POLAR: Placement based on Novel Rough Legalization and Refinement

Tao Lin [1], Chris Chu [1], Joseph R. Shinnerl [2], Ismail Bustany [2] and Ivailo Nedelchev [2]

[1]Department of Electrical and Computer Engineering, Iowa State University, IA
[2]Mentor Graphics Corporation, Fremont, CA
{tlin, cnchu}@iastate.edu, {joseph_shinnerl, ismail_bustany, ivailo_nedelchev}@mentor.com

***Abstract*—A new quadratic global placer called POLAR is proposed. POLAR is based on novel techniques for rough legalization and wirelength refinement. During look-ahead rough legalization (LAL), relative positions of cells are maintained as they are relocated with minimal displacement to relieve excess area density. For each "hotspot" where placement overfill occurs, an expansion region covering the hotspot is constructed. Then the movable cells within each of these expansion regions are evenly assigned to density bins inside the expansion region by displacement-minimizing recursive bisection. In addition, a fast density-preserving and wirelength-reducing discrete refinement is applied to the first few LAL placements before each of these is used to augment the quadratic model used to obtain the next major placement iteration. The experimental results show that POLAR outperforms the state-of-the-art academic placers over the ISPD 2005 benchmarks.**

I. Introduction

Placement is considered one of the most fundamental physical design problems. Although it has been extensively studied for decades, [1] indicates that it is still a "hot topic" considering the gradually increasing problem size and varieties of new constraints and objectives due to technology scaling.

The quadratic approach is one of the most promising approaches for modern placement due to its low CPU runtime and good placement quality. It approximates the wirelength by a convex quadratic function, which can be minimized efficiently by solving a linear system. However, minimizing just the wirelength would lead to considerable cell overlapping. Therefore, various techniques have been proposed to spread out the cells while maintaining the quadratic nature of optimization.

As the most popular spreading technique, the iterative force-directed approach interprets the quadratic placement problem as a classical mechanics problem of finding the equilibrium configuration for a spring system. In each iteration, spreading forces required to spread out the cells are calculated based on current cell positions. Then the modified spring system is solved to obtain the new cell positions. This spreading process continues until the cell distribution is almost even and the wirelength is not improved any more.

There are many ways to generate spreading force. Kraftwerk2 [2] and DPlace [3] are based on density gradient; Kraftwerk2 utilizes a Poisson potential by a generic supply and demand system, while DPlace models the diffusion process by solving a differential equation relating to cell density. mFAR [4] achieves the spreading forces by moving cells from those bins with overflow to those without. FastPlace [5] and RQL [6] move the cells from high density bins to the low density adjacent bins by cell shifting.

In recent years, the placers [7], [8], [9], [10], [11] adopt a new technique called rough legalization. The key idea of rough legalization is that almost legal placement is used to guide the spreading force generation. In SimPL [7], the rough legalization is implemented by top-down geometric partitioning and non-linear scaling. In ComPLx [8], the entire placement process is modelled by subgradient primal-dual Lagrange optimization, and LAL is modelled by a feasibility projection. SimPLR [9] and Ripple [10] both extend SimPL to handle routing congestion. MAPLE [11] combines the rough legalization idea with multilevel clustering and improvement of iterative local refinement.

In this paper, we propose a new quadratic placer called POLAR inspired by SimPL [7] and based upon a novel rough legalization and refinement approach. We notice that while the placement solution by quadratic based wirelength minimization may have considerable overlaps, the relative positions of cells can be trusted in producing a legal placement with good wirelength. Hence, during rough legalization, our goal is to maintain the relative positions of cells as best as we can while minimizing the cell movements. To achieve this goal, for each hotspot, we search for a good expansion region by enumerating many feasible candidates. Then within each expansion region, recursive bisection is performed to evenly assign the movable cells to each bin. In order to further improve the quality of placement produced by rough legalization, a fast density preserving global refinement is applied.

The contributions of this paper are listed as follows:

- An enumerative method is explained to search for a good expansion region for each hotspot. Our method enumerates many feasible candidates and selects the best one. This enumerating method is very helpful in both maintaining the cell relative positions and minimizing cell movements.
- A novel recursive bisection based method is proposed to evenly assign the cells to each bin within each expansion region. This method effectively keeps the relative positions of cells.
- A fast density preserving global refinement is introduced. The refinement is applied right after rough

legalization at the early stage of the placement process in order to improve the quality of the roughly legal placement solution.

The rest of this paper is organized as follows. Section II presents the preliminary. Section III elaborates the POLAR algorithm. Section IV shows the experimental results. Finally, the conclusion is made in Section V.

II. PRELIMINARY

A circuit can be represented by a hypergraph $G = (V, E)$, where V is the set of cells and E is the set of nets. Global placement tries to determine the physical positions of the cells without violating the placement density constraints. We denote the x-coordinates of cells by a vector $\mathbf{x} = \left(x_1, x_2, \cdots, x_{|V|}\right)$, and y-coordinates by $\mathbf{y} = \left(y_1, y_2, \cdots, y_{|V|}\right)$, the objective is to minimize the half-perimeter wirelength (HPWL):

$$\text{HPWL}(\mathbf{x}, \mathbf{y}) = \Sigma_{e \in E}[\max_{i \in e} x_i - \min_{i \in e} x_i + \max_{i \in e} y_i - \min_{i \in e} y_i] \quad (1)$$

A. Quadratic optimization

Assuming that all the nets have 2 pins. So the wirelength is given by Manhattan distance between the cells. If the Manhattan distance is approximated by squared Euclidean distance, the cost function ϕ of global placement is as follows:

$$\phi(\mathbf{x}, \mathbf{y}) = \frac{1}{2}\mathbf{x}^T Q_x \mathbf{x} + \mathbf{c}_\mathbf{x}^T \mathbf{x} + \frac{1}{2}\mathbf{y}^T Q_x \mathbf{y} + \mathbf{c}_\mathbf{y}^T \mathbf{y} + const \quad (2)$$

where the matrices Q_x and Q_y are both sparse symmetric positive definite. Minimizing ϕ is equal to solving the linear system in Eq. (3).

$$\nabla\phi(\mathbf{x}, \mathbf{y}) = Q_x \mathbf{x} + \mathbf{c}_\mathbf{x} + Q_y \mathbf{y} + \mathbf{c}_\mathbf{y} = 0 \quad (3)$$

In POLAR, the preconditioned conjugate gradient (PCG) method with incomplete Cholesky decomposition [12] is used to solve Eq. (3).

B. Net model

Since many nets in the circuit have more than two pins, to get the quadratic cost function (2), every multi-pin net should be transformed into a set of 2-pin nets. Since the net model has significant influence on the matrices Q_x and Q_y, it is important to choose suitable net model for quadratic placer to perform well. Considering both accuracy and runtime, Bound to Bound (B2B) net model has been proved efficient in practice.

C. Spreading force realization

To reduce the cell overlapping, spreading forces are added to guide the cells toward their target positions. [4] proposed a simple way (which is called fixed-point technique) to realize spreading force by adding 2-pin pseudo net connecting the cell's original position to its target position. Then the matrices Q_x and Q_y are updated and linear system (3) is solved again. In POLAR, we also use fixed-point technique.

III. THE POLAR ALGORITHM

A. Algorithm outline

As shown in Fig. 1, POLAR is composed of three stages: initial placement, density-driven placement, local legalization. The initial placement only minimizes the wirelength without considering the cell overlapping. Density-driven placement reduces cell overlapping gradually, its output is a globally even, but locally illegal placement, because all the cells are placed in the center of bins. At the end, an almost legal placement result is achieved by local legalization, which spreads the cells out within each bin.

In initial placement stage, the hybrid net model is responsible for the initial solution, then B2B net model optimizes the wirelength iteration by iteration. Usually, three iterations are enough. In each iteration of density-driven placement, hotspots are first identified as spatially contiguous collections of over-filled bins. The expansion region of each hotspot is then constructed by enumerating and comparing all the candidates. Each expansion region should have sufficient free space for placing the cells assigned to it. Then, within each expansion region, the cells are evenly assigned to each bin. With the new cell positions or target positions, the linear system (3) is updated and solved again. In addition, in order to correct the misleading cell positions at the early stage, a fast density preserving global refinement optimizes the wirelength in the first five iterations. The density-driven placement runs until it satisfies the converge condition, which is defined as that the number of iterations is greater than 70 or the global wirelength is improved by less than 10% in the last 10 iterations. Once the almost-legal placement is obtained, strict legalization removes all remaining overlap, and detailed placement is applied.

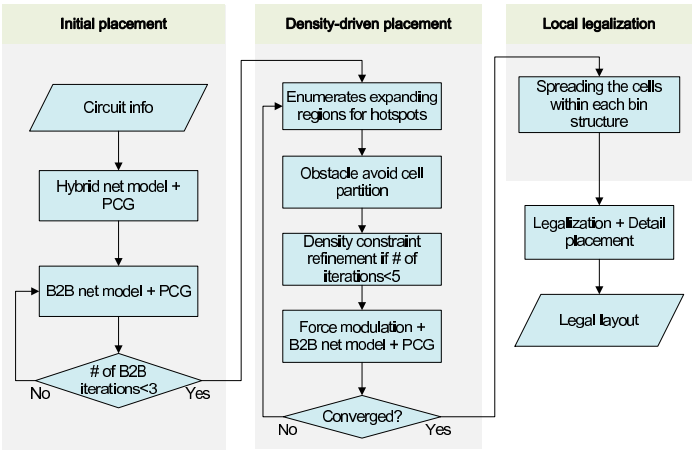

Fig. 1. The outline of POLAR.

The rough legalization aims to get an almost even placement with minimal cell movements simultaneously keeping the relative positions of cells. To simplify the problem, we ignore local legality, which means that the placement region is split into a set of equal sized bins, and all the cells can only be placed in the center of bin (local legality is relaxed). So the rough legalization can be formulated as a cell-to-bin assignment. The bin is expected to accommodate about 15-40$\times$ average movable cell area, for ISPD2005 benchmarks, we used 4 $\times$ 4 standard row height bins.

B. Expansion regions enumeration

The placement region is divided into a uniform grid of $m \times n$ equal-sized bins. Before elaborating the expansion on region enumeration, we formally define bin graph, hotspot, and hotspot expansion region as follows.

Definition 1: The **bin graph** for the uniform bin grid is the graph in which (i) each bin represents a vertex, and (ii) two vertices are joined by a graph edge if and only if the two bins for those vertices are directly adjacent, either horizontally or vertically. That is, referring to a bin by its (row,column) coordinates in the uniform bin grid, bins (i, j) and (k, l) are adjacent if and only if $|k - i| + |l - j| = 1$.

Definition 2: A **hotspot** is a spatially contiguous collection of overfilled bins, i.e., a connected subgraph of overfilled bins in the bin graph. A hotspot is also called a "clump." For any pair of bins in the clump, there is a path in the bin graph connecting them, the edges of the path can only be vertical or horizontal, and this path cannot go through the bins outside of the clump.

Definition 3: The **expansion region** of a hotspot is a set of bins, which completely covers the hotspot and have enough available space to accommodate the movable cells within them.

In this paper, expansion regions for hotspots may overlap, and we only consider the rectangular expansion region, which can be denoted by a quadruple (l_x, l_y, r_x, r_y), where (l_x, l_y) and (r_x, r_y) are respectively the bin coordinates of left-down corner and right-up corner. However, large rectangular expansion regions are split into contiguous aggregations of mutually overlapping smaller windows, as described below. In effect, such splitting produces approximations to non-rectangular, rectilinear covers of irregularly shaped clumps. The insight comes from that an irregularly shaped hotspot may result in an unnecessarily large expansion region, with unnecessarily large cell displacements as a consequence. For example, as shown in Fig. 2, the right figure with two expansion regions achieves a smaller expansion region, possibly with less displacement, than the left one with a larger, single-window expansion region. In POLAR, we try to avoid unnecessary big expansion regions by imposing upper bound on the number of bins in a hotspot. If a hotspot contains more than upper bound number of bins, it will be automatically split into a set of smaller, spatially contiguous hotspots.[1]

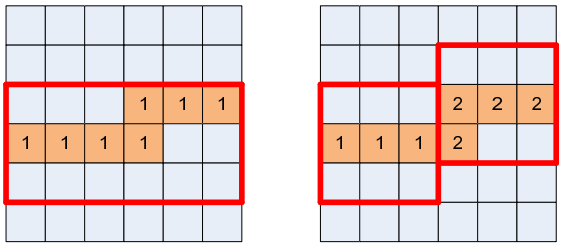

Fig. 2. Imposing upper bound on the number of bins in a hotspot. The overflow bins are coloured , the boundary of expansion region is marked by the red bold line.

[1]Expansion regions for spatially adjacent hotspots are allowed to overlap as needed to capture sufficient available placement area for LAL. Details concerning expansion-region ordering in such situations are left to future work

Suppose the placement region is split to $m \times n$ bin grids, the number of expansion regions is O(m^2n^2). Enumerating all feasible ones is time consuming. To trade off the runtime and quality, the ones whose geometry center is also the gravity center of hotspot, are enumerated. And the one with minimal size and reasonable aspect ratio would be picked.

The algorithm to enumerate the feasible expansion regions is presented in Algorithm 1. The space utilization ratio γ of an expansion region is the ratio of, total area of cells within expansion region, and the area of expansion region. It can be quickly calculated by a looking up table method. Suppose the total area of cells within region $(0, 0, i, j)$ is stored in $O[i][j]$, and the area of region $(0, 0, i, j)$ is stored in $A[i][j]$, the space utilization ratio of expansion region (l_x, l_y, r_x, r_y) can be represented by the equation (4). In addition, the two 2-dimension arrays $O[m][n]$ and $A[m][n]$ can be easily calculated and updated by dynamic programming. Fig. 3 illustrates how dynamic programming works, note that the whitespace (blackspace) of blue boundary window is subtracted twice by the middle two items, so the last item is added.

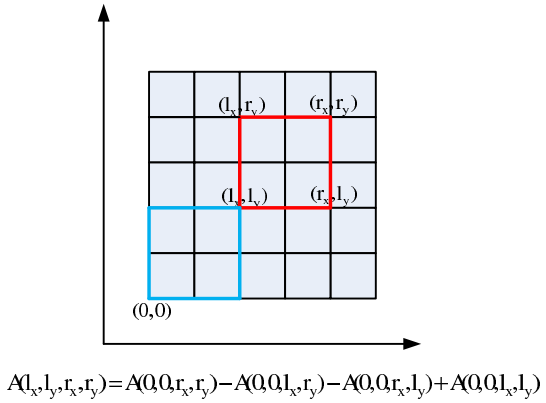

Fig. 3. Lookup table method for space calculation

$$\gamma = \frac{O[r_x][r_y] - O[r_x][l_y] - O[l_x][r_y] + O[l_x][l_y]}{A[r_x][r_y] - A[r_x][l_y] - A[l_x][r_y] + A[l_x][l_y]} \quad (4)$$

C. Recursive bisection based cell distribution

Once the expansion region for hotspot is determined, the cells (within expansion region) are evenly assigned to each bin (within expansion region) by a partition tree. To balance the disturbance of x and y-directed relative positions, the horizontal and vertical cut are applied alternatively similar with the slicing tree of [13].

However, different from [13], the partition tree should split the expansion regions into independent bins. This means that the cut lines can only belong to the set of bin boundaries, and each leaf node corresponds to each bin. Besides, the cells are not really moved during the construction of partition tree. Rather, a path from the root to the leaf (the bin to which it is finally assigned) is maintained for each cell. Hence, once the partition tree has been finished, each cell is assigned to the corresponding bin by going through its path.

The detail of constructing partition tree is shown in Algorithm 2. The partition tree is constructed level by level. For any

Algorithm 1 Expansion region enumeration for a hotspot

Require: $m \times n$ grids of placement region, hotspot H
Ensure: expansion region (l_x, l_y, r_x, r_y) for hotspot H

calculate the coordinate (g_x, g_y) of the bin where the gravity center of cells in the hotspot H locates;
$\Gamma = \emptyset$;
$found = false$;
$\varrho = 3$;
while $not found$ **do**
  for $radius_x = 1 \rightarrow \max\{g_x, m - g_x\}$ **do**
    for $radius_y = 1 \rightarrow \max\{g_y, n - g_y\}$ **do**
      $l_x = \max\{0, g_x - radius_x\}$;
      $l_y = \max\{0, g_y - radius_y\}$;
      $r_x = \min\{m - 1, g_x + radius_x\}$;
      $r_y = \min\{n - 1, g_y + radius_y\}$;
      if region R(l_x, l_y, r_x, r_y) does not cover H **then**
        continue;
      end if
      get the space utilization ratio γ of region R;
      if $\gamma <$ density target && aspect ratio $< \varrho$ **then**
        push region R into Γ;
        break;
      end if
    end for
  end for
  if $\Gamma = \emptyset$ **then**
    $\varrho += 1.0$;
  else
    $found = true$;
  end if
end while
return the (area) minimal region from Γ;

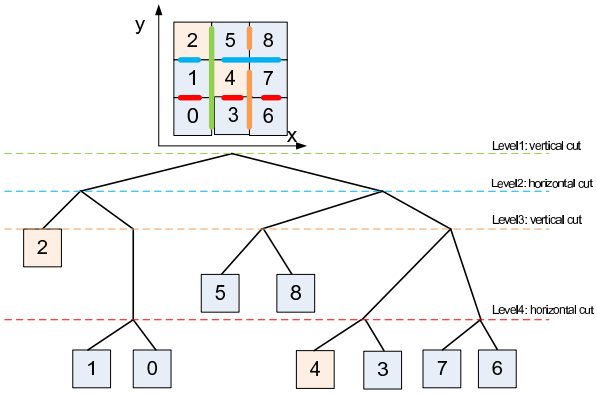

Fig. 4. An example of partition tree for 3×3 expansion region, the vertical cut line is first applied. The bin 2 and 4 are occupied by fixed macros

sub expansion region F (l_x, l_y, r_x, r_y), it is divided into two subregions L (left subregion by vertical cut $\frac{l_x + r_x}{2}$, or bottom subregion by horizontal cut $\frac{l_y + r_y}{2}$) and R (right subregion by vertical cut $\frac{l_x + r_x}{2}$, or top subregion by vertical cut $\frac{l_y + r_y}{2}$). If the available area of L and R are respectively A_L and A_R, then the partition pivot (area) is $\frac{A_L}{(A_L + A_R)} \times A_M$, where A_M is the total area of moveable cells within F. According to this pivot, cells are partitioned to L or R, which means that the maintained path of each cell goes forward a step. There is a special case during the partition. If the sub expansion region is 1×2 (2×1) grids, the vertical (horizontal) cut line can not be applied obviously. Therefore, this sub expansion region is not split in this level, but in the next level when the cut type is changed.

Algorithm 2 Cell distribution for one expansion region

Require: The cells S and their positions, expansion region (l_x, l_y, r_x, r_y)
Ensure: Each cell is placed in the center of its belonged bin.

1: get the cell ordering in both x and y directions;
2: determine the initial cut direction (vertical or horizontal);
3: push the root to a queue Q; ▷ Q stores the sub expansion regions, which have the same level
4: **while** true **do**
5: $\Phi = \emptyset$; ▷ Φ stores the next level sub expansion regions
6: **if** Q is empty **then** break; ▷ all the leaf nodes have been generated
7: **end if**
8: **while** Q is not empty **do**
9: pop a sub expansion region F from Q;
10: **if** F is a 1×2 (2×1) grids && the cut is vertical (horizontal) **then**
11: $\Phi = \Phi \cup F$;
12: **else if** F is not bin **then**
13: divide F into L and R;
14: $\Phi = \Phi \cup \{L, R\}$
15: **if** the cut is vertical **then**
16: The path of each cell goes forward a step according to x-directed cell ordering;
17: **else**
18: The path of each cell goes forward a step according to y-directed cell ordering;
19: **end if**
20: **end if**
21: **end while**
22: change the cut type;
23: push each elements of Φ into Q;
24: **end while**
25: **for** each cell in S **do**
26: find the belonged bin by going through its path from root to leaf, then update the position;
27: **end for**

Fig. 4 shows the partition tree for a 3×3 grids. In the first level, vertical cut is used so that $Q = \{\{0, 1, 2\}, \{3, 4, 5, 6, 7, 8\}\}$. In the second level, horizontal cut is used so that $Q = \{\{2\}, \{0, 1\}, \{5, 8\}, \{3, 4, 6, 7\}\}$. In the third level, since the sub expansion region $\{0, 1\}$ is a 2×1 grids and current cut type is vertical, so the line 11 in algorithm 2 is executed, then $Q = \{\{0, 1\}, \{5\}, \{8\}, \{3, 4\}, \{6, 7\}\}$. Finally, in the last level, $Q = \emptyset$, all the leaves are now bins, the partition tree is completely constructed.

In Algorithm 2, the cell ordering can be got in $O(|S|)$ by bucket sort. The number of levels in partition tree is $O(log[(r_x - l_x)(r_y - l_y)])$. In each level, the cells are scanned by x or y-directed ordering once(line 16 or 18 in Algorithm 2). Besides, for each cell, going through its path from root to leaf needs $O(log[(r_x - l_x)(r_y - l_y)])$. Combining all the above, the time complexity of algorithm 2 is $O(|S|log[(r_x - l_x)(r_y - l_y)])$.

D. Density preserving refinement

The relative positions of cells in the first several iterations have significant influence on the wirelength optimization. Considering that the initial placement ignores the cell overlapping when optimizing the wirelength, and the partition tree

based cell-to-bin assignment algorithm cannot maintain the x and y-directed relative positions simultaneously. Hence, a density preserving global refinement is expected to repair and get better cell positions with little perturbing of the density distribution. Note that this technique only takes effect in the first several iterations.

This refinement technique is based on the optimal region idea [14]. Given all other cells are fixed, the optimal region for cell v_i is defined as the region to place v_i where the wirelength is optimal. For any cell v_i, suppose the set of associated nets is denoted by N_i, for each net $p \in N_i$, the bounding box is denoted by $(x_l[p], y_l[p], x_r[p], y_r[p])$. [14] proves that the optimal region of v_i is $\left(x_l^{opt}, y_l^{opt}, x_r^{opt}, y_r^{opt}\right)$, where x_l^{opt}, x_r^{opt} are the medians of $(x_l[1], x_r[1], x_l[2], x_r[2], \cdots)$, y_l^{opt}, y_r^{opt} are the medians of $(y_l[1], y_r[1], y_l[2], y_r[2], \cdots)$.

The detail of refinement is shown in Algorithm 3. Each cell only has one opportunity to move (if it is locked, then it cannot be moved any more) (line 6 and 19). The unlocked cells which have high connectivity are first considered (from line 1 to line 4). An unlocked cell can be moved to its optimal region depends on the following two conditions: (a) it is not in its optimal region, (b) the destination bin has enough unlocked cells to balance density. (line 11 and line 16). Algorithm 3 would generate many move chains until all the cells are locked. Fig. 5 shows an example of two move chains.

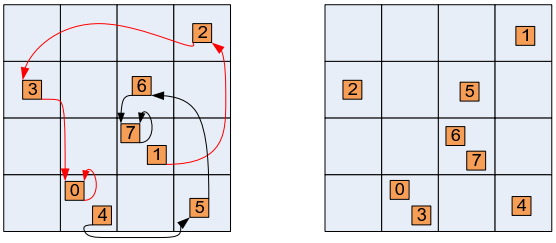

Fig. 5. An example of two move chains, the left figure shows the original cell position and move chains, the right figure shows the new cell position after movement.

E. Force modulation

After the cells' target positions are determined, the linear system (3) is updated and solved again. Firstly, the B2B net model is refreshed based on cells' target positions. Then, for each movable cell, a two pin pseudo net connecting it to its target position is added into the spring system. The weight of pseudo net is calculated as follows:

$$weight = \begin{cases} \varepsilon * \alpha^{i-1} & \text{if } i \leq 20 \\ \varepsilon * \alpha^{19} * \beta^{i-20} & \text{if } i > 20 \end{cases}$$

It is a two-stage force modulation, where ε is a small value, and $\beta > \alpha$. At the early stage, the weight of pseudo net is small in order to avoid big change of placement. While at the latter stage, weight of pseudo net is increased more quickly to speed up the convergence.

IV. Experimental Results

We implemented POLAR in C++, and ran it on a Linux PC with 16GB of memory and Intel Core-i3 3.3GHz CPU,

Algorithm 3 density preserving global refinement

Require: The cells V and net lists E
Ensure: Each cell is at most moved once
1: unlock all the cells, sort V by the number of associated nets in descend order;
2: **for** each bin **do**
3: sort the cells within the bin by the number of associated nets in descend order;
4: **end for**
5: **for** each cell $v_i \in V$ **do**
6: **if** v_i is not locked **then**
7: initialize a queue Q, push v_i into Q;
8: **while** Q is not empty **do**
9: pop a cell from Q, denoted by v_{top};
10: determine the destination bin of v_{top} by optimal region idea [14];
11: **if** the destination bin has no enough unlocked cell to move out to maintain density preserving $\parallel$ v_{top} is already in its optimal region **then**
12: don't move v_{top};
13: **else**
14: move v_{top}, update the bins' cell list;
15: **for** each unlocked cell $v_j \in$ destination bin **do**
16: push v_j into Q until the density preserving is satisfied.
17: **end for**
18: **end if**
19: lock v_{top};
20: **end while**
21: **end if**
22: **end for**

which has two cores. The ISPD 2005 benchmark suite [15] is used to verify the efficiency of POLAR. FastDP [16] is used to perform legalization and detailed placement. Note that we turn off the cell flipping of FastDP. ‘

A. Runtime breakdown of POLAR

The runtime of POLAR is shown in Table I. It is broken down into four components: PCG, rough legalization (which includes enumerative expansion region search, cell-to-bin assignment, and density preserving refinement), others (which includes I/O, wirelength calculation, linear system update, and so on), legalization, and detailed placement. On average, PCG takes the most of the total runtime (48.1%). Rough legalization and others take 13.9% and 15.1%, respectively. Legalization and detailed placement takes 22.9%. Since PCG is the main bottleneck, to achieve speedup, we have also developed a 2-core version in which the x- and y-direction linear systems are solved in parallel using openMP and Intel math kernel library (MKL) [17]. We get about 22% speedup of POLAR using two cores.

B. Comparison with previous placers

We compare POLAR with other state-of-the-art placers, ran them except MAPLE on the same platform (64 bit Linux OS, Intel Core-i3 3.3GHz CPU, 16GB of memory)[2]. The experimental results are shown in Table II. For wirelength, on

[2]NTUplacer3'version is V12.06.05, SimPL'version is V12.12.07, ComPLx's version is V13.07.30. MAPLE binary is unavailable to us, so MAPLE's data is transformed from [11] which show that it is 7.14 $\times$ slower than SimPL.

TABLE I. RUNTIME BREAKDOWN OF POLAR ON ISPD 2005 BENCHMARKS. RUNTIME IS MEASURED IN SECOND.

Benchmark	Global placement			Legalization & detailed placement	Total runtime (1-core)	Total runtime (2-core)	FastPlace3 [5] runtime
	PCG	Rough legalization	Others				
adaptec1	115	25	31	40	211	147	157
adaptec2	143	29	40	63	275	198	214
adaptec3	233	71	70	134	518	431	486
adaptec4	250	73	68	123	514	420	446
bigblue1	116	31	39	61	247	203	224
bigblue2	243	87	104	110	544	403	390
bigblue3	548	222	146	329	1245	1048	1130
bigblue4	1341	396	499	548	2784	2200	2086
Average	0.481	0.139	0.151	0.229	1.00	0.780	0.826

TABLE II. HPWL COMPARISON ON ISPD 2005 BENCHMARK SUITE [15]. RUNTIME IS MEASURED IN MINUTE.

Benchmark	NTUPlace3 [18]		mPL6 [19] + FastDP[16]		FastPlace3 [5]		SimPL [7]		ComPLx [8]		MAPLE [11]		POLAR		
	WL	Time	WL	Time	WL	Time	WL	Time	WL	Time	WL	Time	WL	Time (1-core)	Time (2-core)
adaptec1	79.83	5.1	77.29	19.3	78.66	2.61	77.53	2.43	77.79	2.64	76.36	19.33	76.49	2.85	2.45
adaptec2	90.08	6.7	90.02	20.08	94.06	3.57	91.11	3.50	88.97	3.27	86.95	24.23	86.72	4.33	3.3
adaptec3	232.72	13.5	207	62.76	214.13	8.10	203.79	6.75	203.57	7.58	209.78	50.72	201.9	9.15	7.18
adaptec4	215	16.4	188.87	59.28	197.50	7.43	184.75	5.18	183.22	6.65	179.91	49.22	183.02	8.09	7.00
bigblue1	96.94	9.7	96.18	24.41	96.67	3.73	95.59	4.46	95.30	5.2	93.74	24.70	94.04	4.44	3.38
bigblue2	158.26	24.2	148.91	68.13	155.74	6.50	145.87	5.61	145.39	7.03	144.55	43.82	144.45	7.88	6.71
bigblue3	343.57	27.8	335.53	93.23	365.16	18.83	351.65	17.71	337.96	18.1	323.05	89.51	321.14	24.03	17.46
bigblue4	825.48	81	814.13	212	836.20	34.76	791.29	26.86	788.80	35.8	775.71	231.44	793.56	39.06	36.66
Average	1.080	1.91	1.028	6.13	1.069	0.826	1.025	0.77	1.015	0.883	1.000	5.163	1.000	1.000	0.780

average, POLAR achieves improvement of 8.0%, 2.8%, 6.9%, 2.5%, and 1.5% versus NTUPlace3, mPL6, FastPlace3, SimPL, and ComPLx, respectively. For the total runtime using one core (including legalization and detailed placement), on average, POLAR is 1.91×, 6.13×, and 5.163× faster than NTUPlace3, mPL6 and MAPLE, respectively. POLAR is slightly slower than FastPlace3, SimPL, and ComPLx.

In summary, POLAR is competitive in wirelength and runtime simultaneously. It achieves the best published wirelength similar to MAPLE while is 5.16× faster. It is also almost as fast as the fastest academic placer SimPL but is 2.5% better in wirelength.

V. CONCLUSION AND FUTURE WORK

In this paper, we have proposed an efficient and effective global placer POLAR. It adopts the popular idea of rough legalization, and enhances it. The experimental results on ISPD 2005 benchmark suite verify that our placer is very comparable to the state-of-the-art placers in both runtime and placement quality.

We list the following things as future works. Firstly, enhance the algorithm to find ill-shaped hotspot rather than simply restricting the number of bins. Secondly, extend POLAR to handle with big movable macros. Thirdly, consider more optimization objectives, such as congestion, data path. Additionally, we plan to work on speeding the solver exploiting recent advancements in the solution of symmetric diagonally dominant linear systems.

REFERENCES

[1] C. Alpert, Z. Li, G.-J. Nam, C. N. Sze, N. Viswanathan, and S. I. Ward, "Placement: hot or not?," ICCAD '12, pp. 283–290, 2012.

[2] F. M. J. P. Spindler, U. Schilichtmann, "Kraftwerk2 - a fast force-directed quadratic placement approach using an accurate net model," *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 27, no. 8, pp. 1398–1411, 2008.

[3] T. Luo and D. Z. Pan, "DPlace2.0: a stable and efficient analytical placement based on diffusion," ASP-DAC '08, 2008.

[4] B. Hu, Y. Zeng, and M. Marek-Sadowska, "mFAR: fixed-points-addition-based VLSI placement algorithm," ISPD '05, pp. 239–241, 2005.

[5] N. Viswanathan, M. Pan, and C. Chu, "FastPlace 3.0: A fast multilevel quadratic placement algorithm with placement congestion control," ASP-DAC '07, pp. 135–140, 2007.

[6] N. Viswanathan, G.-J. Nam, C. J. Alpert, P. Villarrubia, H. Ren, and C. Chu, "RQL: global placement via relaxed quadratic spreading and linearization," DAC '07, pp. 453–458, 2007.

[7] M.-C. Kim, D.-J. Lee, and I. L. Markov, "SimPL: an effective placement algorithm," ICCAD '10, pp. 649–656, 2010.

[8] M.-C. Kim and I. L. Markov, "ComPLx: A competitive primal-dual lagrange optimization for global placement," DAC '12, pp. 747–752, 2012.

[9] M.-C. Kim, J. Hu, D.-J. Lee, and I. L. Markov, "A SimPLR method for routability-driven placement," ICCAD '11, pp. 67–73, 2011.

[10] X. He, T. Huang, L. Xiao, H. Tian, G. Cui, and E. F. Y. Young, "Ripple: an effective routability-driven placer by iterative cell movement," ICCAD '11, pp. 74–79, 2011.

[11] M.-C. Kim, N. Viswanathan, C. J. Alpert, I. L. Markov, and S. Ramji, "MAPLE: multilevel adaptive placement for mixed-size designs," ISPD '12, pp. 193–200, 2012.

[12] Y. Saad, *Iterative Methods for Sparse Linear Systems*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2nd ed., 2003.

[13] C. Li, M. Xie, C.-K. Koh, J. Cong, and P. H. Madden, "Routability-driven placement and white space allocation," in *Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design*, ICCAD '04, pp. 394–401, 2004.

[14] S. Goto, "An efficient algorithm for the two-dimensional placement problem in electrical circuit layout," *IEEE Trans. Circuit and Systems*, vol. 28, no. 1, pp. 12–18, 1981.

[15] G.-J. Nam, C. J. Alpert, P. Villarrubia, B. Winter, and M. Yildiz, "The ISPD2005 placement contest and benchmark suite," ISPD '05, pp. 216–220, 2005.

[16] M. Pan, N. Viswanathan, and C. Chu, "An efficient and effective detailed placement algorithm," ICCAD '05, pp. 48–55, 2005.

[17] http://software.intel.com/en-us/intel-mkl.

[18] T.-C. C. et al, "NTUPlace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints," *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 27, no. 7, pp. 1228–1240, 2008.

[19] T. F. Chan, J. Cong, J. R. Shinnerl, K. Sze, and M. Xie, "mPL6: enhanced multilevel mixed-size placement," ISPD '06, pp. 212–214, 2006.