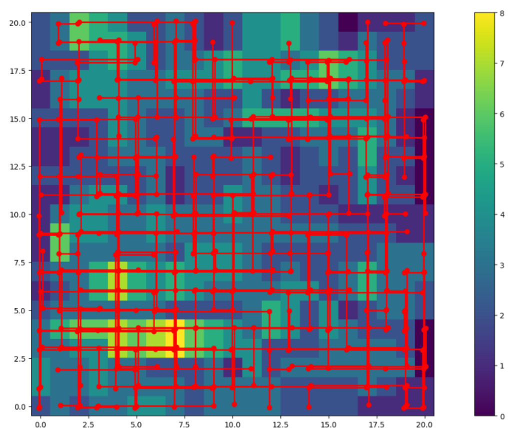

Fig. 7 200 nets on 20x20 grid, this is an example of too few routes

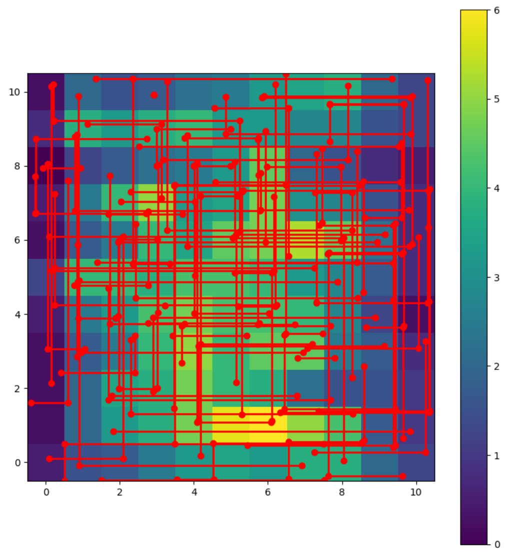

Fig. 8 1000 nets on 20x20 grid, this is an example of too many routes

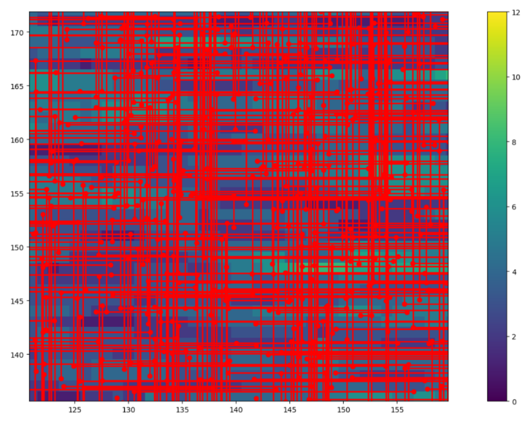

Fig. 9 12000 nets on 300x300 grid, zoomed in

The routing test cases have been made through randomly generating netlists. A minimum and maximum distance between start and endpoint of 2-pin nets was imposed to simulate chip-like netlists.

8 Replicate the Results

The source code for the project discussed above is available on GitHub. It includes all the necessary files to replicate the results and further explore the functionalities of the parallel global VLSI router. You can access the project repository by following this link: GitHub Repository: Parallel Global VLSI Router.

9 Conclusions

This project has developed a global router capable of routing 2 pin nets, as well as demonstrated the effectiveness of OpenMP in speeding up the challenge global routing. Among the other takeaways include-

(1) the low number of iterations of maze routing required to produce a converging global routing solution
(2) The Bellmann-ford algorithm is a feasible way to bring parallelism to global routing

Nomenclature

Net =	a set of points that have to be connected together
Capacity =	The maximum number of wires that can pass through the edge if a grid cell
Grid Graph =	A weighted graph whose nodes are the cells of a cartesian grid, and whose edges represent the capacity of the physical edge that is shared by those two nodes

References

- S. X. Wei, C. K. Koh, and P. H. Madden, "Pattern Routing: Use and Theory for Increasing Predictability and Avoiding Coupling," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 7, July 2002.
- "GAMER: GPU Accelerated Maze Routing"
- Routing Techniques in VLSI Systems Design, ShanghaiTech University.
- Naveed Sherwani, *Algorithms for VLSI Physical Design Automation*. Available online or at major academic libraries.
- FastGR: Global Routing on CPU-GPU with Heterogeneous Task Graph Scheduler

List of Figures

List of Tables