PPA-Relevant Clustering-Driven Placement for Large-Scale VLSI Designs

Andrew B. Kahng
UC San Diego
abk@ucsd.edu

Seokhyeong Kang
POSTECH
shkang@postech.ac.kr

Sayak Kundu
UC San Diego
sakundu@ucsd.edu

Kyungjun Min
POSTECH
kj.min@postech.ac.kr

Seonghyeon Park
POSTECH
seonghyeon98@postech.ac.kr

Bodhisatta Pramanik
UC San Diego
bopramanik@ucsd.edu

ABSTRACT

Today's place-and-route (P&R) flows are increasingly challenged by complexity and scale of modern designs. Often, heuristics must trade off between turnaround time and quality of PPA outcomes. This paper presents a clustered placement methodology that improves *both* turnaround time and final-routed solution quality. Our *PPA-aware* clustering considers timing, power and logical hierarchy during netlist clustering, effectively reducing problem size and accelerating global placement runtime while improving post-route PPA metrics. Additionally, our machine learning (ML)-accelerated *virtualized P&R* methodology predicts the best cluster shapes (i.e., aspect ratios and utilizations) to use in P&R of the clustered netlist. With the open-source OpenROAD tool, our methods achieve up to 47% (average: 36%) global placement runtime improvement with similar half-perimeter wirelength (HPWL) and 90% (29%) improvement in post-route total negative slack (TNS). With the commercial Cadence Innovus tool, our methods achieve up to 3.92% (1%) improvement in power and 99% (49%) improvement in TNS.

ACM Reference Format:
Andrew B. Kahng, Seokhyeong Kang, Sayak Kundu, Kyungjun Min, Seonghyeon Park, and Bodhisatta Pramanik. 2024. PPA-Relevant Clustering-Driven Placement for Large-Scale VLSI Designs. In *61st ACM/IEEE Design Automation Conference (DAC '24), June 23–27, 2024, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3649329.3655991

1 INTRODUCTION

The placement phase of physical design is central to optimization of performance, power and area (PPA) outcomes, as well as to design space exploration at floorplan/RTL levels and above. Complexity and scale have rapidly increased, such that millions of instances must be efficiently placed within stringent runtime limits. Thus, placement tools rely on heuristics that are often challenged by problem scale, and by the tension between improvement of turnaround time (TAT) and improvement of quality of results (QOR). *Clustering* has long been seen as a solution to these challenges [20], [11], [9]. However, traditional clustering heuristics [12], [6] only optimize a cutsize criterion and do not consider design information (logical hierarchy, timing, switching activity, etc.) that strongly affects PPA outcomes. Recent works, such as [9] and [14], revisit the use of netlist clustering to guide and improve placement flows. These works demonstrate that clustering can either reduce runtime but with PPA degradation [9], or improve PPA but with runtime degradation [14]. By contrast, our present work develops a *PPA-aware* clustering methodology, and an improved clustered placement approach based on machine learning (ML)-accelerated *virtualized place-and-route* (V-P&R), to improve *both* runtime and PPA relative to baseline (academic and commercial) flat placement methods.

Following are the key contributions made by our work.

- ***PPA-aware* clustering.** We consider additional netlist information – logical hierarchy, timing criticality of paths, and switching activity of nets – to achieve *PPA-aware* clustering. In doing so, we (i) apply a dendrogram-based approach to extract clusters from the netlist logical hierarchy, and (ii) enhance the multilevel clustering framework of [29] [5] to handle logical hierarchy and switching activity of nets. We experimentally demonstrate that our clustering approach achieves noteworthy PPA benefits, and that it outperforms traditional clustering methods when applied in OpenROAD and Cadence Innovus flows (Sections 3.1 and 4).
- **ML-accelerated virtualized P&R.** In the *seeded placement* approach, a *seed placement* of clusters is used to induce *seed locations* of instances, from which the flat P&R flow is continued. Obtaining a high-quality seed placement of clusters requires two elements: how to form the clusters, and how to feed the clusters into a placer. To this end, we use our *PPA-aware* clusters, and a novel V-P&R framework to determine cluster shapes (utilizations and aspect ratios) to use in the cluster placement. We accelerate the V-P&R framework using a graph neural network (GNN)-based ML model that achieves *mean absolute error (MAE)* of 0.131 (for label values in the range [0.564, 2.96]) and *R2 score* of 0.638, (Sections 3.2 and 4.4).

Experimental confirmations. We evaluate our PPA-aware clustering methodology and ML-accelerated V-P&R framework using both OpenROAD and Innovus flows, along with open testcases from the TILOS MacroPlacement [28] and OpenROAD-flow-scripts [27] GitHub repositories. Our methods achieve (maximum, average) percentage improvements of (47, 36) in global placement runtime with similar half-perimeter wirelength (HPWL), compared to the default (i.e., without any clustering or V-P&R) OpenROAD flow. We also achieve (maximum, average) percentage improvements of (4, 1) and (99, 49) in power and post-route total negative slack (TNS), respectively, compared to a standard Innovus flow. To the best of our knowledge, we are the first work to improve *both* global placement runtime and post-route final PPA simultaneously (Section 4).

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06
https://doi.org/10.1145/3649329.3655991

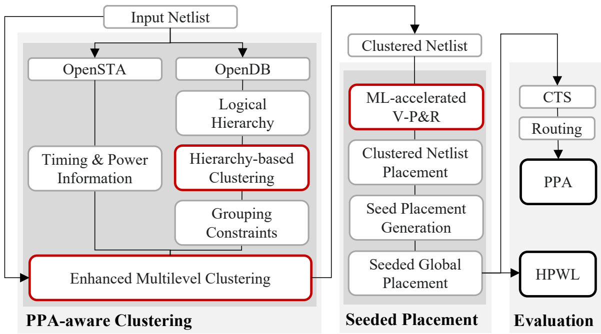

Figure 1: Our overall approach, including *PPA-aware* clustering and the ML-accelerated V-P&R framework.

2 RELATED WORK

Clustering has been widely used in various stages of VLSI physical design, such as partitioning [5], placement [17] and clock tree synthesis [18]. Popular hypergraph clustering heuristics include First Choice (FC) [12], Best Choice (BC) [1] and cut-overlay [6]. FC and BC find clusters of vertices with stronger intra-cluster connectivity compared to inter-cluster connectivity. The cut-overlay method combines multiple clustering solutions to generate better clusters. These methods predominantly rely on local criteria when finding candidate vertices to cluster. By contrast, community detection algorithms such as *Louvain* [4] and *Leiden* [19] adopt a more global perspective, identifying clusters that maximize a *modularity* score.

Several works show benefits of clustering in global placement. [11] proposes a clustering metric that correlates well with post-place wirelength. When applied in clustering-based placement, the metric speeds up placement generation, albeit with some wirelength degradation. [2] integrates BC clustering with placement and demonstrates superior performance over edge-coarsening (EC) and FC. When compared to a standard flow, BC-based placement achieves similar-quality HPWL but with improved placement speed. Similarly, [20] uses clustering in a fast placement methodology. More recently, [9] has proposed a 'blob placement' strategy where placement-relevant clusters are found using Louvain clustering. Integration with RePlAce [7] achieves faster placement runtimes with minor HPWL degradation. Last, the authors of [14] propose an ML-driven clustering framework for PPA optimization. By representing PPA metrics as optimizable loss functions, their method achieves improved PPA results albeit with increased runtime.

With the exception of [14], the above-mentioned works focus on improving global placement runtime. Other gaps are apparent in the literature, e.g., some of the clustering methods used (e.g., BC and Louvain/Leiden) do not scale to large design sizes. Moreover, previous clustering criteria based on cutsize and/or modularity are not well-correlated with PPA outcomes. While [14] proposes a clustering framework that considers additional netlist information for PPA optimization, their approach requires a placed netlist as an input and is runtime-intensive. Our present work addresses these gaps by incorporating PPA-aware clustering and ML-accelerated V-P&R within a clustering-based placement framework that improves both turnaround time and PPA outcomes over strong baseline methods.

3 OUR APPROACH

Our clustering-based placement approach is detailed in Algorithm 1. Figure 1 illustrates the two main components: (i) *PPA-aware* clustering and (ii) ML-accelerated virtualized P&R. The input is a netlist

Algorithm 1: Our overall approach.

Inputs: *Netlist* (.v, .lib, .lef, .def, .sdc), $|P|$ (number of paths), *Tool*
Outputs: *HPWL, rWL, WNS, TNS, Power*

1 `/* Extract logical hierarchy, timing, and power info */`
2 **if** logical hierarchy is present **then**
3 $T(V', E')$ ← Read logical hierarchy using OpenDB and generate logical hierarchy tree
4 P ← Extract top $|P|$ timing paths using OpenSTA
5 S ← Extract switching activity of all nets using OpenSTA
6 C ← Run hierarchy-based clustering of T using Algorithm 2
7 $Cmty$ ← Generate grouping constraints based on C
8 `/* PPA-aware clustering */`
9 C_{enh} ← Run enhanced multilevel clustering using P, S and $Cmty$
10 $Netlist_{clust}$ ← Generate clustered netlist from C_{enh}
11 `/* ML-accelerated V-P&R */`
12 C_{shapes} ← Generate cluster shapes with ML-accelerated V-P&R for clusters containing more than 200 instances
13 $Netlist_{clust}$ ← Create .lef using C_{shapes}
14 `/* Seeded placement */`
15 **if** *Tool == Innovus* **then**
16 Pl_{clust} ← Run placement on $Netlist_{clust}$
17 Place instances in $Netlist$ at their respective cluster centers
18 Build region constraints using Pl_{clust} and C_{shapes}
19 Run place_design *-incremental*
20 Remove region constraints
21 **else if** *Tool == OpenROAD* **then**
22 $Netlist_{clust}$ ← Scale weights on IO nets by 4
23 Run placement on $Netlist_{clust}$
24 Place instances in $Netlist$ at their respective cluster centers
25 Run globalPlacement *-incremental*
26 `/* Placement evaluation */`
27 *HPWL* ← Record HPWL
28 Run CTS and route $Netlist$
29 *rWL, WNS, TNS, Power* ← Record post-route wirelength, worst negative slack, total negative slack and power from *Tool*
30 **return** *HPWL, rWL, WNS, TNS, Power*

file (.v, .lib, .lef, .def[1], .sdc) and a choice of implementation tool (in this work, either *OpenROAD* [25] or *Cadence Innovus* [23]).

Lines 2-7: We use OpenDB [25] to parse the input netlist, extract the logical hierarchy and construct a hierarchy tree T that captures the hierarchical relationship of instances in the netlist. We then use T and Algorithm 2 to find a hierarchy-based clustering of the instances. These clusters are used to induce grouping constraints [5]. We also extract the top P most critical timing paths, along with vectorless switching activity of nets, using OpenSTA [26].

Lines 9-10: *Enhanced multilevel clustering* adds PPA-awareness to the open-source FC implementation of [29]. Hierarchy-based grouping constraints and path timing criticality (slacks) are applied similarly to [5]. We also introduce hyperedge *switching costs* to distinguish nets with high switching activity.[2]

Lines 12-13: Given the clustered netlist, we estimate the best choices of individual cluster shapes (aspect ratios and utilizations) using an ML-accelerated V-P&R framework (Section 3.2). We run ML-accelerated V-P&R only for clusters that contain more than 200 instances.[3] Cluster shapes are then updated in the cluster .lef file.

Lines 15-25: The clustered netlist is placed to obtain a *seed placement*, according to the choice of *Tool*. The coordinates of the seed placement are then used to induce a flat *seeded placement*. When *Tool* is Innovus, we generate the *seeded placement* through a three-step process (Lines 16-20): (i) placing all instances in a cluster at the cluster center; (ii) setting region constraints for clusters whose

[1]The .def file provides floorplan bounding box, pin placements and macro preplacements.
[2]The enhanced FC-based multilevel clustering often generates several singleton clusters. Our background studies show that merging these singleton clusters into a single larger cluster can significantly degrade post-route PPA, hence we do not merge them.
[3]Hyperparameter tuning studies established that a lower bound of 200 instances leads to the best PPA outcomes.

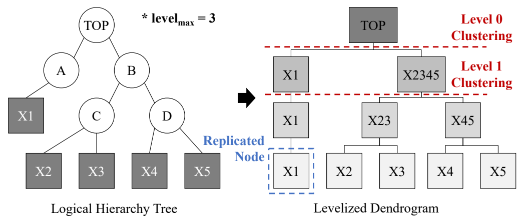

Figure 2: Illustration of hierarchy-based clustering.

shapes were estimated using ML-accelerated V-P&R; and (iii) running incremental placement. When *Tool* is OpenROAD (Lines 22-25), we first scale IO net weights by 4 [9], then generate the *seeded placement* using the above steps (i) and (iii) only, since OpenROAD cannot handle region constraints.

Lines 27-30: To assess quality of the *seeded placement*, we collect post-place wirelength (HPWL), then execute CTS and routing to obtain post-route PPA metrics: routed wirelength (rWL), worst negative slack (WNS), total negative slack (TNS) and total power.

3.1 *PPA-aware* Clustering

Our *PPA-aware* clustering considers logical hierarchy, timing and power information in addition to physical connectivity. Logical hierarchy is obtained using *OpenDB*, while timing and power information is obtained using *OpenSTA*.

Logical hierarchy. Netlist clustering based on logical hierarchy is natural to consider, since functionally 'similar' or 'related' instances are often in spatial proximity in the final placement. However, clustering based only on the logical hierarchy can lead to suboptimalities, since other factors such as timing or connectivity also affect the placement and hence the final PPA. Thus, we use the hierarchy-based clusters as clustering guides (or, grouping constraints), as in [5]. Algorithm 2 formally describes our hierarchy-based clustering; see also Figure 2. Additional details are as follows.

Lines 2-5: We interpret the logical hierarchy tree T as the output of hierarchical clustering and construct a dendrogram, T_{den}, to visualize the hierarchical relationships derived from T (see Figure 2).

Lines 7-12: We *levelize* T_{den} such that all leaf nodes in T_{den} are at the same level (i.e., having the same path distance from the root). The levelization process replicates all leaf nodes that have levels less than $level_{max}$, the largest level of any leaf node of T_{den}. For example, node $x1$ in Figure 2 is replicated once.

Lines 14-24: We evaluate $level_{max} - 1$ clusterings of the netlist, respectively corresponding to the $level_{max}-1$ levels of T_{den}. Evaluation is according to a weighted average Rent exponent criterion [8], defined by

$$R_{c_i} = \frac{\ln(E(c_i)/(Int(c_i) + Ext(c_i)))}{\ln(|c_i|)} + 1; \quad R_{\text{avg}} = \frac{\sum_{c_i \in C}(R_{c_i} \times |c_i|)}{|V|} \tag{1}$$

Here, R_{c_i} is the Rent exponent for cluster c_i [8]; $E(c_i)$ is total *external* hyperedges (i.e., that connect to vertices in other clusters); $Ext(c_i)$ is total pins in c_i that connect to *external* hyperedges; $Int(c_i)$ is total pins that connect to *internal* hyperedges (i.e., that only connect vertices within c_i); and $|c_i|$ is the number of vertices in c_i. A "good" cluster has a lower value of R_{c_i}. We pick the clustering solution with minimum R_{avg} over all $level_{max} - 1$ clustering solutions.

Timing and power. We extract timing information (top $|P|$ timing-critical paths and net slacks) and net switching activity using the *OpenSTA* tool. We use the *findPathEnds* function from Search.hh available at [26], with group count (|P|) = 100000, endpoint count = 1, unique pins = true, and sort by slack = true. Net switching power is obtained from vectorless power analysis with default tool settings. In particular, we use the *findClkedActivity* function from Sta.hh available at [26]. We leverage the timing information in our PPA-aware clustering by calculating (i) timing cost t_p for critical path p and (ii) timing cost t_e of hyperedge e, as in [5]. We consider power and switching activity by defining a *switching cost* of hyperedge e:

$$s_e = (1 + \frac{\theta_e}{\sum_{e \in E} \theta_e})^{\mu} \tag{2}$$

where θ_e is the switching activity of a hyperedge e and μ (default=2) is a scaling factor. The *heavy-edge rating function* of [5] is then extended as:

$$r_{\text{overall}}(u, v) = \sum_{e \in I(v) \cap I(u)} \frac{\langle \alpha, w_e \rangle + \beta t_e + \gamma s_e}{|e| - 1} \tag{3}$$

where u and v denote the pair of cluster candidates while $I(v)$ denotes the incident hyperedges of v. The parameters α, β and γ are scaling factors.

Algorithm 2: Hierarchy-based clustering.

```
   Inputs: Hypergraph H(V, E), Logical hierarchy tree T(V', E')
   Output: Cluster assignments C
1  /* Construct the dendrogram                                    */
2  T_den(V_den, E_den) ← Initialize dendrogram using T
3  leaf_vertices ← Leaf vertices in T_den
4  levels ← Generate levels of each node in T_den
5  level_max ← max(levels)
6  /* Levelizing the dendrogram by replicating leaf nodes        */
7  for each v ∈ V_den do
8      if v is a leaf node and level(v) < level_max then
9          for k ← level(v); k < level_max; k ← k + 1 do
10             v_copy ← Create a copy of node v
11             Assign v_copy to be v's child
12             v ← v_copy
13 /* Find best clustering                                        */
14 C ← Initialize array of cluster assignments for V
15 R_avg_best ← ∞
16 for k ← 0; k < level_max − 1; k ← k + 1 do
17     V_den_k ← All vertices in V_den that have level k
18     C_k ← Clustering solution at level k
19     R_avg_k ← Weighted average of Rent's parameter (Equation 1)
20     if R_avg_k < R_avg_best then
21         R_avg_best ← R_avg_k
22         C ← C_k
23 /* Return best clustering                                      */
24 return C
```

3.2 V-P&R and ML-based *Acceleration*

Cluster shapes (utilization and aspect ratio) significantly impact seed placement and PPA outcomes, as documented in Section 4.4 below. We therefore introduce a *virtualized P&R* (V-P&R) framework to determine the best shape for each cluster. We further apply GNN-based ML modeling to accelerate the V-P&R framework.

Virtualized P&R. The basic idea of V-P&R is that by running place-and-route on the sub-netlist induced by a cluster, we can gain insight into how to model that cluster during seed placement. Figure 3 shows our V-P&R framework. For each given cluster, we first induce the sub-netlist over the instances in the cluster. For each inter-cluster net that is incident to the given cluster, we create input (output) ports in the sub-netlist, corresponding to any sinks (driver) in the cluster. This sub-netlist is passed to the V-P&R framework, along with 20 different combinations of 5 aspect ratios and 4 utilizations. For each combination of aspect ratio and utilization,

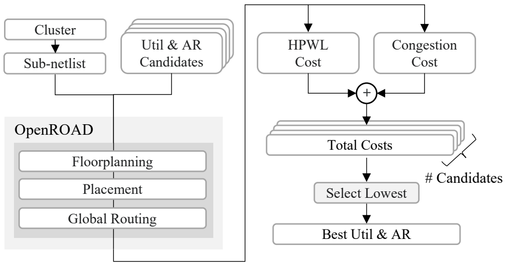

Figure 3: Virtualized P&R flow.

we initialize the floorplan of a "virtual die", then run placement and global routing on the sub-netlist using the default OpenROAD flow script [27]. We then record HPWL and routing congestion.[4] To identify a combination of aspect ratio and utilization that achieves both good HPWL and low congestion, we define *HPWL Cost* as

$$Cost_{HPWL} = \frac{HPWL_{avg}}{Width_{Core} + Height_{Core}} \tag{4}$$

where $HPWL_{avg}$ is the average HPWL of nets in the sub-netlist, and $Width_{Core}$ and $Height_{Core}$ are respectively the width and height of the virtual die. We also define *Congestion Cost* as

$$Cost_{Congestion} = \frac{\sum^{TopX\%\ GCells} Congestion}{TopX\%\ GCells} \tag{5}$$

where X is a hyperparameter (default = 10). Following [13], we define overall *Total Cost* as: $Total\ Cost = Cost_{HPWL} + \delta * Cost_{Congestion}$, where δ is a normalization factor (default = 0.01). The (aspect ratio, utilization) combination that achieves best *Total Cost* is used to create the cluster's .lef model during seed placement.

ML Modeling to Accelerate V-P&R. As described above, V-P&R determines each cluster's ideal shape by running OpenROAD 20 times through the end of global routing (each run can require as much as 3 seconds). This effort grows linearly with design size, and can reach undesirable levels. To address this, we implement an ML-based strategy that in practice accelerates our V-P&R framework by approximately 30×. Specifically, we use a GNN-based model to predict *Total Cost*, replacing execution of OpenROAD in Figure 3.

Our ML model training uses a diversity of clusters generated by perturbing seed and coarsening hyperparameters [29] in our PPA-aware clustering. The training, validation and testing datasets respectively consist of 22700, 5600 and 3200 clusters. Following [9], we sweep aspect ratio in the range [0.75, 1.75] with step size 0.25,[5] and sweep utilization in the range [0.75, 0.90] with step size 0.05. This results in 20 distinct (aspect ratio, utilization) combinations, i.e., 20 candidates for the cluster shape. For each candidate, we run (i) V-P&R, (ii) calculate *Total Cost*, and (iii) use *Total Cost* as a label.

Figure 4 shows our GNN-based model architecture. We convert a cluster's sub-netlist to an undirected graph ("Cluster Graph") using standard clique expansion with edge weight $1/(|e| - 1)$ for each hyperedge e [16]. Each node in the graph has 28 features, extending [15] with two new features italicized below. These features are:

- **Design parameters:** floorplan utilization and aspect ratio.
- **Cluster-level features:** #cells, #nets, #pins, #nets w/ fanout 5-10, #nets w/ fanout > 10, #internal nets, #border nets, total cell area, average cell degree, *average net degree*, average clustering coefficient, density, diameter, radius, edge connectivity, #colors for greedy coloring and average global efficiency.
- **Cell-level features:** Cell area, cell degree, average neighborhood degree, betweenness centrality, closeness centrality, *degree centrality* [21], clustering coefficient, eccentricity and cell type.

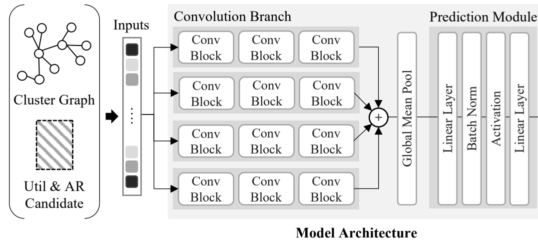

Figure 4: GNN-based architecture to predict *Total Cost*.

Our model takes as input a cluster graph and a candidate shape. Structurally, our model comprises four distinct convolution branches, with each branch containing three convolution blocks (Figure 4). Dimensions of the input, hidden and output layers of the convolution branches are 35, 64 and 32, respectively. The convolution blocks are implemented with hypergraph convolution [3], batch normalization, and skip connections that are specifically used when the input and output dimensions match. As the input data passes through each convolution branch, the results from these branches are accumulated; then, global mean pooling is applied to generate an embedding vector for the cluster. This vector is subsequently passed to a prediction module that predicts the *Total Cost*. Our prediction module is built with two linear layers, batch normalization and an activation function (Figure 4). Here, dimensions of the input, hidden and output layers are 32, 64 and 1, respectively. After predicting the *Total Cost* for all candidates, the candidate with the lowest *Total Cost* is returned as the output of the ML-accelerated V-P&R framework. (Full details and scripts can be seen at [22].)

4 EXPERIMENTAL EVALUATION

Our PPA-aware clustering framework is written in C++ and built on the OpenROAD infrastructure. Our ML model is implemented using *PyTorch Geometric*. We make available all codes and scripts at our GitHub repository [22]. We run all experiments on a server with four 2.4 GHz Intel Xeon(R) Gold 6148 processors and 376 GB RAM. For evaluation we use testcases that are publicly available in the MacroPlacement [28] and OpenROAD [27] GitHub repositories. We use six designs (aes, ariane, BlackParrot, jpeg, MegaBoom and MemPool Group) and the NanGate45 [24] open enablement in our experiments. Table 1 lists the main statistics of these benchmarks. We evaluate our PPA-aware clustering and ML-accelerated V-P&R with OpenROAD and Innovus *v.21.1*.[6] For clarity, we divide our validation efforts into two categories: (i) validation of runtime and PPA with OpenROAD (Section 4.1) and (ii) validation of PPA with Innovus (Section 4.2). To show the benefits of our PPA-aware clustering and ML-accelerated V-P&R methods, we present ablation studies in Sections 4.3 and 4.4. Finally, we explore the tuning of hyperparameters in Section 4.5.

4.1 PPA and Runtime Validation (OpenROAD)

We evaluate our PPA-aware clustering and ML-accelerated V-P&R methods using OpenROAD. We compare our post-place HPWL and

[4]Based on aspect ratio and utilization, a floorplan .lef is created for P&R of the sub-netlist. The IO ports are placed with the OpenROAD pin placer; in the Nangate45 enablement these use the *metal2* (vertical) and *metal3* (horizontal) layers.

[5]More extreme cluster aspect ratios (< 0.75 or > 1.75) generally result in poor PPA.

[6]We do not perform any benchmarking of commercial EDA tools. Further, to avoid inadvertent benchmarking, we mask the target clock period values TCP_{Inv} in Table 1.

Table 1: Specifications of benchmarks.

Design (NG45)	#Insts	#Nets	*TCP_{OR}	*TCP_{Inv}
aes	15547	16338	0.55	-
jpeg	53042	58898	0.80	-
ariane	119256	142226	1.80	-
BlackParrot (BP)	768851	998716	2.30	-
MegaBoom (MB)	1086920	1443755	NA	-
MemPool Group (MP-G)	2729729	3087191	NA	-

*TCP_{OR} (ns) and TCP_{Inv} denote the target clock periods used in the OpenROAD and Innovus flows, respectively. TCP_{Inv} is masked to avoid inadvertent benchmarking of the Innovus tool.

runtime with [9] and the default OpenROAD flow [27] in Table 2. We do not compare with [14] as code was not available from the authors. We first report the cumulative runtimes of clustering and seeded placement (for [9] and our approach) and then normalize these numbers by the placement runtime recorded from the default flow. We observe that our methods achieve up to: (i) 60% improvement in runtime and 5% improvement in HPWL compared to [9] and (ii) 47% improvement in runtime and 1% improvement in post-place HPWL. For MegaBoom and MemPool Group, the clustering runtime of [9] is significantly larger (~2X) than the placement runtime of OpenROAD. We hence omit these with "NA" (in Table 2). We separately give the runtime breakdown of our approach in [22]. We also measure the post-route PPA in Table 3. Here, we exclude MegaBoom and MemPool Group since OpenROAD fails to route for these designs. In Table 3, we exclude [9] in our post-route PPA, since we evaluate our method using a superior community detection algorithm (Leiden) in Section 4.3. We observe that our methods achieve maximum (average) percentage PPA improvements of 5 (2), 63 (26), 90 (29) and 0.7 (0.2) in *rWL*, *WNS*, *TNS* and *Power*, respectively, when compared to the default flow. We observe marginal improvement in power compared to the default flow. However, when compared to clustering methods with no PPA-awareness, our method achieves up to 5% improvement in power (Section 4.3).

Table 2: Evaluation of post-place results with OpenROAD.

Design	[9]		Ours	
	HPWL	CPU	HPWL	CPU
aes	1.007	1.567	**1.000**	**0.802**
jpeg	**0.973**	1.000	1.011	**0.524**
ariane	1.046	0.931	**0.989**	**0.604**
BlackParrot	0.996	1.824	**0.989**	**0.738**
MegaBoom	NA	NA	**0.997**	**0.664**
MemPool Group	NA	NA	1.001	**0.533**

Table 3: Evaluation of post-route results with OpenROAD.

Design	Flow	Post-route PPA			
		rWL	WNS	TNS	Power
aes	*Default*	1.00	-220	-32.08	0.296
	Ours	**0.98**	**-210**	**-31.85**	**0.294**
jpeg	*Default*	1.00	**-410**	**-130.45**	0.441
	Ours	**0.95**	-470	-177.73	**0.438**
ariane	*Default*	1.00	-200	-139.21	**0.655**
	Ours	**0.99**	**-100**	**-14.21**	0.661
BP	*Default*	1.00	-410	-1441.24	4.93
	Ours	**0.99**	**-150**	**-563.69**	**4.91**

The unit of WNS is ps, the unit of TNS is ns, and the unit of Power is W.

4.2 PPA Validation (Cadence Innovus)

In this section, we validate our PPA-aware clustering and ML-accelerated V-P&R methods with Cadence Innovus *v.21.1*. Table 4 compares post-route PPA metrics to those obtained with the standard Innovus flow [28]. For all designs, our methods significantly improve most PPA metrics. We achieve maximum (average) percentage improvements of 1.9 (0.2), 98 (35), 99 (49) and 4 (1) in *rWL*, *WNS*, *TNS* and *Power*, respectively. We observe similar runtime compared to the standard Innovus flow.

Table 4: Evaluation of post-route results with Innovus.

Design	Flow	Post-route PPA			
		rWL	WNS	TNS	Power
aes	*Default*	**1.000**	-72	-7.94	**0.050**
	Ours	1.014	**-60**	**-7.37**	**0.050**
jpeg	*Default*	**1.000**	-41	-1.17	0.284
	Ours	1.005	**-6**	**-0.06**	**0.273**
ariane	*Default*	1.000	-97	-52.73	**0.841**
	Ours	**0.998**	**-80**	**-39.86**	0.842
BP	*Default*	1.000	**-134**	-548.74	4.492
	Ours	**0.999**	-165	**-372.80**	**4.481**
MB	*Default*	1.000	-28	-1.34	1.611
	Ours	**0.981**	**-5**	**-0.014**	**1.586**
MP-G	*Default*	1.000	-37	-0.96	**2.671**
	Ours	**0.994**	**-29**	**-0.63**	2.679

The unit of WNS is ps, the unit of TNS is ns, and the unit of Power is W.

4.3 Comparison of PPA-awareness

We now assess the PPA-relevance of our clustering method, relative to Leiden clustering and TritonPart's default clustering method (multilevel FC, denoted as MFC in Table 5). We use post-route PPA metrics obtained from using Leiden and multilevel FC in our overall flow. Table 5 presents evaluation with OpenROAD (more results are available in [22]). Routed wirelength is normalized to the value obtained with the default OpenROAD flow. We observe that compared to Leiden, our clustering approach leads to better PPA outcomes – up to 5% improvement in *rWL*, *WNS* and *TNS* and up to 2% improvement in *Power*. Compared to the multilevel FC, our clustering achieves better percentage PPA improvements (up to 6, 13, 10 and 2, respectively) on the same metrics. These results indicate that consideration of additional netlist information (logical hierarchy, timing path slacks, and switching activity of nets) during clustering helps to improve the final PPA. Thus, the Table 5 data confirm PPA-relevance of our proposed clustering methodology.

Table 5: Evaluation of our PPA-aware clustering framework.

Design	Method	Post-route PPA			
		rWL	WNS	TNS	Power
aes	*Leiden*	0.991	-211	-33.41	0.295
	MFC	1.028	-214	-42.42	0.311
	Ours	**0.980**	**-210**	**-31.85**	**0.294**
jpeg	*Leiden*	0.998	-472	-177.82	0.446
	MFC	1.011	-542	-196.42	0.459
	Ours	**0.950**	**-470**	**-177.73**	**0.438**
ariane	*Leiden*	0.996	-105	-14.29	0.672
	MFC	1.001	-134	-16.28	0.669
	Ours	**0.972**	**-100**	**-14.21**	**0.661**

The unit of WNS is ps, the unit of TNS is ns, and the unit of Power is W.

4.4 V-P&R Model Evaluation

We evaluate the performance of our GNN-based model using two metrics: (i) mean absolute error (*MAE*), which evaluates the average magnitude of absolute prediction errors, aid (ii) *R2 score*, which quantifies the amount of variance in the predicted values. The values of the (*Total Cost*) labels lie in the range [0.564, 2.96] and have a mean of 1.703 with standard deviation 0.727. Our results show that we achieve an *MAE* of 0.105, 0.113, and 0.131 for the training, validation, and test datasets, respectively. Our *R2 score* is 0.788, 0.753, and 0.638 for the three datasets. These metrics confirm the model prediction accuracy of our GNN-based architecture.

Table 6 presents PPA benefits of our ML-accelerated V-P&R framework, using Innovus, again with more results available in [22]. In this study, we first substitute the ML-accelerated V-P&R framework (denoted as $V\text{-}P\&R_{ML}$ in the table) with (i) random cluster shape assignments (*Random*) or with (ii) fixed cluster shape assignments where each cluster is assigned utilization = 0.9 and

Table 6: Evaluation of our ML-based V-P&R framework.

Design	Shape	Post-route PPA			
		rWL	WNS	TNS	Power
ariane	*Random*	0.992	-94	-59.80	**0.840**
	Uniform	1.000	-103	-66.12	**0.840**
	V-P&R$_{ML}$	**0.977**	**-80**	**-39.86**	0.842
jpeg	*Random*	1.010	-11	-0.29	0.288
	Uniform	1.000	-19	-0.33	0.279
	V-P&R$_{ML}$	**0.996**	**-6**	**-0.06**	**0.273**
MB	*Random*	0.999	-17	-0.71	1.596
	Uniform	1.000	-14	-0.40	1.601
	V-P&R$_{ML}$	**0.961**	**-5**	**-0.014**	**1.586**

The unit of WNS is ps, the unit of TNS is ns, and the unit of Power is W.

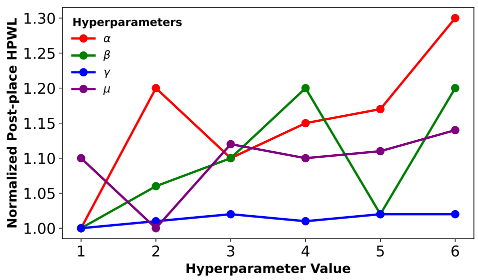

Figure 5: Hyperparameter validation.

aspect ratio = 1.0 (*Uniform*). Then, we run the clustering-driven placement and collect the post-route PPA metrics. In Table 6, all *rWL* values for each design are normalized to the value obtained with uniform cluster shape assignments. The results show that compared to (random, uniform) assignments, ML-accelerated V-P&R achieves (arithmetic) average percentage improvements of $(2, 2)$, $(44, 52)$, $(85, 73)$ and $(2, 1)$ in *rWL*, *WNS*, *TNS* and *Power*, respectively,

4.5 Hyperparameter Selection

We have determined default values for the hyperparameters (α, β, γ and μ) by performing a study involving three designs: *aes*, *jpeg*, and *ariane*. We define the score value as the arithmetic mean of the post-place HPWL improvement.[7] In our experiments, for each design, we: (i) vary each parameter while keeping the other three constant[8]; (ii) run our flow with OpenROAD and record the post-place HPWL; and (iii) normalize the HPWL to the value obtained with our default hyperparameter settings (see Figure 5). From our findings, we observe that our default setting of the hyperparameters is a reasonable choice. A more detailed exploration of the hyperparameter space – with more designs and impact on post-route PPA– is provided in our GitHub repository [22].

5 CONCLUSION

We have developed new *PPA-aware clustering* and ML-accelerated *virtualized P&R* methods that improve *seed placements* for large-scale global placement. For PPA-aware clustering, we adapt the multilevel clustering framework of [29] to consider logical hierarchy, timing paths and switching activities. Our ML-accelerated V-P&R framework efficiently predicts beneficial aspect ratios and utilizations to apply with clusters produced by the PPA-aware clustering. As noted above, our ML-model accelerates the V-P&R framework by 30× with a one-time training cost. Together, these elements enable generation of a high-quality *seed placement* that leads to final placements with improved post-route PPA metrics. Experimental results confirm both PPA *and* runtime benefits of our methods. When integrated with the open-source OpenROAD tool, our methods can improve *both* PPA and runtime, with 29% average TNS improvement and 36% runtime speedup. With the commercial Cadence Innovus tool, we achieve better PPA, with 49% average TNS improvement. Our ongoing research pursues confirmation of the benefits from our methods on additional testcases, design enablements and P&R tools. We are also studying the effects of different cluster shapes (L-shaped, diamond, circle, etc.) on placement, and enhancing power-awareness of our clustering methodology to further improve the post-route power metric. Last, we plan to study the benefits of our *PPA-aware* clustering and ML-accelerated V-P&R framework in the context of 3D placement.

[7] We consider HPWL for faster TAT and more extensive parameter space exploration.

[8] The x-axis in Figure 5 denotes multipliers applied to the default values of the hyperparameters. Our study assesses whether there are benefits to larger relative weighting of the terms in Equation 3. Hence we sweep our multipliers of default values in the range $[1, 6]$ with step size 1.

ACKNOWLEDGMENTS

We thank the authors of [9] for sharing scripts and codes to run their flows. We also thank the authors of [15] for sharing their feature extraction codes and ML model. This work is supported in part by DARPA HR0011-18-2-0032 (OpenROAD), the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT) (No. 2021-0-00754, Software Systems for AI Semiconductor Design) and by the Samsung Advanced Institute of Technology.

REFERENCES

[1] C. J. Alpert, A. B. Kahng, G.-J. Nam, S. Reda and P. Villarrubia, "A Semi-Persistent Clustering Technique for VLSI Circuit Placement", *Proc. ISPD*, 2005, pp. 200–207.
[2] C. Alpert, A. Kahng, G.-J. Nam, S. Reda and P. Villarrubia, "A Fast Hierarchical Quadratic Placement Algorithm", *IEEE Trans. CAD* 25(4) (2006), pp. 678–691.
[3] S. Bai, F. Zhang and P. H. S. Torr, "Hypergraph Convolution And Hypergraph Attention", *Pattern Recognition*, 110 (2021), 107637.
[4] V. D. Blondel, J. L. Guillaume, R. Lambiotte and E. Lefebvre, "Fast Unfolding of Communities in Large Networks", *J. Stat. Mech. Theor. Exp.*, 10 (2008), pp. 1-12.
[5] I. Bustany, G. Gasparyan, A. B. Kahng, Y. Koutis, B. Pramanik and Z. Wang, "An Open-Source Constraints-Driven General Partitioning Multi-Tool for VLSI Physical Design", *Proc. ICCAD*, 2023, pp. 1-9.
[6] I. Bustany, A. B. Kahng, Y. Koutis, B. Pramanik and Z. Wang, "SpecPart: A Supervised Spectral Framework for Hypergraph Partitioning Solution Improvement", *Proc. ICCAD*, 2022, pp. 1-9.
[7] C.-K. Cheng, A. B. Kahng, I. Kang and L. Wang, "RePlAce: Advancing Solution Quality and Routability Validation in Global Placement", *IEEE Trans. CAD* 38(9) (2019), pp. 1717-1730.
[8] Y. Cheon and D. F. Wong, "Design Hierarchy Guided Multilevel Circuit Partitioning", *Proc. ISPD*, 2002, pp. 30-35.
[9] M. Fogaça, A. B. Kahng, E. Monteiro, R. Reis, L. Wang and M. Woo, "On the Superiority of Modularity-Based Clustering for Determining Placement-Relevant Clusters", *Integration: The VLSI Journal* 74 (2020), pp. 32-44.
[10] L. C. Freeman, "Centrality in Social Networks Conceptual Clarification", *Social Networks* 1(3) (1978) pp. 215-239.
[11] B. Hu and M. Marek-Sadowska, "Fine Granularity Clustering-Based Placement", *IEEE Trans. CAD* 23(4) (2004), pp. 527-536.
[12] G. Karypis and V. Kumar, "Multilevel K-Way Hypergraph Partitioning", *Proc. DAC*, 1999, pp. 343-348.
[13] M.-C. Kim, N. Viswanathan, C. J. Alpert, I. L. Markov and S. Ramji, "MAPLE: Multilevel Adaptive Placement For Mixed-size Designs", *Proc. ISPD*, 2012.
[14] Y.-C. Lu, T. Yang, S. K. Lim and H. Ren, "Placement Optimization Via PPA-Directed Graph Clustering", *Proc. MLCAD*, 2022, pp. 1-6.
[15] K. Min, S. Kwon, S.-Y. Lee, D. Kim, S. Park and S. Kang, "ClusterNet: Routing Congestion Prediction and Optimization Using Netlist Clustering and Graph Neural Networks", *Proc. ICCAD*, 2023, pp. 1-9.
[16] T. Lengauer, *Combinatorial Algorithms For Integrated Circuit Layout*, Wiley Teubner, 1990.
[17] J. A. Roy, S. N. Adya, D. A. Papa and I. L. Markov, "Min-cut Floorplacement", *IEEE Trans. CAD* 25(7) (2006), pp. 1313—1326.
[18] R. S. Shelar, "An Efficient Clustering Algorithm for Low Power Clock Tree Synthesis", *Proc. ISPD*, 2007, pp. 181–188.
[19] V. A. Traag, L. Waltman and N. J. van Eck, "From Louvain To Leiden: Guaranteeing Well-Connected Communities", *Scientific Reports* 9 (2019).
[20] N. Viswanathan, M. Pan and C. Chu, "FastPlace 3.0: A Fast Multilevel Quadratic Placement Algorithm With Placement Congestion Control", *Proc. ASPDAC*, 2007, pp. 135-140.
[21] J. Zhang and Y. Luo, "Degree Centrality, Betweenness Centrality, and Closeness Centrality in Social Network", *Intl. Conf. on MSAM*, 2017, pp. 300–303.
[22] ABKGroup/BlobPlacement GitHub Repository. https://github.com/ABKGroup/BlobPlacement
[23] Cadence Innovus Implementation System, version 21.1. www.cadence.com
[24] NanGate45 PDK. https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts/tree/master/flow/platforms/nangate45
[25] The OpenROAD Project (GitHub). https://theopenroadproject.org and https://github.com/The-OpenROAD-Project/OpenROAD
[26] OpenSTA Static Timer. https://github.com/The-OpenROAD-Project/OpenSTA
[27] OpenROAD-Flow-Scripts. https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts
[28] TILOS MacroPlacement repository. https://github.com/TILOS-AI-Institute/MacroPlacement
[29] TritonPart: An Open-Source Constraints-Driven Partitioner. https://github.com/The-OpenROAD-Project/OpenROAD/tree/master/src/par