TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 07/17 pp41-50
Volume 16, Number 1, February 2011

Partition-Based Global Placement Considering Wire-Density Uniformity for CMP Variations*

DONG Changdao (董昌道), ZHOU Qiang (周 强)**, CAI Yici (蔡懿慈), LIU Dawei (刘大为)

Tsinghua National Laboratory for Information Science and Technology, Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China

Abstract: This paper presents a multilevel hypergraph partitioning method that balances constraints on not only the cell area but also the wire weight with a partition-based global placement algorithm that maximizes the wire density uniformity to control chemical-mechanical polishing (CMP) variations. The multilevel partitioning alternately uses two FM variants in the refinement stage to give a more uniform wire distribution. The global placement is based on a top-down recursive bisection framework. The partitioning algorithm is used in the bisectioning to impact the wire density uniformity. Tests show that, with a 10% constraint, the partitioning produces solutions with more balanced edge weights that are 837% better than from hMetis, 1039.1% better than MLPart, and 762.9% better than FM in terms of imbalance proportion and that this global placement algorithm improves ROOSTER with a more uniform wire distribution by 3.1% on average with an increased wire length of only 3.0%.

Key words: partitioning; placement; chemical-mechanical polishing (CMP); design for manufacturing (DFM); wire-density

Introduction

As nanometer technologies advance, the chemical-mechanical polishing (CMP) step in the copper metalization (Damascene) process may cause some loss of interconnections. A non-uniform feature density distribution in one interconnect layer causes the CMP to over or under polish, so thickness variations may accumulate in each layer and finally be very significant[1,2]. Dummy features are usually filled into layouts to restrict the variations in each layer[3]. However, dummy features may degrade interconnect performance and cause mask data explosion[4]. Therefore, to minimize the side effects of aggressive post-layout dummy filling, the wire density uniformity must be considered in the early design stages such as for the global placement.

Several CMP-aware methods have been developed for the early design stages[5]. Cho et al.[6] proposed a predictive copper (Cu) CMP model to evaluate topography variations to guide CMP-aware global routing. Chen et al.[7] proposed a wire density driven full-chip routing which considers not only the wire density inside a routing gcell but also its gradient between routing gcells. Chen et al.[8] proposed a metal-density driven placement algorithm based on an analytical framework which improves the metal-density uniformity by 3% while increasing the wire length by 4% compared with a cell-density driven placement. Chen et al.[8] described a probabilistic routing model to estimate the wire density and the metal density and thickness using by the CMP model proposed in Cho et al.[6]

Received: 2009-04-27; revised: 2010-12-08

* Supported by the National Natural Science Foundation of China (Nos. 60876026 and 60833004)

** To whom correspondence should be addressed.
E-mail: zhouqiang@tsinghua.edu.cn; Tel: 86-10-62785564

The wire density distribution will determine the layout pattern density[6], which closely correlates to the post-CMP dielectric thickness[4], so it is desirable to minimize the global variation of the wire density[7]. Since placement is a crucial design stage which strongly impacts the subsequent routing results, a uniformly distributed wire density during placement becomes very important for CMP variation control. The wire density is closely related to routing congestion, with many congestion driven placement algorithms proposed to improve routability. However, the wire density is not the same as routing congestion, so routability driven placement may only contribute modestly to the wire distribution uniformity.

ROOSTER[9], a state-of-the-art congestion driven placement algorithm, recursively applies multilevel hypergraph partitioning, such as MLPart[10] and hMetis[11], to bisection large bins containing hundreds or more standard cells during the global placement stage. hMetis[11] is one of the most well-known multilevel partitioning algorithms, which produces solutions with nearly minimized cut sizes. Of the three clustering methods evaluated by Karypis et al.[11], the FM variant using random vertex selection was used in the refinement stage of the multilevel paradigm. MLPart[10] is another leading-edge multilevel partitioning algorithm comparable with hMetis. MLPart employs the PinEC scheme to speed up pin removal during vertice clustering and CLIP[12] to improve the refinement during the initial partitioning and uncoarsening stage. Although the wire distribution between the two sub-bins is largely determined after each bisection, its uniformity is not a significant issue in traditional partitioning methods. MLPart and hMetis often produce solutions with highly unbalanced numbers of edges between the two blocks.

The motivation of this work is to design a multilevel hypergraph partitioning algorithm that balances the wire distribution and then apply this algorithm to the recursive bisection in global placement to generate results with more uniformly distributed wire densities for better CMP variation control. The multilevel partitioning method alternates between an FM variant capable of balancing edge weights between the two blocks and another FM variant capable of minimizing the cut size for tight edge weight constraints in the refinement stage. The first FM variant generates solutions satisfying the edge weight constraint and minimizes the imbalance with a time cost almost linearly related to the number of pins in the circuit. The second FM variant minimizes the cut size for the given edge weight constraint. The inheritance of edge weights by corresponding clusters is also considered in the coarsening stage. The global placement framework uses the wire weight balancing constraint in the partitioning. A multilevel hypergraph partitioning with dual balancing constraints on both the cell area and the wire weight is recursively performed in the global placement. The wire weight is assigned such that it estimates the wire distribution in each placement bin. Tests show that the algorithm is capable of generating solutions with more uniformly distributed wire densities and outperforms ROOSTER in terms of the standard deviation of the horizontal and vertical wire density of routing gcells, with a small increment in the total wire length.

1 Preliminaries

1.1 Hypergraph partitioning with constraints on the wire density balancing weight

Formally, a hypergraph $H=(V, E)$ is defined as a set of vertices V and a set of edges E, where each edge is also called a hyperedge and is a subset of the vertex set V. A vertex is said to be incident to an edge, if $v \in e$. Each vertex and edge has one or more weights associated with it. The hypergraph partitioning is a decomposition of V into two disjoint subsets V_1 and V_2, such that $\cup V_i = V$. Each subset is called one partitioning block. An edge e is said to be cut if $\{v \mid v \in V_1 \cap v \in e\} \neq \varnothing \cap \{v \mid v \in V_2 \cap v \in e\} \neq \varnothing$.

This analysis associates each vertex with a weight $w(v)$ and each edge with two weights, the cut weight, $w_c(e)$, and the wire density balancing weight, $w_b(e)$. Define the cut set $C=\{e \mid e \text{ is cut}\}$ and the cut size of a partitioning $S(C)=\sum_{e \in C} w_c(e)$. Given the set of internal edges of block V_i defined as $I(V_i)=\{e \mid e \notin C \cap (\exists v \text{ s.t. } v \in e \cap v \in V_i)\}$, $i=1, 2$, employ the concept of total wire density balancing weight of block V_i defined as

$$W_E(V_i)=\sum_{e_i \in I(V_i)} w_b(e_i)+\sum_{e_c \in C} w_b(e_c)/2 \qquad (1)$$

Similarly, the total vertex weight of block V_i is defined as $W_V(V_i)=\sum_{v \in V_i} w(v)$. Given the target vertex

weight of the two blocks as $(W_V^{t_1}, W_V^{t_2})$, $W_V^{t_1} + W_V^{t_2} = W_V(V)$, the imbalance proportion is defined as $\delta_{W_V} = \max_{i=1,2} \frac{|W_V(V_i) - W_V^{t_i}|}{W_V^{t_i}}$ and

$$\delta_{W_E} = \max_{i=1,2} \frac{\left| W_E(V_i) - W_E(V) \frac{W_V(V_i)}{W_V(V)} \right|}{W_E(V) \frac{W_V(V_i)}{W_V(V)}} \quad (2)$$

A partitioning satisfies the δ_{CW_V} constraint on vertex weight if $\delta_{W_V} \leqslant \delta_{CW_V}$ and satisfies the δ_{CW_E} constraint on wire density balancing weight if $\delta_{W_E} \leqslant \delta_{CW_E}$.

The hypergraph partitioning problem with dual constraints is defined as follows. Given a hypergraph $H = (V, E)$ with wire density balancing weight constraint δ_{CW_E} and vertex weight constraint δ_{CW_V}, compute a partitioning of V that satisfies both δ_{CW_E} and δ_{CW_V} and minimizes $S(C)$.

1.2 FM heuristic iteration

The FM heuristic iteration is the fundamental algorithm of most modern move-based partitioning methods. It starts with a random initial partition and moves vertices from one block to the other in several passes until the last pass fails to optimize the solution. During each pass, vertices are moved one by one in order with the vertex having the greater gain always preferred to move. After a vertex is moved, it is locked in the pass, with the unlocked vertices called free. The gain of a vertex $g(v)$ is the direct reduction of $S(C)$ if v moves from one block to the other. The methods for the gain calculation and to update the free vertices enable the algorithm to have a time consumption per pass that is linearly related to the size of the hypergraph. The analysis of the critical edges and the gain bucket structure makes this linear time consumption possible.

1.3 Multilevel paradigm

The multilevel paradigm is widely used in modern partitioning algorithms, such as hMetis and MLPart, which are capable of producing nearly minimized cut sizes for a given vertex weight balancing constraint. There are three stages in this paradigm. The first is the coarsening stage, during which the hypergraph size is successively decreased level by level with more tightly connected vertices clustered to one new vertex with a large weight. When the hypergraph size becomes small enough, the initial partitioning is performed, with a solution computed on the top level hypergraph. During the uncoarsening and refinement stage, the solution is successively refined as it is projected to larger graphs level by level.

Other techniques are also employed by modern multilevel algorithms, such as the multi-start and the V-cycle paradigms. The multi-start paradigm selects the best solution from those of the previous stage as the input to next stage. The V-cycle paradigm performs multiple coarsening and uncoarsening phases to seek better solutions.

1.4 Top-down min-cut placement

Top-down placement algorithms decompose a given placement instance into smaller instances by recursively subdividing the placement regions, cutting the netlist and then assigning the modules to subregions. Min-cut placers generally use either bisection or quadrisection to divide the placement area and the netlist. Each placement instance is induced from a bin, which contains a placement region with allowed module locations in which a collection of circuit modules are to be placed. Fixed modules and pins outside a bin that are adjacent to modules inside this bin are called terminals.

ROOSTER[9] is a state-of-the-art congestion driven placement framework based on Capo[13], which employs a Steiner-Tree to better predict the wire length after global routing and performs cut-line shifting guided by the prediction of a probabilistic congestion map[14] to further minimize the peak congestion.

2 Multilevel Partitioning with Dual Constraints (MLDC)

The MLDC algorithm satisfies dual constraints on both the vertex weight and the wire density balancing weight while maintaining the small cut size. Average test results on ISPD-98[15] benchmarks are listed in Table 1 when solutions are constrained to 10% δ_{CW_E} in the present algorithm and 10% δ_{CW_V} in all of the algorithms and the target $W_V^{t_1}$ is equal to $W_V^{t_2}$. In Table 1, $w_b(e)$ is simply set to the degree of edge e in the present algorithm and δ_{PN} is the imbalance on the number of pins in the two blocks. The results show that

Table 1 Comparisons of the present algorithm with hMetis, MLPart, and FM on the IBM01-18 databases solutions are constrained to 10% δ_{CW_E} in the present algorithm and 10% δ_{CW_V} in all of the algorithms.

Number of IBM	Algorithm	δ_{W_E}	δ_{PN}	Cut	Time (s)
01	hMetis	32.403	29.726	238.667	19.780
	MLPart	74.086	68.528	217.250	40.078
	FM	70.182	65.007	261.167	30.072
	Present	8.655	7.695	278.583	46.646
03	hMetis	87.389	87.845	705.167	43.512
	MLPart	87.813	88.316	710.083	99.233
	FM	50.335	50.942	1447.000	66.755
	Present	8.883	7.800	988.500	228.530
05	hMetis	5.757	5.525	1724.500	70.450
	MLPart	5.579	5.010	1730.580	139.488
	FM	6.905	5.585	2467.330	147.823
	Present	6.634	6.311	1725.830	221.013
07	hMetis	32.548	30.371	749.333	110.572
	MLPart	40.222	37.880	756.667	179.623
	FM	38.592	36.393	1049.500	172.508
	Present	8.828	7.084	1012.920	238.989
09	hMetis	13.826	15.415	523.167	108.188
	MLPart	13.978	15.527	532.500	169.458
	FM	19.332	20.307	1280.170	178.416
	Present	1.571	2.191	664.500	259.494
11	hMetis	30.159	31.781	698.833	172.009
	MLPart	31.702	33.291	726.083	164.084
	FM	28.232	28.224	2439.750	375.005
	Present	9.020	9.998	987.833	429.029
13	hMetis	2.938	1.164	884.917	220.097
	MLPart	6.010	4.033	869.417	348.829
	FM	16.695	15.392	1460.580	337.701
	Present	5.477	4.003	914.083	548.614
15	hMetis	44.333	43.845	1907.250	641.516
	MLPart	44.287	43.981	2031.420	788.930
	FM	22.295	22.141	5593.580	671.946
	OUR	7.817	7.817	3042.670	1167.130
17	hMetis	10.985	10.896	2323.920	1158.360
	MLPart	15.619	15.959	2243.830	968.930
	FM	9.102	9.048	3880.170	1165.370
	Present	8.981	8.961	2411.330	1033.410
Avg. ratio	hMetis	8.372	6.156	0.775	0.478
	MLPart	10.391	7.556	0.779	0.618
	FM	7.629	5.377	1.423	0.696
	Present	1.000	1.000	1.000	1.000

hMetis and MLPart do not strongly consider the wire distribution between the two blocks, which is much more balanced in the present algorithm.

2.1 Minimization of edge weight imbalance

Randomly generated initial solutions may satisfy the vertex balancing constraint but violate the edge weight balancing constraint. Thus, an edge weight balancing FM variant (FMEB) was used to minimize the edge weight imbalance with the time cost being almost linearly related to the hypergraph size.

This algorithm minimizes the imbalance between the edge weights between the two partitioning blocks. Given the constraint δ_{c}, define the cost function as

$$\mathrm{Cost(EB)} = |W_E(V_1) - W_E(V_2)| - [W_E(V_1) + W_E(V_2)]\delta_{\mathrm{c}} \tag{3}$$

The algorithm uses the FM heuristics to iteratively move a vertex block to block and to select the vertex with the greatest gain to move given the vertex weight balancing constraint in one pass. The gain reduction in Cost(EB) by moving a vertex is calculated based on the edge weight balancing gain (EBGain) calculation. Algorithm 1 computes the initial EBGains for each free vertex v_{f}.

After each vertex movement, the EBGains are stored in sorted buckets to be updated. All the gains in the EBGain algorithm will change if the block with the greatest edge weight changes. The EBGain update algorithm should then again calculate the gains of all free vertices.

Algorithm 1 Calculate EBGain of v_{f}

$V_{\mathrm{F}} \Leftarrow$ the block containing v_{f}
$V_{\mathrm{T}} \Leftarrow$ the other block
$W_{E,\mathrm{temp}}(V_{\mathrm{F}}) \Leftarrow W_E(V_{\mathrm{F}})$
$W_{E,\mathrm{temp}}(V_{\mathrm{T}}) \Leftarrow W_E(V_{\mathrm{T}})$
For all edge e adjacent to v_{f} **do**
  $N_{\mathrm{F}} \Leftarrow$ number of vertices on e in V_{F}
  $N_{\mathrm{T}} \Leftarrow$ number of vertices on e in V_{T}
  If $N_{\mathrm{T}} = 0$ or $N_{\mathrm{F}} = 1$ **then**
    $W_{E,\mathrm{temp}}(V_{\mathrm{F}}) \Leftarrow W_{E,\mathrm{temp}}(V_{\mathrm{F}}) - w_{\mathrm{b}}(e)/2$
    $W_{E,\mathrm{temp}}(V_{\mathrm{T}}) \Leftarrow W_{E,\mathrm{temp}}(V_{\mathrm{T}}) + w_{\mathrm{b}}(e)/2$
  End if
End for
$\mathrm{EBGain}(v_{\mathrm{f}}) \Leftarrow |W_E(V_{\mathrm{F}}) - W_E(V_{\mathrm{T}})| - |W_{E,\mathrm{temp}}(V_{\mathrm{F}}) - W_{E,\mathrm{temp}}(V_{\mathrm{T}})|$

If the block with the greatest edge weight does not change when a vertex is moved, which is the most

likely situation, only the vertices adjacent to the moved vertex must be updated. The six situations to be considered for vertices in a critical edge are illustrated in Fig. 1.

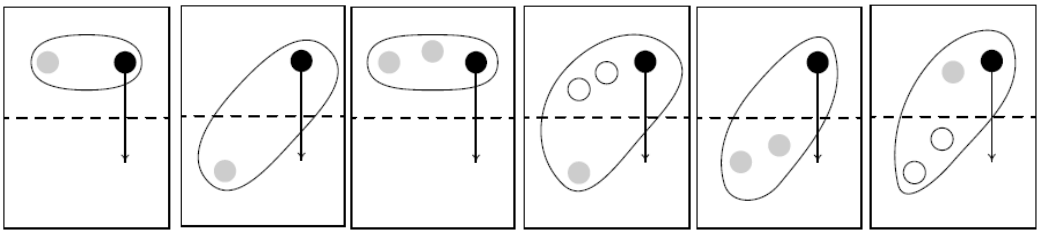


Fig. 1 Situations considered by the EBGain update. Each black vertex is to be moved (v_m) and each grey vertex is a free vertex (v_f) to be considered.

Let the amount of EBGain for free vertex v_f contributed by edge e be GC(v_f, e), and GC(v_f, e) has opposite values depending on whether $W_E(V_F) > W_E(V_T)$. The following analysis assumes that $W_E(V_F) > W_E(V_T)$. In Figs. 1a and 1b, GC(v_f, e) is $w_b(e)$ or $-w_b(e)$ before the movement and remains the same after the movement, so there is no need to update EBGain(v_f). In Figs. 1c and 1d, GC(v_f, e) is $w_b(e)$ or $-w_b(e)$ before the movement and becomes 0 after the movement, so EBGain(v_f) should be updated by $-w_b(e)$ or $w_b(e)$. In Figs. 1e and 1f, GC(v_f, e) is 0 before the movement, and becomes $-w_b(e)$ or $w_b(e)$ after the movement, so EBGain(v_f) should be updated by $-w_b(e)$ or $w_b(e)$. If $W_E(V_F) < W_E(V_T)$, all the EBGain updates should be reversed.

Algorithm 2 updates the EBGains for the free vertices.

Algorithm 2 Update EBGains after movement v_m

$V_F \Leftarrow$ the block containing v_m
$V_T \Leftarrow$ the other block
If the block with the greater EW changes **then**
  For all free vertex v_f in V_F or V_T **do**
    Apply algorithm 1 to v_f
  End for
Else if $W_E(V_F) \Leftarrow W_E(V_F)$ **then**
  For all edge e adjacent to v_m **do**
    $N_F \Leftarrow$ number of vertices on e in V_F
    $N_T \Leftarrow$ number of vertices on e in V_T
    If $N_F + N_T > 2$ **then**
      If $N_T = 0$ **then**
        For all free vertex v_f on e in V_F **do**
          EBGain(v_f) $\Leftarrow$ EBGain(v_f) $- w_b(e)$
        End for
      End if
      If $N_T = 1$ **then**
        For all free vertex v_f on e in V_T **do**
          EBGain(v_f) $\Leftarrow$ EBGain(v_f) $+ w_b(e)$
        End for
      End if
      If $N_F = 1$ **then**
        For all free vertex v_f on e in V_T **do**
          EBGain(v_f) $\Leftarrow$ EBGain(v_f) $- w_b(e)$
        End for
      End if
      If $N_F = 2$ **then**
        For all free vertex v_f on e in V_F **do**
          EBGain(v_f) $\Leftarrow$ EBGain(v_f) $+ w_b(e)$
        End for
      End if
    End if
  End for
Else
  Update corresponding EBGain in the opposite direction when $W_E(V_F) > W_E(V_F)$
End if

The FMEB iteration contains several passes and stops when the Cost(EB) is no longer reduced or when it becomes negative. Since the final goal of the FMEB algorithm is to produce a solution with a negative Cost(EB), it stops within one vertex movement pass once the goal is achieved. There may exist a vertex v such that EBGain(v) > $|W_E(V_1) > W_E(V_2)|$ after performing Algorithm 2 for the EBGain update with this large EBGain not the real gain of Cost(EB) reduction because $\text{Cost(EB)}_{\text{before}} - \text{EBGain}(v) \neq \text{Cost(EB)}_{\text{after}}$. In this situation, if EBGain(v) $< 2\delta_{CW_E}|W_E(V_1) + W_E(V_2)|$, the pseudo Cost(EB) still satisfies δ_{CW_E}, and the pass should still stop immediately after moving v, otherwise, Cost(EB) violates δ_{CW_E} and stopping the pass should be decided according to the hypergraph size. For large hypergraphs size, the algorithm stops within one pass once the Cost(EB) becomes negative even if it is the pseudo event because this situation does not often occur. For small hypergraphs, this pseudo terminating condition should be avoided, however. Since all FMEB operations are proportional to FM in one pass when the

block with the greatest edge weight does not change, which happens in most cases, the time consumption of FMEB is almost linearly related to the hypergraph size[16].

When performing FMEB, the total cut size $S(C)$ usually increases. To prevent $S(C)$ from increasing too fast, the FMEB iteration is run in two phases. During the first phase, the maximum increment of $S(C)$ is limited, and only vertices satisfying both this limit and the vertex weight balancing constraint δ_{CW_V} are allowed to move. If the first phase ends with a positive Cost(EB), the second phase is performed. During the second phase, all vertices satisfying δ_{CW_V} are allowed to move to finally get a negative Cost(EB) and generate a solution satisfying the edge weight balancing constraint δ_{CW_E}. This two phase iteration is alternated with the other FM variant to satisfy the dual constraints in the multilevel paradigm described in the next section.

2.2 Multilevel partitioning for dual constraints

The two phase FMEB and FMDC algorithm is integrated into a multilevel paradigm which includes multi-start and V-cycle schemes. The multilevel paradigm is shown in Fig. 2 with some changes added to improve the algorithm.

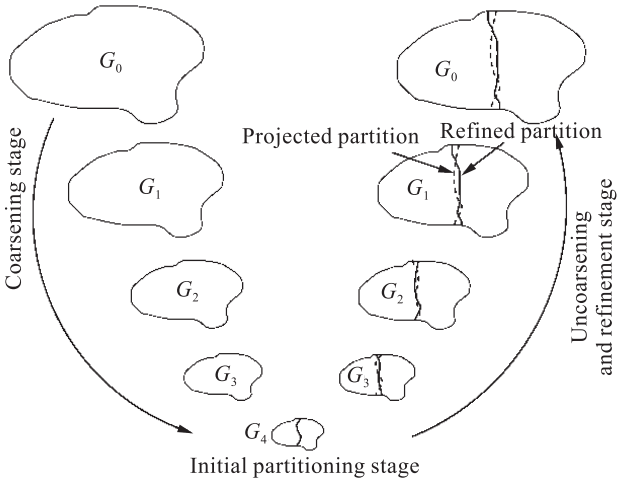


Fig. 2 Multilevel hypergraph bipartition paradigm[11]

During the coarsening stage of the multilevel algorithm, the size of the original hypergraph is successively decreased level by level from bottom to top by vertex clustering. The PinEC scheme[10] is used for the clustering to encourage removal of more pins and discourages merging of large clusters.

At levels other than the bottom level, some edges will disappear in the coarsened hypergraph if they are totally contained in a cluster, so considering only the edge weights in the coarsened hypergraph will not accurately evaluate the edge weight imbalance proportion of the original hypergraph and refinement on these coarsened hypergraphs may achieve the wrong objective. To solve this problem, the algorithm uses edge weight inheritance of the vertices $w_i(v)$. Let v_p be one new cluster vertex and $v_{c1}, v_{c2}, \cdots, v_{cm}$ be the m children clusters to be clustered into v_p during clustering and $e_{c1}, e_{c2}, \cdots, e_{cn}$ be the n edges connecting these children clusters and which are totally contained in v_p. Then assign

$$w_i(v_p) = \sum_{k=1}^{m} w_i(v_{ck}) + \sum_{j=1}^{n} w_b(e_{cj}) \tag{4}$$

so that the edge weights are inherited recursively level by level.

Algorithm 3 computes the EBGains in FMEB and is adapted from Algorithm 1.

Algorithm 3 Calculate EBGain of v_f with edge weight inheritance

$V_F \Leftarrow$ the block containing v_f
$V_T \Leftarrow$ the other block
$W_{E,\text{temp}}(V_F) \Leftarrow W_E(V_F) - w_i(v_f)$
$W_{E,\text{temp}}(V_T) \Leftarrow W_E(V_T) + w_i(v_f)$
For all edge e adjacent to v_f **do**
  $N_F \Leftarrow$ number of vertices on e in V_F
  $N_T \Leftarrow$ number of vertices on e in V_T
  If $N_T = 0$ or $N_F = 1$ **then**
    $W_{E,\text{temp}}(V_F) \Leftarrow W_{E,\text{temp}}(V_F) - w_b(e)/2$
    $W_{E,\text{temp}}(V_T) \Leftarrow W_{E,\text{temp}}(V_T) + w_b(e)/2$
  End if
End for
$\text{EBGain}(v_f) \Leftarrow |W_E(V_F) - W_E(V_T)| - |W_{E,\text{temp}}(V_F) - W_{E,\text{temp}}(V_T)|$

Note that, as long as the blocks with the greatest edge weight does not change, the EBGain update is only affected by the critical edges and there is no need to adjust the update algorithm; otherwise, all the EBGains of free vertices should be recomputed as in the similar to gain computing algorithm and the inherited weights should be considered as well.

The current FM variant FMDC is similar to LIFO-FM[17] with the only difference being the moving vertex selection scheme. In FMDC, the vertices are examined before selection, and ones whose movement violates δ_{CW_V} or δ_{CW_E} are not allowed to move. Note that the FMDC input produced by FMEB may not satisfy the dual constraints during the uncoarsening stage

at levels other than the bottom level, so vertices whose movements reduce the edge weight imbalance are also allowed to move for the purpose of minimizing the imbalance proportion.

The initial multilevel partitioning stage and uncoarsening stage alternates between the two phase FMEB and the FMDC. The top level first performs the procedure with constraints looser than the given constraints to expand the search space and allow vertices with large clustering weights to move. After the initial partitioning, the solution may still violate the edge weight constraint δ_{CW_E} because of the existence of vertices with large inherited weights or those connected to edges with extremely large weights, so the two phase FMEB is always performed before the FMDC to ensure that the final solution satisfies δ_{CW_E}.

3 Wire Density Driven Global Placement

MLDC is used in this top-down global placement framework to produce the wire density uniformity. The hyperedge weights are designed to better estimate the contribution of its corresponding circuit net to the wire density of each subregion of a bin's bisection.

3.1 Wire density weight model of hyperedge

In traditional partition-based placement, the vertex weights $w(v)$ are set equal to be the area of the corresponding circuit modules, and the cut weight of a hyperedge $w_c(e)$ is designed to take into account the terminal propagation so that minimizing of the hypergraph partitioning cut size minimizes the total wire length for the circuit placement. The wire density hyperedge balancing weight $w_b(e)$ used in this partitioning has dual constraints. Satisfying constraint δ_{cEW} means that the average wire weight per area is balanced between the two subregions of each bisection; thus, the recursive bisection based on the wire density balancing during the top-down placement will result in a uniform wire distribution throughout the entire layout.

The key factor is to determine $w_b(e)$ such that it is closely related to the circuit net contribution to the wire density of the subregion after routing. A net is called a bin's internal net if all the modules on the net will be placed inside the bin; otherwise, it is called the bin's external net. Figure 3 shows that both the internal and external nets of each subregion can cause an unbalanced wire density distribution. Thus, the wire hyperedge density weight during bisection of a bin is set as $w_b(e) = w_w(e) \cdot w_l(e)$, where $w_w(e)$ represents the width factor of the corresponding circuit net and $w_l(e)$ reflects the predicted wire length of each net in a subregion after bisection of a bin. Let PN(e) be the number of pins of a hyperedge inside the bin to be bisected, then the wire length factor is

$$w_l(e) = \alpha \mathrm{PN}(e) + \varepsilon(e) \tag{5}$$

where $\varepsilon(e)$ is 0 if e represents the bin's internal net or a non-zero number β, if e represents the bin's external net. α and β are empirical factors. PN(e) appears in the length weight factor because a net adjacent to more modules in a bin has greater probability to be longer after routing and a pin in a net represents at least one wire connected to other pins. However, the pin density alone cannot reflect the wires lengths between a bin's inside modules and its terminals, so $\varepsilon(e)$ is add to account for external nets.

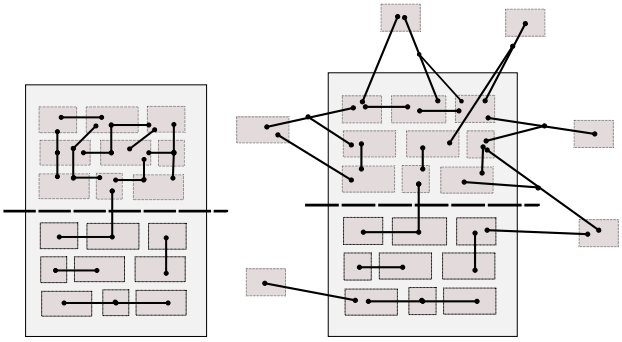


Fig. 3 (a) Unbalanced wire distribution caused by internal nets in a subregion. (b) Unbalanced wire distribution caused by external nets for a subregion. The final wire density of the upper subregion is greater that for the other.

3.2 Top-down global placement framework

During a bin's bisection in top-down global placement, a hypergraph is produced from its inside modules and terminals, with partitioning then performed on the hypergraph to determine the modules to be placed in each subregion. In MLDC, $w_c(e)$ and $w_b(e)$ are assigned to each hyperedge e. The present wire density weight model is combined with the Steiner-Tree minimization scheme of ROOSTER for the edge weight assignment. For each net, w_1 is calculated as the Steiner-Tree weight when all the inside modules are placed in subregion 1, as w_2 when all are placed in subregion 2

and as w_{12} when they are in both subregions. Two corresponding hyperedges are created in the partitioning hypergraph with one new hyperedge having $w_c(e) = w_{12} - \max(w_1, w_2)$ and $w_b(e) = w_{width}(e) \cdot \alpha PN(e)$ which connects all the bin's inside modules on the net, which the other new hyperedge has $w_c(e) = |w_1 - w_2|$ and $w_b(e) = w_{width}(e) \cdot \varepsilon(e)$ which connects all the bin's inside modules on the net to the propagated terminal in subregion 2 if $w_1 > w_2$ or to the terminal in subregion 1 if $w_1 < w_2$.

For large bins with hundreds of modules, MLDC is used to seek a more uniform wire distribution. For bins with less than 100 modules, flat FM partitioning is used. Recursive partitioning is performed until a bin is small enough, followed by detailed placement. Algorithm 4 illustrates the top-down global placement framework.

Algorithm 4 Wire density driven top-down global placement

```
Enqueue(top-level placement bin)
while queue not empty do
  bin ⇐ Dequeue()
  if bin small enough then
    Process bin with end-case placer
  else
    Choose a cut-line for the bin
    Build a partitioning hypergraph from the bin
    for all net adjacent to a module in the bin do
      Add proper w_c(e) and w_b(e) to hyperedges
    end for
    if bin very large then
      Apply MLDC
    else
      Apply flat FM partitioning
    end if
    Bisect the bin into two child bins
    Enqueue (each child bin)
  end if
end while
```

4 Test Results

4.1 Measurement of wire density uniformity

The wire distribution uniformity of a placement solution should be measured after routing. The results are evaluated using FGR[18], a global router which won first place in the ISPD 2007 Global Routing Contest.

For a global routing gcell c, the horizontal wire density $D^h(c)$ is defined as the ratio of the number of horizontal nets crossing the gcell to the horizontal capacity of the two sides. δ_D^h denotes the standard deviation of $D^h(c)$ for all the routing gcells. The definitions of $D^v(c)$ and δ_D^v for vertical nets are similar. The uniformity of the wire distribution can be measured by δ_D^h and δ_D^v, when small values imply greater uniformity.

4.2 Comparison to ROOSTER

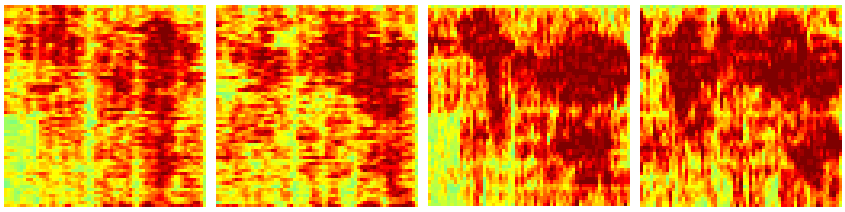
The algorithm was implemented in C++ based on UMPack[19], a well-known suite of open source physical design tools including ROOSTER. The test environment was a GNU/Linux server with a Xeon 3.0 GHz CPU and 5.9 GB memory. The IBMv2 placement benchmark suite[20] is used, with the routing configurations given in the FGR website as shown in Table 2. Since these placement benchmarks lack information on the width of the nets, the wire width factor was simply set to one in the current. The wire density balancing constraint δ_{cEW} for the multilevel partitioning was set to 5%. The parameters α and β were set to 1 and 2. The FGR maximum runtime was limited to one hour.

Table 2 Configurations of IBM benchmarks for global routing

Circuit	Grids	V/H cap
IBM01	64×64	1214
IBM02	80×64	2234
IBM07	192×64	2136
IBM08	192×64	2132
IBM09	256×64	1428
IBM10	256×64	2740
IBM11	256×64	1435
IBM12	246×64	2448

The results for the current algorithm are compared to ROOSTER for the IBMv2 placement benchmarks in Table 3. The column labeled rWL is the total wire length after the global routing, and o.f. is the routing overfill. The run time is the time cost of the placement. The results show that our algorithm is capable of producing solutions with a more uniformly distributed wire density, outperforming ROOSTER by 3.1% and 2.3% on average in terms of the minimization of the standard deviation of the routing gcell horizontal and vertical wire densities, i.e., δ_D^h and δ_D^v. The total wire length increment is 3.0% and the overfill is limited.

The placement time consumption increase is 18% on average. Figure 4 shows the horizontal and vertical wire density maps after applying FGR to the solutions from ROOSTER and the current work on the IBM01 hard benchmark. Gcells with higher wire densities are scattered more uniformly throughout the layout in Fig. 4b than in Fig. 4a due to the effectiveness of the current global placement framework.



(a) (b) (c) (d)

Fig. 4 Horizontal and vertical wire density maps for the IBM01-hard benchmark after applying FGR. (a) , (c) are from the placement by ROOSTER and (b), (d) are from the current framework.

Table 3 A comparision of present results to ROOSTER on the IBMV2 benchmarks

Circuits	Current					ROOSTER				
	rWL	o.f.	δ_D^{h}	δ_D^{v}	Time (s)	rWL	o.f.	δ_D^{h}	δ_D^{v}	Time (s)
IBM01e	61 247	0	0.264	0.275	2691	58 602	0	0.268	0.290	2346
IBM01h	62 013	0	0.263	0.275	2894	58 319	0	0.273	0.289	2451
IBM02e	171 658	0	0.277	0.283	7259	168 581	0	0.287	0.290	5290
IBM02h	172 317	0	0.274	0.273	7837	168 536	0	0.277	0.281	5548
IBM07e	465 846	3604	0.278	0.246	16 175	435 927	0	0.273	0.242	13 015
IBM07h	424 382	0	0.268	0.260	14 792	409 857	0	0.280	0.257	10 978
IBM08e	463 243	3423	0.236	0.230	16 880	459 866	348	0.233	0.243	15 219
IBM08h	449 007	907	0.240	0.245	15 024	447 991	17	0.237	0.252	10 686
IBM09e	424 726	0	0.246	0.244	15 649	415 361	0	0.260	0.253	9854
IBM09h	425 865	0	0.243	0.239	15 184	414 625	0	0.261	0.256	11 076
IBM10e	673 145	206	0.251	0.196	22 162	637 910	201	0.261	0.185	17 591
IBM10h	664 770	0	0.246	0.194	19 084	636 352	77	0.267	0.189	18 346
IBM11e	548 291	904	0.257	0.207	16 370	538 611	577	0.261	0.196	15 191
IBM11h	547 386	674	0.249	0.201	16 279	535 042	603	0.261	0.209	14 179
IBM12e	788 856	0	0.251	0.166	20 415	761 381	0	0.260	0.172	17 272
IBM12h	855 833	2695	0.251	0.180	19 977	845 815	2799	0.261	0.200	20 295
Ratio	1.000		1.000	1.000	1.000	0.970		1.031	1.023	0.818

5 Conclusions

A wire density driven top-down global placement algorithm was developed that is capable of producing solutions with more uniformly distributed wire densities for better CMP variation control. A multilevel hypergraph partitioning satisfying dual constraints on both the vertex weight and the wire density weight is performed recursively in the top-down global placement framework. Two flat FM variants, a two-phase FMEB to generate more balanced solutions and an FMDC to minimize the cut size for dual constraints, are iterated in the initial partitioning stage and the refinement stage of the multilevel algorithm.

Tests show that the partitioning algorithm produces solutions with much more balanced edge weights between two partitioning blocks than the leading-edge hMetis and MLPart with only a small increment in the cut size and run time. The algorithm also outperforms FM in terms of both the edge weight balancing and the cut size minimization. The partitioning algorithm enables this global placement algorithm to produce more uniform wire density distributions than ROOSTER with only a small increase in the total wire length and run time.

Acknowledgements

The author thanks the contributors of the UMPack open source physical design tool, on which the experiments of this work is based, and the contributors of FGR who helped evaluate the results.

References

[1] Park T H. Characterization and modeling of pattern dependencies in copper interconnects for integrated circuits [Dissertation]. Department of EECS, MIT, USA, 2002.

[2] Tian R, Wong D F, Boone R. Model-based dummy feature placement for oxide chemical-mechanical polishing manufacturability. In: Proceedings of the International Conference on Computer Aided Design. Los Angeles, California, USA, 2000: 667-670.

[3] Leung K S. Spider: Simultaneous post-layout ir-drop and metal density enhancement with redundant fill. In: Proceedings of the International Conference on Computer-Aided Design. San Jose, California, USA, 2005: 33-38.

[4] Chen Y, Gupta P, Kahng A B. Performance-impact limited area fill synthesis. In: Proceedings of the International Conference on Computer Aided Design. Anaheim, California, USA, 2003: 22-27.

[5] Pan D Z, Cho M. Synergistic physical synthesis for manufacturability/variability in 45 nm designs and beyond. In: Proceedings of the Asia and South Pacific Design Automation Conference. COEX, Seoul, Korea 2008: 220-225.

[6] Cho M, Pan D Z, Xiang H, et al. Wire density driven global routing for cmp variation and timing. In: Proceedings of the International Conference on Computer-Aided Design. New York, NY, USA, 2006: 487-492.

[7] Chen H Y, Chou S J, Wang S L, et al. Novel wire density driven full-chip routing for CMP variation control. In: Proceedings of the International Conference on Computer Aided Design. San Jose, California, USA, 2007: 831-838.

[8] Chen T C, Cho M, Pan D Z, et al. Metal-density driven placement for cmp variation and routability. In: Proceedings of the International Symposium on Physical Design. New York, NY, USA, 2008: 31-38.

[9] Roy J A, Lu J F, Markov I L. Seeing the forest and the trees: Steiner wirelength optimization in placemen. In: Proceedings of the International Symposium on Physical Design. New York, NY, USA, 2006: 78-85.

[10] Caldwell A E, Kahng A B, Markov I L. Improved algorithms for hypergraph bipartitioning. In: Proceedings of the Asia and South Pacific Design Automation Conference. New York, NY, USA, 2000: 661-666.

[11] Karypis G, Aggarwal R, Kumar V, et al. Multilevel hypergraph partitioning: Application in VLSI domain. In: Proceedings of the International Conference on Design Automation. New York, NY, USA, 1997: 526-529.

[12] Dutt S, Deng W. VLSI circuit partitioning by cluster-removal using iterative improvement techniques. In: Proceedings of the International Conference on Computer-aided Design. San Jose, California, USA, 1996: 194-200.

[13] Caldwell A E, Kahng A B, Markov I L. Can recursive bisection alone produce routable placements? In: Proceedings of the Design Automation Conference. Bergen, Norway, 2000: 477-482.

[14] Westra J, Bartels C, Groeneveld P. Probabilistic congestion prediction. In: Proceedings of the International Symposium on Physical Design. New York, NY, USA, 2004: 204-209.

[15] Alpert C J. Partitioning Benchmarks for the VLSI CAD Community. http://vlsicad.ucsd.edu/UCLAWeb/cheese/ispd98.html. 2010.

[16] Fiduccia C M, Mattheyses R M. A linear-time heuristic for improving network partitions. In: 25 Years of DAC: Papers on Twenty-Five Years of Electronic Design Automation. New York, NY, USA, 1988: 241-247.

[17] Hagen L, Huang D, Kahng A. On implementation choices for iterative improvement partitioning algorithms. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 1997, **16**(10): 1199-1205.

[18] Roy J A, Markov I L. High-performance routing at the nanometer scale. In: Proceedings of the International Conference on Computer-Aided Design. Piscataway, NJ, USA, 2007: 496-502.

[19] Adya S N, Caldwell A E, Kahng A B, et al. UMICH Physical Design Tools. http://vlsicad.eecs.umich.edu/BK/PDtools. 2010.

[20] Yang W, Choi B K, Sarrafzadeh M. Routability driven white space allocation for fixed-die standard-cell placement. In: Proceedings of the International Symposium on Physical Design. New York, NY, USA: ACM, 2002: 42-47.