Integrated VLSI Layout Compaction and Wire Balancing on a Shared Memory Multiprocessor: Evaluation of a Parallel Algorithm

Raghu P. Chalasani[‡], K. Thulasiraman[‡‡] and M.A. Comeau[‡]
[‡]Dept. of Electrical & Computer Engineering, Concordia University, Montreal, Canada
[‡‡]Dept. of Computer Science, University of Oklahoma, Norman, OK 73019-0631

Abstract

We first present a unified formulation to three problems in VLSI physical design: Layout compaction, Wire balancing and Integrated layout compaction and wire balancing problems. The aim of layout compaction is to achieve minimum chip width. Whereas wire balancing seeks to achieve minimum total wire length, integrated layout compaction and wire balancing seeks to minimize wire length maintaining the chip width at the optimum value. Our formulation is in terms of the dual transshipment problem. We then review our recent work on a parallel algorithm for the dual transshipment problem. We show how this algorithm called Modified Network Dual Simplex Method provides a unified approach to solve the three problems mentioned above and present experimental results. Our implementations have been on the BBN Butterfly machine. We draw attention to certain rather unusual results and argue that if the MNDS method is used, then integrated layout compaction and wire balancing will achieve minimum chip width and a total wire length close to the optimum achieved by the wire balancing algorithm.

1. Introduction

The rapidly increasing complexity of VLSI circuits puts pressure on VLSI CAD tools from two directions. First, the amount of data to be processed rises dramatically. Second, users have come to rely on their tools more heavily and demand higher performance. This demand for speed up of the VLSI physical design process calls for an application of parallel computing technology. In recent years, there has been considerable interest in research in the application of parallel computing technology to VLSI CAD. Recent reviews of this research may be found in [1], [2], [8].

Layout compaction is an important final step in VLSI physical design process. Compaction is very critical for full-custom layouts, especially for high performance designs. In *layout compaction* one starts with an initial layout and seeks to achieve a final mask layout (without changing the topology) which has minimum chip area and is consistent with the design rules. Invariance of network topology is required in order not to render the previous steps of placement and routing obsolete. At the end of layout compaction relative positions of all the circuit elements will be available. Changing the positions of these elements (to be precise, those elements which lie on a longest path between the chip boundaries) will result in increased chip width. But the positions of the others could be varied without causing design rule violations and yet maintaining compacted chip width. Complete details of different types of compaction and their relative advantages and disadvantages may be found in [6], [7], [11]. In the *wire balancing* problem, one seeks to achieve minimum overall wire length by adjusting the positions of the elements without violating any design rules. The aim of *integrated layout compaction and wire balancing* is to minimize total wire length without increasing the area of the compacted layout. In other words, integrated layout compaction and wire balancing achieves minimum total wire length by adjusting the positions of only those elements which do not lie on the longest paths mentioned above.

2. The Constraint Graph Approach

The constraint graph approach [10] to the layout compaction and wire balancing problem proceeds as follows. From the initial layout a graph G = (V, E), called the *constraint graph*, is constructed. We assume that the layout is Manhattan, i.e., edges of circuit elements are either horizontal or vertical. Each node of G represents a circuit element or a group of circuit elements that are physically connected. Each node v_i is associated with a variable y_i representing the position of the corresponding circuit element. In the following the circuit element corresponding to node v_i will also be referred to as v_i. In G, there is an edge between two nodes, if there is a design rule constraint between the corresponding elements. Each edge in this graph is

0-8186-6507/94 $4.00 © 1994 IEEE

associated with a real number, called the *token* of the edge, which captures the minimum spacing constraint between the nodes of the corresponding edge. There are three types of constraints: *minimum*, *maximum* and *equality* constraints.

A minimum constraint of the type $y_j - y_i \geq a$ states that v_i is to the left of v_j and there is a minimum spacing requirement of 'a' units between them. This constraint can also be stated as $y_i - y_j \leq -a$ and is represented in G by an edge (v_j, v_i) directed from v_j to v_i with an associated token m_{ji} of value $-a$.

A maximum constraint of the form $y_j - y_i \leq b$ or equivalently $y_i - y_j \geq -b$ is represented by an edge (v_i, v_j) directed from v_i to v_j with token m_{ij} of value "b".

An equality constraint can be regarded as a pair of minimum and maximum constraints. Thus an equality constraint will be represented by a pair of oppositely directed edges both with zero token.

Two special nodes, called the source (v_s) and the sink (v_t), are used to represent the right most boundary and the left most boundary of the layout, respectively. Circuit elements which correspond to nodes with no outgoing edges could be placed at the left boundary, and so we add to G edges directed from each one of these nodes to the sink v_t. These edges have zero token. For a similar reason, we add to G zero-token edges directed from the source v_s to the nodes with no incoming edges. The additional edges so added ensure that the circuit elements will not be moved beyond the left and the right boundaries. These edges play a key role in the wire balancing phase.

Besides tokens, edges are also associated with *weights* to indicate the relative costs of wires. We derive node weights from edge weights as follows. Let w_{ij} denote the weight of the edge (v_i, v_j) directed from v_i to v_j. Then the weight w_i of node v_i is given

$$w_i = \sum_j w_{ij} - \sum_j w_{ji}$$

Here we assume that $w_{ij} = 0$ if there is no wire between v_i and v_j. Thus a negative node weight indicates that moving the node to the right will decrease the overall wire length and a positive node weight indicates that moving a node to the right will increase the overall wire length.

3. Layout Compaction

Let Y denote the vector of y_i's, M^t the vector of m_{ij}'s and W, the vector of node weights. Let A be the incidence matrix of G. Then in layout compaction we seek to obtain a $Y \geq 0$ such that

$$A^t Y \geq -M^t$$

and that the difference between the largest and the smallest y_i's is as small as possible. Thus we can formulate this problem as an LP problem as follows.

Minimize: $y_s - y_t$

subject to

$$A^t Y \geq -M^t$$
$$Y \geq 0.$$

Assuming that the position of the sink is at zero coordinate, the above formulation can be written as

Minimize: y_s

subject to

$$A^t Y \geq -M^t$$
$$Y \geq 0. \qquad (1)$$

The works in [4], [5] present an algorithm to obtain a feasible solution of the constraints in (1). This algorithm called FEASIBLE, in fact solves the above layout compaction problem if there are no negative-token directed circuits. (A directed circuit has negative token if the sum of the tokens of all edges in the circuit is negative.) In other words, the y_i values obtained by algorithm FEASIBLE will in fact result in minimum chip width. If we assume that there are no negative-token directed circuits, then the y_i-values obtained at the end of FEASIBLE represent the positions of the different circuit elements. Each node v_i with $y_i = 0$ will be at the left boundary and each v_i with the maximum y_i will be at the right boundary. It can be shown that the maximum y_i-value in fact is the length of the most negative token directed path in G from the source to the sink. In traditional approaches, such a path in fact corresponds to a longest path from the sink v_t to the source v_s.

4. Wire Balancing

The wire balancing problem where we seek to find a Y which minimizes $\sum_i w_i y_i$ can be formulated as the following linear programming problem.

Minimize: $W^t Y$

subject to

$$A^t Y \geq -M^t$$

$$Y \geq 0. \tag{2}$$

The above linear programming problem is the dual of the transshipment problem discussed in detail in [3], and is therefore called the *dual transshipment problem* (DTP).

Note that layout compaction (1) is a special case of the wire balancing problem. Algorithm FEASIBLE determines y_i's which satisfy $A^t Y \geq -M^t$ and so this algorithm determines a feasible solution for both (1) and (2). Thus algorithm FEASIBLE serves as a link between the layout compaction and wire balancing problems.

5. Integrated VLSI Layout Compaction And Wire Balancing

As Algorithm FEASIBLE progresses, the value $y_i - y_j + m_{ij}$ of each edge is modified. We shall call $y_i - y_j + m_{ij}$ as the *residual token* of edge (i, j). When all the residual edge tokens are non-negative (that is $y_i - y_j + m_{ij} \geq 0$), the algorithm terminates. Let $y_i = \lambda_i$ at the end of Algorithm FEASIBLE. Interestingly, at termination the residual tokens of the edges on the most negative token directed path in G from v_s to v_t will all be zero. In other words, for every (i, j) on such paths $y_i - y_j + m_{ij} = 0$. So by a simple traversal of G starting from v_s, we can identify all the nodes on these most negative token paths. Let S_l denote the set of these nodes.

Our aim in integrated compaction and wire balancing is to keep each node $v_i \in S_l$ at $y_i = \lambda_i$ and adjust the y-values of those nodes not in S_l so that total wire length can be minimized without increasing the area of the layout. Thus we have the following formulation of the integrated compaction and wire balancing problem.

Minimize: $W^t Y$

subject to

$$A^t Y \geq -M^t$$

$$Y \geq 0. \tag{3}$$

$$y_i = \lambda_i, \text{ for } v_i \in S_l, \tag{4}$$

$$y_i \geq 0, \text{ for all other } v_i.$$

But, as seen above constraint (4) can be replaced by:

$$y_i - y_j + m_{ij} = 0 \tag{5}$$

for every edge (v_i, v_j) on a most negative token path from v_s to v_t.

Now it is a simple matter to represent each of the above equality constraints by adding to G oppositely oriented edges with tokens m_{ij} and m_{ij} between v_i and v_j , and between v_j and v_i and then proceed with the solution of the resulting optimization problem. However, such an approach will result in a significant increase in the size of the graph. In fact, we can considerably reduce the size of the graph as discussed below.

Constraint (5) suggests that in any new solution of the wire balancing problem,

$$y_i = \lambda_i + k, \text{ for } v_i \in S_l$$

where k is a fixed constant. We can take advantage of this property as follows.

We construct a new graph G′ by replacing the nodes in S_l by a single new node, say, v_l, and then removing all the edges connecting the nodes in S_l. Consider now an edge (v_i, v_j) in G. If $v_i \notin S_l$ and $v_j \notin S_l$, G′ will have an edge (v_i, v_j) with token m_{ij}. If $v_i \in S_l$, then G′ will have an edge (v_l, v_j) with token $(\lambda_i + m_{ij})$. If $v_j \in S_l$, then G′ will have an edge (v_i, v_l) with token $(-\lambda_j + m_{ij})$. If after this contruction, there are parallel edges, in G′, between two nodes then we can remove all of them except the one with the smallest token. Let the new graph be denoted by G″. In G″ the weight of every node $v_i \notin S_l$ will be equal to w_i. For the new node v_l representing S_l, the weight w_l will be given by

$$w_l = \sum_{v_i \in S_l} w_i .$$

We now have the following formulation of the integrated compaction and wire balancing problem, where w_i, y_i, m_{ij} refer to the quantities defined for G''.

$$Minimize \sum_i w_i y_i$$

subject to

$$A^t Y \geq -M^t$$
$$Y \geq 0, \qquad (6)$$

where A is the incidence matrix of G''.

Note that the y_i-values given by algorithm FEASIBLE, define a feasible solution of (2). In this feasible solution $y_l = 0$, and $y_i = \lambda_i$, $i \neq l$ where the node v_l represents graph S_l of nodes and λ_i is the value of y_i given by algorithm FEASIBLE.

6. Modified Network Dual Simplex Method: A Review

The simplex method of linear programming when applied to solve the dual transshipment problem (DTP) is called the *network dual simplex method.* A detailed presentation of the simplex method and its applications to network optimization problems may be found in [3].

The following are the main steps in the network dual simplex method:

1. Construct an initial basic feasible solution Y_0, if it exists. This is achieved by constructing an auxiliary graph and applying the dual simplex method on this graph. This step detects infeasibility of the DTP, whenever this is the case.
2. Perform a pivot operation, if this is permissible. This results in an improved value for the objective W^tY.
3. Check the new basic feasible solution for optimality. (The solution is optimum, if the corresponding basic feasible solution permits no pivot operation.) If the solution is optimum, then the algorithm terminates. Otherwise, repeat step (2) starting from the current basic feasible solution.

As can be seen from the above, the network dual simplex method moves from one basic solution to another, performing one pivot operation at a time. Any parallel implementation of this method will focus on the parallelisation of the pivot operation. But during a pivot operation only a small subgraph of the given graph will be involved. Thus, such an approach does not offer much scope for achieving good speed up.

The above difficulty can be resolved by permitting concurrent pivot operations. But at the end of concurrent pivots, the resulting solution may not be basic. Thus, we need an algorithm to generate a basic feasible solution from a given feasible solution. But while doing so, we should ensure that the objective value W^tY never increases. In [9], we have described a new method called the Modified Network Dual Simplex (MNDS) which addresses these concerns.

An outline of MNDS is as follows.

Modified Network Dual Simplex (MNDS) Method

1. Test feasibility of the DTP. (Apply Algorithm FEASIBLE.) If feasible, construct a feasible solution Y; otherwise, the algorithm terminates.
   (Steps 2 - 4 construct, from a given feasible solution Y, a basic feasible solution Y' with $W^tY' \leq W^tY$.)
2. Construct the subgraph of the given graph which contains all the edges with zero residual tokens; determine the connected components of this subgraph. Coalesce the nodes in each component and construct a condensed graph G'.
3. Perform shortest path computations on the condensed graph and update the y_i-values.
4. Repeat steps (2) and (3) until all the nodes coalesce into a single node. (At this point, the edges with zero residual tokens will span all the nodes of the original graph; a spanning tree with all its edges having zero residual tokens will define a basic feasible solution.)
5. Check the optimality of the basic feasible solution obtained in step 2. If it is optimal, the algorithm terminates; otherwise, perform concurrent pivot operations resulting in a new feasible solution Y.
6. Repeat steps 2 - 5. ❑

Note that MNDS does not require constructing an auxiliary graph to construct a basic feasible solution.

This is an attractive feature from the point of view of parallel/distributed implementation.

7. Application of MNDS to Layout Compaction, Wire Balancing and Integrated Layout Compaction and Wire Balancing Problems

We now show how the MNDS method provides a unified approach to solving the layout compaction, wire balancing and integrated layout compaction and wire balancing problems.

To achieve **layout compaction**, apply Algorithm FEASIBLE to the constraint graph (Step 1 of MNDS described in Section 6).

To achieve **wire balancing**, apply the MNDS method on the constraint graph with weights assigned to nodes. (See Section 2.)

To achieve **integrated layout compaction and wire balancing**, adapt MNDS as follows:

1. Apply Algorithm FEASIBLE (Step 1 of MNDS) and obtain the feasible solution Y.
2. Coalesce all the nodes which lie on the longest paths from the sink to the source and construct the condensed graph G'' (described in Section 5). Using Y, obtain the feasible solution Y' for G''.
3. Starting from the feasible solution Y', construct a basic feasible solution for G''. (See Steps 2 - 4 of MNDS.)
4. If the basic feasible solution obtained at step 3 is not optimal, perform concurrent pivot operations which result in a new feasible solution Y'.
5. Repeat steps 3 and 4 until the solution is optimal. ❑

Note that steps 3 - 5 above correspond to applying MNDS on G''. (See Section 5.)

8. Experimental Evaluation and Conclusion

We have applied our algorithm for integrated layout compaction and wire balancing described in Section 7 on large industrial designs received from Cadence Design Systems. Our implementations have been on the BBN Butterfly shared memory machine. The execution times and speed-ups for different numbers of processors are given in Tables 1 - 5.

For these designs, we have also summarised in Table 6, the savings in layout area and total wire length achieved by wire balancing as well as by integrated layout compaction and wire balancing. The corresponding algorithms are as outlined in Section 7.

Results of Table 6 lead to the following two important observations.

Observation 1

For all the graphs, in particular large graphs, the chip width at the end of wire balancing (MNDS algorithm) is not significantly larger than the optimum chip width obtained by Algorithm FEASIBLE. (Step 1 of MNDS).

Observation 2

For all the graphs, in particular large graphs, the total wire length at the end of integrated layout compaction and wire balancing is not significantly larger than the optimum wire length obtained by wire balancing.

The above, rather unusual, results can be explained as follows.

Consider the MNDS algorithm of Section 6. We shall refer to one application of Steps 2 - 5 of this algorithm as one pass of MNDS. Similarly, one application of the corresponding steps 3 and 4 of the integrated layout compaction and wire balancing algorithm will be referred to as one pass of this algorithm.

Now, we have the following.

1. In several cases that we have considered, MNDS requires no more than 5 passes. Also, a significantly large percentage of the wire length minimization occurs in the first pass itself.
2. In their first steps, both the wire balancing (MNDS) and the integrated layout compaction and wire balancing algorithms place the nodes at positions which give minimum chip width.
3. Step 2 of both these algorithms perform identical computations (determining the condensed graph).
4. Also, in the first pass, steps 3 and 4 of the wire balancing algorithm and step 3 of the integrated layout compaction and wire balancing algorithm perform identical operations.
5. Step 5 of MNDS and Step 4 of the integrated layout compaction and wire balancing algorithm differ in the first pass. Whereas pivots performed by MNDS may change the y_i values which may result in

larger chip width, the pivots performed by the integrated layout compaction and wire balancing algorithm do not change the chip width since these operations are on G''.

Thus, both these algorithms start with the same initial solution (optimum chip width) and do not significantly differ in their first passes. Combining this with the fact that MNDS achieves a significant amount of optimization in the first pass itself, we have strong reason to infer that:

- **The integrated layout compaction and wire balancing algorithm of Section 7 achieves optimum chip width and a total wire length which is not significantly different from the optimum wire length achieved by the wire balancing algorithm.**
- **Since a good part of the integrated layout compaction and wire balancing algorithm operates on a condensed graph, usually it takes about 30% less time than the wire balancing algorithm. As a result, one would prefer to use the integrated layout compaction and wire balancing algorithm to achieve optimum chip width and a total wire length close to optimum.**

We should add that this important inference may not be true if the traditional dual simplex method is used to solve wire balancing problem.

Acknowledgements

We would like to thank Mr. Ravi Varadarajan of Cadence Design Systems, Inc. for providing us the test graphs for layout compaction and wire balancing problems. We would also like to thank Dr. C. Tropper of McGill University for providing access to the BBN Butterfly parallel computer.

9. References

[1] Agarwal, P., *"VLSI Computer Aided Design Using Multiprocessing Systems,"* Technical Report, AT&T, Bell Labs, Murray Hill, N.J., 1991.

[2] Banerjee, P., "The Use of Parallel Processing for VLSI CAD Applications: A Tutorial," *Proc. Intl. Conf. on CAD*, ICCAD-88, 1988.

[3] Chvatal, V., *Linear Programming*, Freeman Company, Potomac, Maryland, 1983.

[4] Comeau, M.A. and K. Thulasiraman, "Structure of the Submarking Reachability Problem and Network Programming," *IEEE Trans. Circuits and Systems*, Vol. CAS-35, 89-100, 1988.

[5] Comeau, M.A., K. Thulasiraman and K.B. Lakshmanan, "An Efficient Asynchronous Distributed Protocol to Test Feasibility of the Dual Transshipment Problem," Proc. Allerton Conf. on Communication, Control and Computer, Univ. of Illinois, Urbana, 634-640, Sept. 1987.

[6] Lengauer, T., *Combinatorial Algorithms for Integrated Circuit Layout*, John Wiley & Sons, England, 1990.

[7] Sherwani, N., *Algorithms for VLSI Physical Design Automation*, Kluwer Academic Publishers, MA, 1993.

[8] Tham, K.Y., "Parallel Processing for CAD Applications," *IEEE Design Test*, 13-17, October 1987.

[9] Thulasiraman, K., R.P. Chalasani and M.A. Comeau, "Parallel Network Dual Simplex Method on a Shared Memory Multiprocessor," *Proc. 5th IEEE Symp. on Parallel and Distributed Processing*, 408-415, 1993.

[10] Yoshimura, T., "A Graph Theoretical Compaction Algorithm," *Proceedings of the 1985 IEEE International Symposium on Circuits and Systems*, 1455-1458, 1985.

[11] Wolf, W.H. and A.E. Dunlop, "Symbolic Layout and Compaction," in "Physical Design Automation of VLSI Systems," Edited by Preas, B. & M. Lorenzetti, 1988.

No of Processors	Time in Seconds	Parallel Speed up
1	16.77	1.0
2	10.01	1.7
3	8.04	2.1
4	6.58	2.5
5	5.99	2.8
6	5.84	2.9
7	5.35	3.1
8	5.06	3.3
9	4.93	3.4
10	4.92	3.4

Table 1: Graph with 149 nodes

No of Processors	Time in Seconds	Parallel Speed up
1	9.94	1.0
2	5.80	1.7
3	4.32	2.3
4	3.69	2.7
5	3.39	2.9
6	3.18	3.1
7	2.65	3.8
8	2.81	3.5
9	2.22	4.5
10	2.10	4.7

Table 2: Graph with 157 nodes

No of Processors	Time in Seconds	Parallel Speed up
1	6246.28	1.0
2	2940.98	2.1
3	2099.63	3.0
4	1654.00	3.8
5	1206.52	5.2
6	1169.54	5.3
7	1131.48	5.5
8	949.17	6.6
9	793.67	7.9
10	901.98	6.9
11	663.51	9.4
12	609.81	10.2
13	653.85	9.6
14	741.68	8.4
15	724.64	8.6

Table 3: Graph with 1265 nodes

No of Processors	Time in Seconds	Parallel Speed up
1	10013.66	1.0
2	4882.85	2.1
3	3055.94	3.3
4	2512.66	4.0
5	2353.02	4.3
6	1922.75	5.2
7	1731.23	5.8
8	1567.94	6.4
9	1415.48	7.1
10	1487.65	6.7
11	1320.82	7.6
12	1389.09	7.2
13	1175.26	8.5
14	1176.46	8.5
15	1307.63	7.7

Table 4: Graph with 1612 nodes

No of Processors	Time in Seconds	Parallel Speed up
1	33738.35	1.0
2	16606.27	2.0
3	11830.75	2.9
4	9313.02	3.6
5	8157.26	4.1
6	6807.90	5.0
7	5655.07	6.0
8	5126.10	6.6
9	5162.44	6.5
10	4557.98	7.4
11	4453.89	7.6
12	4217.13	8.0
13	4089.29	8.3
14	4075.00	8.3
15	3881.05	8.7

Table 5: Graph with 2087 nodes

	No of Nodes	Objective x 10^8	Improvement %	Chip Width	Improvement %
Layout Compaction	149	35.65788	-	49600	-
Wire Balancing		33.35350	6.46	49600	0
Integrated Layout Compaction & Wire Balancing		33.35350	6.46	49600	0
Layout Compaction	157	8.270312	-	123800	-
Wire Balancing		6.396859	22.65	128300	–3.63
Integrated Layout Compaction & Wire Balancing		6.654659	19.54	123800	0
Layout Compaction	1265	11348.29	-	331450	-
Wire Balancing		5636.508	50.33	370250	–11.71
Integrated Layout Compaction & Wire Balancing		6239.136	45.02	331450	0
Layout Compaction	1612	52485.97	-	321000	-
Wire Balancing		28140.84	46.38	321000	0
Integrated Layout Compaction & Wire Balancing		28140.84	46.38	321000	0
Layout Compaction	2087	378.7866	-	1016000	-
Wire Balancing		307.5949	18.79	1025500	–0.94
Integrated Layout Compaction & Wire Balancing		307.7869	18.74	1016000	0

Table 6: Savings in Chip Width and Total Wire Length