SkyPlace: A New Mixed-size Placement Framework using Modularity-based Clustering and SDP Relaxation

Jaekyung Im and Seokhyeong Kang
Pohang University of Science and Technology, Pohang, Korea
{jkim97,shkang}@postech.ac.kr

ABSTRACT

Electrostatics-based placement has made a great success and inspired many placement algorithms. However, the recent direction of improvement is missing two important problems for mixed-size placement − 1) how to initialize placement and 2) how to handle large macros in the analytical placement. In this paper, we propose our new mixed-size placer, *SkyPlace* which is enhanced by novel placement initialization using macro-aware clustering and semidefinite programming. Experimental results show that SkyPlace clearly outperforms the leading-edge placer on academic benchmarks.

1 INTRODUCTION

The goal of VLSI placement is finding the best positions of cells with the objective of achieving better wirelength, signal delay and routability. Abundant previous studies [1] - [12] demonstrated that the analytical placement can handle modern mixed-size designs that include thousands of large macro cells. Since all the analytical placers basically use a gradient-based solver to iteratively find the optimal solution, finding a differentiable formulation of cell density is very important. Compared with the penalty function heuristics [1, 2], electrostatics-based placers [3] - [12] can compute the smoothed cell density in a decent way by assuming each instance has electrical charge and solving the Poisson's equation.

Though these *ePlace-family* have made a great progress in the placement research, they commonly have two critical drawbacks.
Placement initialization : Since the nonlinear placement is a non-convex problem, the convergence quality is highly sensitive to its initial solution, especially for mixed-size designs. However, only a few works have addressed the placement initialization problem. The minimum quadratic wirelength method used in [3] - [8] turned out to have no benefits compared to the random initialization [11, 14]. Fogaca *et al* [15] demonstrated that the modularity-based clustering can obtain more placement-relevant clustering and proposed a *blob placement* flow that regards the location of each cluster as a seed for placement. However, blob placement focused on just rapid evaluation of the placement rather than improving the core placement algorithm, as mentioned in their paper. Chen *et al* [14] formulated the placement initialization problem as a quadratic-constrained quadratic programming problem and proposed a novel method to solve it. However, their method suffers extremely large runtime overhead and does not take into account large macro cells.

DAC '24, June 23–27, 2024, San Francisco, CA, USA
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0601-1/24/06.
https://doi.org/10.1145/3649329.3655984

Table 1: Summary of previous methods.

Placer	Mixed-size Placement	GPU Acceleration	Key Features
ePlace [3] - [5]	✓	✗	Nesterov Opimization, Preconditioning
Pplace [6] - [7]	✓	✗	Analytical Poisson Solver Fast Computation Scheme
RePlAce [8]	✓	✗	Routability optimization, Dynamic Step Size, Local Density
DREAMPlace [9] - [11]	✓	✓	GPU Acceleration, Deep Learning Toolkit
Xplace [12]	✗	✓	Speed-up Techniques, Neural Network Plug-in
SkyPlace (Proposed)	✓	✓	**Hypergraph Clutering, SDP-based Initialization, Dynamic Density Weighting**

Handling large macros : Special attention for macros in the analytic placement is quite controversial in the literature. ePlace-MS [5] generalized their electric potential and treated both macros and standard cells equally. However, Agnesia *et al* [26] found that this can lead to large wirelength degradation due to the legalization issues. When legalizing mixed-size designs, macro cells are legalized first while ignoring standard cells [5, 8, 10] and this causes large displacement of standard cells.
To tackle these problems, we propose a new mixed-size placer named *SkyPlace*. SkyPlace uses macro-aware clustering and semidefinite programming relaxation for placement initialization. Experimental results show that SkyPlace clearly improves placement quality on the modern mixed-size placement benchmarks [22] compared to the leading-edge placer, DREAMPlace [11]. SkyPlace also utilizes GPU acceleration, so it shows comparable runtime with [11]. Table 1 summarizes the main differences between SkyPlace and previous works. Our key contributions are as follows.

- We integrate the modularity-based clustering algorithm [18] and a clustering refinement procedure into our framework, thereby reducing the problem size without loss of detailed view on large macro cells. **[Section 3.1, 3.2]**.
- We propose a novel placement initialization method based on semidefinite programming (SDP). To handle the non-convexity of quadratic constrained quadratic programming (QCQP)-based placement initialization, we relax the QCQP into a SDP problem, which is a convex optimization problem. **[Section 3.3]**.
- To properly estimate the spatial effect of large macro cells, we propose dynamic density weighting technique that imposes high density penalty on macro cells and gradually reduces the penalty near the convergence. **[Section 3.4]**.

In addition to the algorithmical contributions, SkyPlace is developed only with C++/CUDA unlikely to other PyTorch-based placers [11, 12]. We believe this will be a strong implementation baseline of GPU-accelerated placer for the open-source EDA community.

2 PRELIMINARIES

2.1 Analytical Placement

For a hypergraph $G_H(V, E)$ which represents the netlist, the analytical placement problem is defined as:

$$\min_{x,y} \sum_{e \in E} W_e(x, y) \text{ s.t. } D_b(x, y) \leq \rho_t \ , \forall b \in B, \quad (1)$$

where $W_e(\cdot)$ is wirelength for a net e and $D_b(\cdot)$ is the density of each bin b, which has to be less than the target density ρ_t. Since routing information is usually not accessible during placement, the weighted average (WA) wirelength [16], which is a differentiable nonlinear model to approximate HPWL is widely used.

To obtain smoothed density function, *ePlace* treats cell instances as electrical charges and solves Poisson's equation to obtain potential energy ψ_b for each bin b. ψ_b is used to capture the bin density D_b; thus we can obtain an evenly distributed placement. Finally, a penalty parameter λ is used to transform the problem of (1) into unconstrained optimization problem:

$$\min_{x,y} \sum_{e \in E} W_e^{WA}(x, y) + \lambda \sum_{b \in B} \psi_b(x, y) \quad (2)$$

2.2 Modularity-based Clustering

When clustering the graph, modularity [17] is a metric of clustering quality that measures the total expected number of internal edges within each cluster. For a graph $G(V, E)$ and its cluster set C, modularity is defined as :

$$Q = \frac{1}{2|E|} \sum_{i \in V} e_{i \to C(i)} - \frac{1}{4|E|^2} \sum_{c \in C} \Big(\sum_{e \in c} w_e\Big)^2 , \quad (3)$$

where $e_{i \to C(i)}$ is the weight sum of internal edges. Fogaca *et al* [15] showed that the modularity is helpful to find placement-relevant clustering, where the cells within each cluster remain more closer when compared to other clustering methods (e.g. min-cut partitioning). The *Louvain clustering* [18] iteratively finds the best clustering of each vertex that maximizes modularity and aggregates each vertex within cluster until no further improvement can be achieved.

2.3 Placement Initialization

For a given graph $G(V, E)$, let x_m be movable cells, x_f be fixed cells. The total quadratic wirelength is computed as below (y is omitted for simplicity):

$$\begin{aligned} &\sum_{e_{ij} \in E} w(e_{ij})\,(x_i - x_j)^2 = x^T L x \\ &= \begin{bmatrix} x_m & x_f \end{bmatrix}^T \begin{bmatrix} L_{mm} & L_{mf} \\ L_{fm} & L_{ff} \end{bmatrix} \begin{bmatrix} x_m & x_f \end{bmatrix} \\ &= x_m^T L_{mm} x_m + 2x_f^T L_{fm} x_m + constant , \end{aligned} \quad (4)$$

where $L = D - A$ is the *Graph Laplacian* (D and A is degree matrix and weighted adjacency matrix, respectively). L_{mm} represents the connection information between movable cells and the same convention is applied for L_{mm}, L_{mf} and L_{fm}. Though the minimizer of (4) can be obtained by simply solving $L_{mm}x_m = -L_{mf}x_f$, previous studies [11,14] showed that only pursuing minimum quadratic wirelength does not produce a good initial placement. Chen *et al* [14] formulated a QCQP problem by adding quadratic constraints to spread out the cells and linear constraints to center the cells as below (assume that the center coordinate of layout is zero):

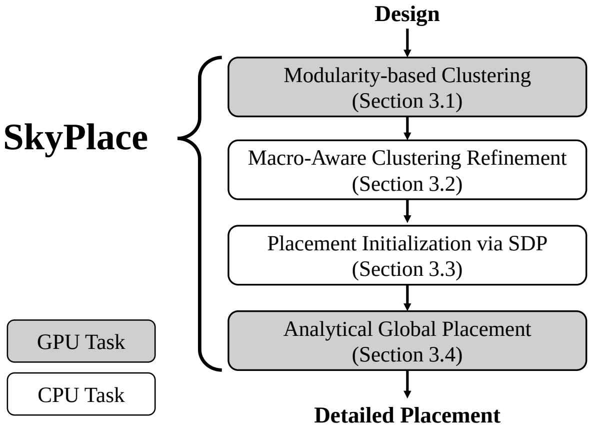
Design
SkyPlace
Modularity-based Clustering (Section 3.1)
Macro-Aware Clustering Refinement (Section 3.2)
Placement Initialization via SDP (Section 3.3)
Analytical Global Placement (Section 3.4)
GPU Task
CPU Task
Detailed Placement

Figure 1: The proposed mixed-size placement framework.

$$\textbf{QCQP} \quad \begin{aligned} &\min_{x_m} \quad x_m^T L_{mm} x_m + 2x_f^T L_{fm} x_m \\ &s.t. \quad v^T x_m = 0, \ \ x_m^T \ diag(v) \ x_m = k, \end{aligned} \quad (5)$$

where v is the vector of cell area and k is a constant to handle the variance of coordinates. Though the problem is now non-convex due to the sphere constraint, solving (5) can provide more pre-optimized initial solution to the placer [14].

3 PROPOSED METHOD

The overall flow of SkyPlace is illustrated in Figure 1. We first break down the netlist into a clustered netlist using GPU-accelerated Louvain clustering [19]. The clustered netlist is subsequently refined to produce more "reasonable" clustering. We then determine the positions of each cluster via SDP-based initialization, followed by analytical mixed-size global placement.

3.1 GPU-Accelerated Hypergraph Clustering

We first transform the hypergraph netlist into a clique-graph, which enables us to apply the graph clustering algorithm. Motivated by [15], we choose the Louvain clustering [18] for our clustering algorithm. SkyPlace uses a GPU-accelerated version [19] to minimize runtime overhead during the pre-placement stages. To briefly introduce the parallelization scheme, in the refinement phase, multiple threads are assigned to each vertex and the threads find the best move by computing the gain of moving the vertex to other clusters. In the aggregation phase, multiple threads are assigned to each cluster to reconstruct the cluster graph while summing up the edge weights in parallel. Since more details are beyond the scope of this paper, interested readers are referred to [19].

3.2 Macro-Aware Clustering Refinement

After clustering, we obtain a graph of clusters G_C. The number of clusters becomes the matrix dimension for SDP solver that will be explained later. If there are thousands of clusters in G_C, it will make the SDP problem intractable. Therefore, we must reduce

Algorithm 1: Macro-Aware Cluster Refinement

```
Input  : Graph of clusters $G_C$
Output : Refined graph of clusters $G$
1  Initialize $clusterList \leftarrow \phi$
2  foreach $cluster\ c \in G_C$ do
3      if $c.num\_macro > num\_max\_macro$ then
4          $\{c_{partition}\} \leftarrow minCutPartitioning(c)$ [20]
5          $clusterList.insert(c_{partition})$
6      end
7      else if $|c| \leq num\_min\ and\ c.num\_macro \neq 0$ then
8          $removeCluster(c)$
9      end
10     else if $c\ has\ a\ set\ of\ too\ large\ macros\ M$ then
11         foreach $macro\ m \in M$ do
12             $clusterList.insert(m)$
13         end
14     end
15     else
16         $clusterList.insert(c)$
17     end
18 end
19 $G \leftarrow$ merge each cluster of $clusterList$
20 return $G$
```

the size of G_C but it is also important not to degrade clustering quality. Furthermore, the vanilla Louvain clustering method does not differentiate between large macros and small standard cells, thus we have to refine the clustering to get more detailed view on macro cells (Algorithm 1).

- If there are too many macros within a cluster, we call a simple min-cut partitioner [20] to break down the cluster (Lines 3 - 5).
- If a cluster has only few standard cells, remove the cluster (Lines 7 - 9). We have observed that this can remove 60% ~ 80% of clusters in G_C (depending on *num_min*).
- If a cluster has too large macros, make a new cluster with each macro (Lines 10 - 14). We define a macro is *too large* macro if its width (height) is larger than 20% of the die width (height).

We ignore highly-connected nets that have 500+ pins during clique decomposition, which makes *floating* cells that are not connected to any net. Each floating cell becomes a single cluster by Louvain clustering, but these tiny clusters will be removed from G_C during the refinement (thereby ignored during placement initialization and just placed in the center of die). As a result of the refinement, G_C shrinks to only hundreds of clusters, which is tractable size for the SDP solver. We set *num_max_macro* as 100 and *num_min* as 5.

3.3 Placement Initialization via SDP

Regarding each cluster as a vertex, we use the QCQP formulation of (5) to determine the positions of clusters. Our method relaxes the non-convex QCQP problem into a SDP problem, which is a

Algorithm 2: SDP-based Initialization

```
Input  : Refined graph of clusters $G$,
         Area vector of clusters $v$,
         Coordinates of fixed vertices $x_f, y_f$
Output : Initial coordinates of the clusters $x_{init}, y_{init}$
1  Create graph laplacian $L_x$, $L_y$ with $G, x_f, y_f$
2  $k_x \leftarrow \alpha_x \cdot A \cdot W^2,\quad k_y \leftarrow \alpha_y \cdot A \cdot H^2$
3  $X_{SDP} \leftarrow$ solveSDP$(L_x, x_f, v, k_x)$
4  $Y_{SDP} \leftarrow$ solveSDP$(L_y, y_f, v, k_y)$
5  Extract $x_{init}, y_{init}$ from $X^{SDP}, Y^{SDP}$
6  return $x_{init}, y_{init}$
```

well-studied convex optimization problem defined as [21]:

$$\textbf{SDP}\qquad \begin{aligned} \min_{X\in\mathcal{S}_+^n}\quad & Tr(M_0X) \\ s.t.\quad & Tr(M_iX) = k_i,\quad i = 1,2,\ldots \\ & Tr(P_iX) \leq l_i,\quad i = 1,2,\ldots, \end{aligned} \tag{6}$$

where $\mathcal{S}_+^n$ is set of symmetric positive semidefinite matrix. Assuming $X = \begin{bmatrix} 1 \\ x_m \end{bmatrix} \begin{bmatrix} 1 & x_m^T \end{bmatrix}$, we can relax the QCQP formulation of (5) to the SDP formulation of (6). The relaxation procedure starts from converting the QCQP formulation of (5) into a form of:

$$\begin{aligned} \min_{X\in\mathcal{S}_+^n}\quad & Tr(M_0X) \\ s.t.\quad & Tr(M_1X) = k,\ \ Tr(M_2X) = 0,\ \ Tr(M_3X) = 0 \\ & M_0 = \begin{bmatrix} 0 & x_f^T L_{fm} \\ L_{mf}x_f & L_{mm} \end{bmatrix},\quad rank(X) = 1 \\ & M_1 = \begin{bmatrix} 0 & \mathbf{0}^T \\ \mathbf{0} & diag(v) \end{bmatrix},\ M_2 = \begin{bmatrix} 0 & v^T \\ v & \mathbf{0}\mathbf{0}^T \end{bmatrix},\ M_3 = \begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{0} & \mathbf{0}\mathbf{0}^T \end{bmatrix} \end{aligned} \tag{7}$$

Theorem 1. *Suppose* $rank(X) = 1$ *and* $X = \begin{bmatrix} 1 & x_m^T \end{bmatrix}^T \begin{bmatrix} 1 & x_m^T \end{bmatrix}$. *Then the problem of (5) is equivalent to the problem of (7).*

Proof. For brevity, let $x_m = x$, $L_{mm} = L$ and $L_{mf}x_f = b$.

$$\begin{aligned} Tr(x^TLx + 2b^Tx) &= Tr(Lxx^T + b^Tx + x^Tb) \\ &= Tr(\begin{bmatrix} 0 & b^T \\ b & L \end{bmatrix} \begin{bmatrix} 1 & x^T \\ x & xx^T \end{bmatrix}) = Tr(M_0X) \end{aligned} \tag{8}$$

Without loss of generality, we can obtain M_1, M_2 and M_3 in the similar way. Due to the constraint of $Tr(M_3X) = 0$, X_{00} is forced to be 1. Since X is a *rank*-1 matrix, we can always retrieve the optimal x from the optimal X of the problem of (7) and the proof is finished. □

If we drop out the *rank*-1 constraint, (7) is now in the form of SDP in (6). We can now apply our SDP-based initialization method (Algorithm 2). We first create graph laplacian matrix L_x, L_y with the given graph (Line 1). k_x, k_y are parameters corresponding to k in (7), where A is sum of total cluster area and $W(H)$ is width (height) of the die (Line 2). Larger α_x, α_y means clusters are more likely to spread out across the die. We used 0.4 for both α_x and α_y. We now solve the SDP problem using a well-designed solver [23] (Lines 3-4). Note that X^{SDP}, Y^{SDP} are the optimal solutions due to the convexity of SDP.

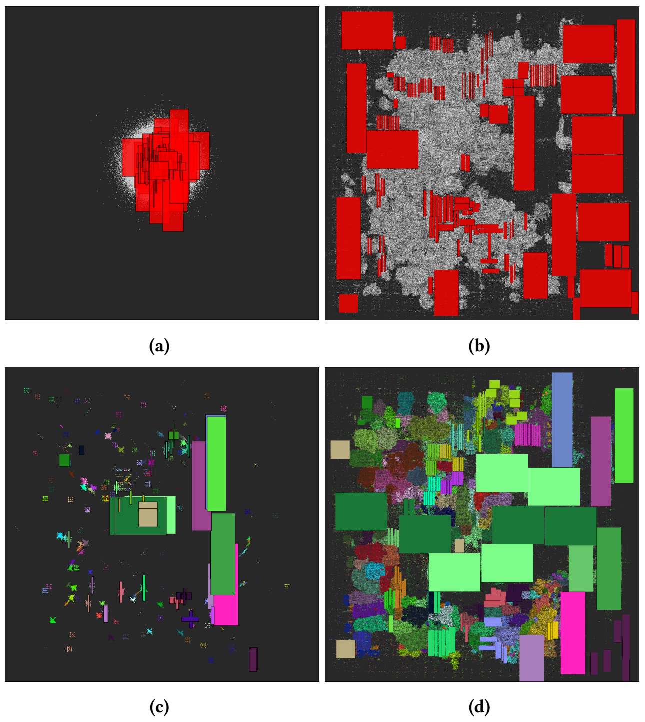

Figure 2: Placement of newblue7. (a) Random initialization with normal distribution. (b) Final results from random initialization (HPWL = 977.7). (c) SDP-based initialization (colored clusters). (d) Final results from SDP-based initialization (HPWL = 863.0).

To retrieve original x, it is necessary to extract the largest eigen pair from the low rank approximation of resulting X [27]. However, it is time-consuming to perform additional linear algebraic operation on a large matrix since low rank approximation requires singular value decomposition. Therefore, we assume resulting X^{SDP} is in the form of $X = \begin{bmatrix}1 & x_m^T\end{bmatrix}^T \begin{bmatrix}1 & x_m^T\end{bmatrix}$ (though this is not guaranteed). Then, we extract x_{init} from X^{SDP}_{10} to $X^{SDP}_{|x_m|0}$ (Line 5). Though x_{init}, y_{init} is not the exact solution of (5) anymore, experimental results show that these are sufficient to produce high-quality initial placement.

As mentioned earlier, the non-convex nature of nonlinear placement makes it very sensitive to the initial solution. Especially when it comes to mixed-size designs which have a sea of macros, the movement of macro cell is highly dependent to its initial position, making the placer easily fall into a local optimal. Figure 2 well describes this problem. Both Figure 2(a) and Figure 2(c) show the initial placement using random variable from normal distribution and the solution from SDP, respectively. As shown in Figure 2(b) and Figure 2(d), the final layouts have dramatically different HPWL results with each other, as well as the macro locations. Since our SDP-based initialization guarantees the *optimal* solution of cluster positions, we can avoid bad initial solutions misguide the placer and ruin the final placement quality.

The proposed method is different from a similar previous work [28] in several aspects. First, the derivation of SDP formulation is different. They assumed each module as a shape of circle and formulated a overlap constraint while our method starts from the QCQP formulation of [14]. Also, the main goal of [28] is more like a floorplanning of large modules, rather than initialization of mixed-size placement. Therefore, they lack consideration of large number of standard cells that are relatively small.

3.4 Global Placement with Density Weighting

For the global placement, SkyPlace utilizes common features of electrostatics-based placers [3] - [12], including the Nesterov optimizer [13] to solve (2). For GPU acceleration, we use parallelization techniques proposed in [11, 25]. The main difference in the global placement part is that we are giving higher density weight to macros. In the *eDensity* formulation [3], the electrical density ψ_b is obtained by solving the Poisson's equation below:

$$\nabla \cdot \nabla \psi_b(x, y) = -\rho_b(x, y), \tag{9}$$

where $\rho(x, y)$ represents the cell density of a bin $b \in B$ and (x, y) is the index. For brevity, we omit the boundary conditions. A visualization of ψ_b is attached in Figure 3 to make better understanding.

Obviously, highly congested bin will make larger electrical potential, thereby affected by more stronger electrical force. eDensity can successfully help us spread out placement instances when it comes to designs that only have standard cells. However, when it comes to the mixed-size designs that have a great imbalance in the cell sizes, eDensity often fails to precisely capture the overlap among instances. Figure 4 exemplifies this problem. The figures on the left shows the global placement results without density weighting. We can found that there are many overlaps of instances around the macros. Though cell overlaps will be removed by subsequent legalization step, too many overlaps in the global placement cause large displacement, which in turn degrade final wirelength. We believe that this is due to "underestimation" on the density effect of macros. To address this problem, we will change the way bin density is calculated. SkyPlace defines the bin density as:

$$\rho_b(x, y) = \sum_{c \in b} w_d(c) \cdot l_x(c, b) \cdot l_y(c, b), \tag{10}$$

where $c \in b$ means that the cell c has overlap with bin b and $l_x(c, b), l_y(c, b)$ denotes the overlapped length across x, y direction between a cell c and a bin b. More detailed explanations on l_x, l_y are referred to [3]. $w_d(c)$ is the newly introduced variable, which means the density weight of a cell c:

$$w_d(c) = \begin{cases} w_{macro} & \text{if } c \text{ is a macro cell} \\ 1 & \text{otherwise} \end{cases} \tag{11}$$

To properly estimate the spatial impact of macro cells, we give higher density weight ($w_{macro} \geq 1$) to macro cells. By doing this, the placer will put more efforts for removing the overlap area among the macros cells. However, it is obvious that "overestimation" of macro density will also make quality degradation and thus we multiply a decay term ϵ (< 1.0) for every placement iteration k.

$$w^{k+1}_{macro} = w^k_{macro} \times \epsilon \tag{12}$$

(10) makes the placer impose higher penalty on macro density to avoid the small standard cells are trapped in the center of large macro cells in the initial steps. The penalty is relaxed as the placement goes to convergence. This method is similar to *macro halo* used by AutoDMP [26] because both are trying to insert white space

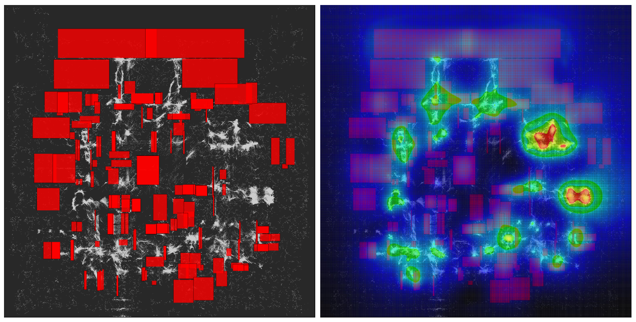

Figure 3: A distribution of cells (left) and its heat map of electrical potential ψ_b (right).

around macros and minimize the gap between global placement results and post-legalization results. However, macro halo is more likely to a static method because the size of halo cannot be changed anymore once it is inserted. On the other hand, our density weighting technique can control the balance between macro and standard cell density dynamically during the placement.

The right figures of Figure 4 shows the impact of dynamic density weighting, where standard cells are pushed away from the center of large macros and thus the overlapped area with macros cells is significantly reduced. We empirically find the best w_{macro} and ϵ in the range of (1.01, 1.10) and (0.999, 0.9999). In our experiences, higher w_{macro} makes the placer more difficult to reach the stopping overflow. Therefore, smaller ϵ is preferred for high w_{macro}.

4 EXPERIMENTAL RESULTS

4.1 Experimental Setup

SkyPlace is implemented with approximately 19K lines of C++ and CUDA, including its own parsers and database. For the SDP solver, we use academic license of *Mosek Fusion* C++ API [23]. We compare the experimental results with DREAMPlace [11], which is the state-of-the-art GPU-accelerated placer. The benchmarks that we use are modern mixed-size (MMS) benchmarks [22]. Our experiments are executed on a Linux machine equipped with NVIDIA GeForce Titan RTX GPU and Intel Xeon Gold CPU 2.9GHz. We use ABCDPlace [24] as our detailed placer. To follow the convention of literature, NTUplace3 [2] binary is used to evaluate HPWL of bookshelf format benchmarks.

4.2 Comparative Study

For ablation studies, we run four different settings (Table 2). The column named "DREAMPlace" shows the results from default DREAMPlace flow for mixed-size placement [10]. "SkyPlace (w/o SDP, DW)" column shows the results without both SDP initialization (SDP) and dynamic density weighting (DW). "SkyPlace (w/o DW)" column shows the results with SDP and "SkyPlace" columns shows the results including both SDP and DW.

The mixed-size placement frequently ends up with divergence and its results are quite noisy due to the large wirelength degradation after legalization. To reduce the noise of HPWL results by minimizing the subsequent displacement, we find the minimum stopping overflow that does not make divergence for SkyPlace. We

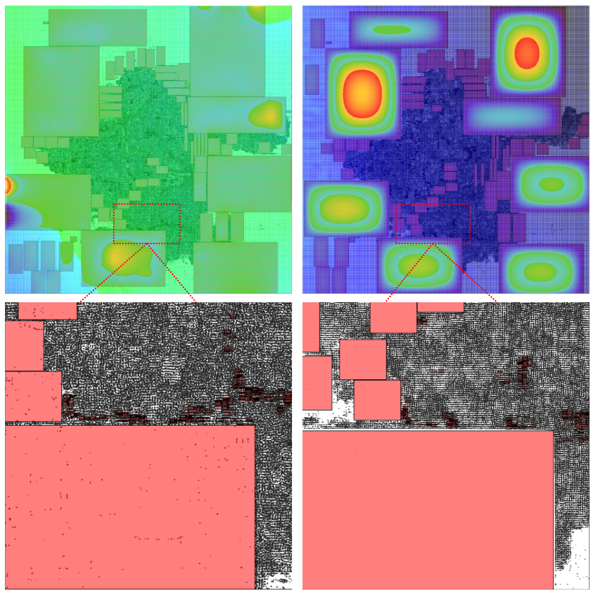

Figure 4: The heatmap of electrical potential ψ_b on bigblue3 (top) and final placement layout (bottom). Results without density weighting (left) and with density weighting (right).

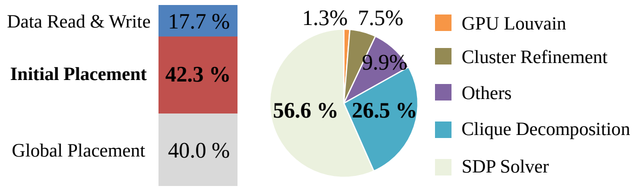

Figure 5: Overall flow runtime breakdown (left) and initial placement runtime breakdown (right).

would like to argue that this does not harm the fairness of comparison because DREAMPlace already has a heuristic to quit placement when divergence is detected and returns the best HPWL results obtained so far. It is worth mentioning that SkyPlace achieves 3.18% improvements on average compared to DREAMPlace, though there are increase of runtime. The detailed runtime breakdown is included in the **Section 4.3**.

4.3 Runtime Breakdown

Figure 5 illustrates the runtime breakdown of SkyPlace on the newblue7 benchmark which is the largest design among MMS benchmarks. Owing to the power of GPU acceleration, Louvain clustering takes up only 1.3% of the initial placement runtime. It is remarkable that solving SDP takes 56.6% of the initial placement runtime. This is because the *Mosek* SDP solver [23] forces us to transform an intrinsic data type of C++ to a specific data structure that fits the solver. We expect optimizing the way of solving SDP in the SkyPlace will reduce the runtime significantly.

Table 2: Post detailed placement HPWL ($\times 10^6$) and runtime (s) results on MMS benchmark [22]. $|V|$, $|V_M|$ and ρ_t mean # total movable instances, # movable macros and target density, respectively. We use ABCDPlace [24] as a detailed placer and NTUplace3 [2] binary for HPWL evaluation. The runtime includes time for database reading and global placement.

Benchmark				DREAMPlace [11]		SkyPlace (w/o SDP, DW)		SkyPlace (w/o DW)		SkyPlace	
Name	**$\lvert V \rvert$**	**$\lvert V_M \rvert$**	**ρ_t**	**HPWL**	**Runtime**	**HPWL**	**Runtime**	**HPWL**	**Runtime**	**HPWL**	**Runtime**
adaptec1	211K	63	100%	66.08	7.7	65.75	**7.0**	65.67	13.5	**65.62**	13.5
adaptec2	255K	127	100%	77.13	**14.2**	**72.98**	19.2	78.04	24.4	73.96	24.5
adaptec3	451K	58	100%	159.69	**16.3**	160.83	24.6	158.52	30.3	**157.09**	29.8
adaptec4	495K	69	100%	141.03	**26.5**	142.48	50.2	142.43	62.6	**142.35**	62.0
adaptec5	843K	76	50%	319.26	**28.4**	**313.68**	35.3	316.13	47.1	315.01	49.3
bigblue1	278K	32	100%	85.39	9.5	85.30	**6.1**	**85.22**	10.4	85.35	10.6
bigblue2	536K	959	100%	**126.53**	17.4	132.66	**10.0**	126.90	60.5	126.84	61.0
bigblue3	1097K	2549	100%	284.11	**44.2**	280.72	96.1	276.43	123.0	**275.67**	128.4
bigblue4	2169K	199	100%	654.57	**77.1**	673.26	101.4	**654.27**	196.4	655.77	187.0
newblue1	330K	574	80%	62.93	**11.5**	62.64	32.6	58.95	48.3	**58.92**	50.4
newblue2	440K	730	90%	157.73	**14.8**	156.95	16.1	155.23	21.3	**155.04**	22.5
newblue3	483K	1318	80%	324.83	**14.0**	295.65	18.9	271.80	42.5	**268.95**	32.6
newblue4	643K	1009	50%	240.02	20.5	232.45	**16.2**	**230.15**	30.4	230.49	33.0
newblue5	1233K	1254	50%	436.23	**45.0**	398.60	75.7	393.65	105.9	**392.75**	115.7
newblue6	1255K	929	80%	409.88	**42.7**	412.72	52.9	409.39	91.6	**408.80**	95.2
newblue7	2508K	1077	80%	895.28	117.1	996.37	**76.0**	890.98	157.3	**886.77**	181.0
Normalized Average				100.00	**100.00**	100.95	125.91	97.14	210.19	**96.82**	216.30

5 CONCLUSION

In this paper, we propose SkyPlace, a new mixed-size placement framework using modularity-based clustering, SDP-based initialization and dynamic density weighting. SkyPlace can solve such problems: (1) misguided mixed-size placement due to bad initial solution and (2) difficulty of handling large macros in the analytic placement. Our comparative study shows that SkyPlace produces better quality of placement compared with the leading-edge placer, DREAMPlace [11]. The source codes of SkyPlace are available in the link: https://github.com/jkim971201/SkyPlace.

ACKNOWLEDGEMENT

This work was supported by Samsung Electronics Co., Ltd and Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT (No. RS-2023-00222609). This material is also based on work that is partially funded by an unrestricted gift from Google.

REFERENCES

[1] A. B. Kahng and Q. Wang, "Implementation and Extensibility of an Analytic Placer", *IEEE TCAD* 24(5) (2005), pp. 734-747
[2] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu et al., "NTUplace3: An Analytical Placer for Large-Scale Mixed-Size Designs With Preplaced Blocks and Density Constraints", *IEEE TCAD* 27(7) 2008, pp. 1228-1240
[3] J. Lu, P. Chen, C.-C. Chang et al., "ePlace: Electrostatics-based Placement using Fast Fourier Transform and Nesterov's Method", *ACM TODAES* 20(2) 2015
[4] J. Lu, P. chen, C.-C Chang et al., "ePlace: Electrostatics based Placement using Nesterov's method", *Proc. DAC*, 2014
[5] J. Lu, H. Zhuang, P. Chen et al., "ePlace-MS: Electrostatics-based Placement for Mixed-Size Circuits", *IEEE TCAD* 34(5) 2015, pp. 685-698
[6] W. Zhu, Z. Huang, J. Chen et al., "Analytical Solution of Poisson's Equation and Its Application to VLSI Global Placement", *Proc. ICCAD*, 2018
[7] K. Peng and W. Zhu, "Pplace-MS: Methodologically Faster Poisson's Equation-Based Mixed-Size Global Placement", *IEEE TCAD* 43(2) 2024, pp. 613-626
[8] C.-K. Cheng, A. B. Kahng, I. Kang et al., "RePlAce: Advancing Solution Quality and Routability Validation in Global Placement", *IEEE TCAD* 38(9) 2018, pp. 1717-1730
[9] Y. Lin, S. Dhar, W. Li et al., "DREAMPlace: Deep Learning Toolkit-enabled GPU Acceleration for Modern VLSI Placement", *Proc. DAC*, 2019
[10] Y. Lin, D. Pan, H. Ren et al., "DREAMPlace 2.0: Open-Source GPU-Accelerated Global and Detailed Placement for Large-Scale VLSI Designs", *Proc. CSTIC*, 2020
[11] Y. Lin, S. Dhar, W. Li et al., "DREAMPlace: Deep Learning Toolkit-enabled GPU Acceleration for Modern VLSI Placement", *IEEE TCAD* 40(4) 2021, pp. 748-761
[12] Y. Lin, S. Dhar, W. Li et al., "Xplace: An Extremely Fast and Extensible Global Placement Framework", *Proc. DAC*, 2022
[13] Y. E. Nesterov, "A Method of Solving the Convex Programming Problem with Convergence Rate $O(\frac{1}{k^2})$", *Soviet Math. Dokl.* 27(2) 1983, pp. 372-376
[14] P. Chen, C.-K. Cheng, A. Chern et al., "Placement Initialization via Sequential Subspace Optimization with Sphere Constraints", *Proc. ISPD*, 2023
[15] M. Fogaça, A. B. Kahng, E. Monteiro et al., "On the Superiority of Modularity-Based Clustering for Determining Placement-Relevant Clusters", *Integration: The VLSI Journal*, 74 (2020), pp. 32-44.
[16] M. Hsu, V. Balabanov and Y. Chang, "TSV-aware Analytical Placement for 3-D IC Designs based on a Novel Weighted-Average Wirelength Model", *IEEE TCAD* 32(4) (2013), pp. 497-509
[17] M. E. Newman, "Modularity and Community Structure in Networks", *Proc. National Academy of Sciences*, 103 (23), 2006, pp. 8577-8582
[18] V. D. Blondel, J. Guillaume, R. Lambiotte et al., "Fast Unfolding of Community Hierarchies in Large Networks", *Journal of Statistical Mechanics: Theory and Experiment*, Vol. 10, 2008, pp. P10008
[19] Md. Naim. F. Mannee, M. Halappanavar et al., "Community Detection on the GPU", *Proc. IPDPS*, 2017
[20] C. M. Fidduccia, R. M. Mattheyses, "A Linear-Time Heuristic for Improving Network Partitions", *Proc. DAC*, 1982
[21] S. Boyd and L. Vandenberghe, "Convex Optimization", Cambridge Press, 2004
[22] J. Z. Yan, N. Viswanathan and C. Chu, "Handling Complexities in Modern Large-Scale Mixed-Size Placement", *Proc. DAC* 2009
[23] MOSEK 10.1, https://www.mosek.com
[24] Y. Lin, W. Li, H. Ren et al., "ABCDPlace: Accelerated Batch-Based Concurrent Detailed Placement on Multithreaded CPUs and GPUs", *IEEE TCAD* 39(12) 2020
[25] Z. Jiang, J. Gu, D. Z. Pan et al., "A New Acceleration Paradigm for Discrete Cosine Transform and Other Fourier-Related Transforms", *arXiv:2110:01172*, 2021
[26] A. Agnesia, P. Rajvanshi, T. Yang et al., "AutoDMP: Automated DREAMPlace-based Macro Placement", *Proc. ISPD*, 2023
[27] J. C. Bedoya, A. Abdelhadi, C-C. Liu et al., "A QCQP and SDP Formulation of the Optimal Power Flow Including Renewable Energy Resources", *Proc. ISSE*, 2019
[28] W. Li, F. Wang, J. M. F. Moura et al., "Global Floorplanning via Semidefinite Programming", *Proc. DAC*, 2023