RegPlace: Regularity-Aware Placement for Full-System DNN Accelerator Designs

Jiaxi Jiang, Yuan Pu, Mingjun Li, Yuxuan Zhao, Peiyu Liao, Zuodong Zhang, Yibo Lin, *Member, IEEE*, and Bei Yu, *Senior Member, IEEE*

***Abstract*—The rise of deep neural network (DNN) accelerators demands physical design tools that recognize spatial regularity patterns. Traditional placers, unaware of the regularity of spatial arrays, produce suboptimal solutions. This work proposes RegPlace, a regularity-aware placement algorithm for full-system DNN accelerators that automatically identifies processing element using graph convolutional networks and employs a variance-based soft regularity loss to guide optimization. Compared to state-of-the-art methods, our approach achieves up to 6% wirelength reduction while maintaining comparable runtime, with post-placement metrics further confirming its effectiveness.**

***Index Terms*—Physical design, partitioning, placement, systolic arrays.**

I. Introduction

With the boom of deep neural networks (DNNs), highly parameterized hardware accelerators [1], [2] and complete system-on-chip (SoC) integrations [3] have emerged. A crucial feature of these DNN accelerators is their spatial regularity, derived from the commonly used spatial arrays. Ignoring this regularity during physical design leads to inferior wirelength issues and routing congestion.

DNN accelerators differ from conventional CPU designs through spatial regularity in their dataflow, primarily from spatial arrays where parallel execution units are laid out in structured patterns. These arrays distribute computation across processing elements (PEs), each performing localized operations, allowing high parallelism and data reuse. Modern full-system DNN hardware integrates these specialized accelerators with traditional processors within a unified SoC, with Gemmini [3] being a notable example that combines RISC-V CPU cores with spatial-array-based accelerators.

Neglecting spatial regularity during placement causes performance degradation in two ways: first, large macros can disrupt the spatial array structure as cells in the array become dispersed [4]; second, the array's shape becomes irregular due to compression from other components or curved forces in nonlinear placement [5].

The benefits of regularity-aware placement have been recognized for decades [6]. Since then, extensive research has explored effective techniques for regularity-driven placement. The fundamental paradigm shared between prior regularity-aware algorithms is the presumption that components with regularity require specific placement constraints [5], [6], [7], [8], [9], [10].

Received 8 October 2024; revised 18 February 2025 and 22 May 2025; accepted 13 June 2025. Date of publication 3 July 2025; date of current version 22 January 2026. This work was supported in part by The Research Grants Council of Hong Kong SAR under Grant CUHK14210723 and Grant CUHK14211824. This article was recommended by Associate Editor V. Pavlidis. *(Corresponding author: Bei Yu.)*

Jiaxi Jiang, Yuan Pu, Mingjun Li, Yuxuan Zhao, Peiyu Liao, and Bei Yu are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, SAR (e-mail: byu@cse.cuhk.edu.hk).

Zuodong Zhang and Yibo Lin are with the School of Integrated Circuits and the Beijing Advanced Innovation Center for Integrated Circuits, Peking University, Beijing 100871, China, and also with the Institute of Electronic Design Automation, Peking University, Wuxi 221000, China.

Digital Object Identifier 10.1109/TCAD.2025.3585796

The initial phase of regularity-aware placement involves identifying and extracting circuits that exhibit regularity. For bit-slice datapaths, automated extraction techniques [11], network flow-based methods [12], and data learning approaches [13] have been developed. After extraction, these approaches apply different constraint mechanisms to enforce regularity in placement. These regularity enforcement techniques generally fall into two categories: 1) soft constraints and 2) hard constraints. Soft constraint approaches, such as those in [8], utilize pseudo nets to connect datapath-grouped cells, using variance along the dataflow direction as the objective. Hard constraint methods, as in [9], cluster datapath components into large macros, forcing regular placement.

The regularity in DNN accelerators differs from traditional bit-slice datapaths. In DNN accelerators, each PE connects in a highly structured 2-D array pattern rather than the 1-D patterns typical in bit-slice designs. Current research approaches include hierarchical macro-based placement with predefined arrays and flat placement integrating PE cells with other logic [5], [10]. While macro-based approaches offer faster runtime, flat placement potentially yields better results by addressing PE-to-logic wirelength minimization [5]. For 2-D spatial arrays, the approach in [5] first determines cell orientation using IO pin locations and dataflow direction, then implements array regularization during optimization to maintain regularity. DG-RePlace [10] achieves efficient global placement for machine learning accelerators by employing physical hierarchy extraction, dataflow-driven initial placement, and constructing specialized datapath constraints for 2-D PE arrays to ensure regularity. Their methods reduced wirelength by 7% and 6% compared to the best nonregularity-aware baseline. Such improvements are particularly noteworthy in the placement research domain, where the field has matured considerably and even marginal gains often represent substantial technical advances.

Existing studies on regularity-aware placement focus on the accelerator in isolation. In full-system designs, where spatial arrays occupy smaller portions, excessive emphasis on regularity could be counterproductive, yet ignoring it leads to some problems. To balance these concerns, we implement a variance-based soft regularity constraint for 2-D spatial arrays. This approach guides the optimizer's search direction. By treating PEs and arrays as soft macros instead of isolated standard cells, we prevent disruption of array structures while allowing flexibility for overall placement optimization.

In this work, we explore a new research direction for full-system DNN accelerator designs: using spatial array regularity to guide whole-system placement. We propose RegPlace, a regularity-aware placement algorithm with soft constraints, as illustrated in Fig. 1. Our contributions are as follows.

1) We are the first to explore 2-D spatial array regularity for placement in full-system DNN accelerator designs.
2) We propose a method using netlist hierarchy information and graph convolutional networks to automatically extract PEs.
3) We introduce a variance-based soft regularity constraint to guide placement while preserving 2-D spatial array structures.

1937-4151 © 2025 IEEE. All rights reserved, including rights for text and data mining, and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on February 03,2026 at 02:27:39 UTC from IEEE Xplore. Restrictions apply.

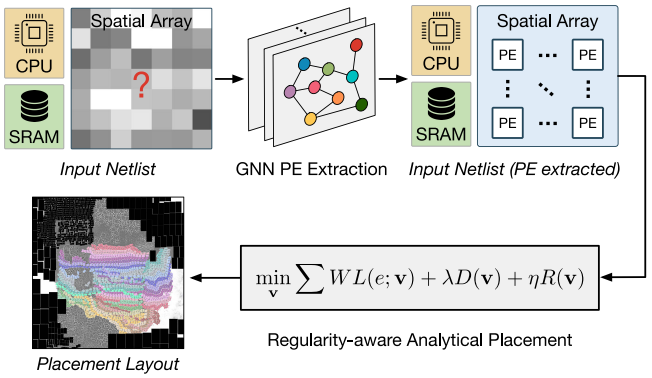


Fig. 1. Overview of RegPlace workflow.

4) Compared to state-of-the-art placers, we achieve a 6.0% reduction in wirelength with comparable runtime, with post-placement metrics confirming our approach's effectiveness.

II. Algorithms

A. Problem Formulation

Modern placers use analytical methods to minimize the half-perimeter wirelength (HPWL) while meeting density targets. State-of-the-art solutions [14], [15] employ electrostatics-based models, treating cells as charges that reach equilibrium when properly distributed.

The optimization objective can be formulated as

$$\min_{\boldsymbol{v}} \sum_{e \in E} WL(e;\boldsymbol{v}) + \lambda D(\boldsymbol{v}) \tag{1}$$

where E is the set of netlists, e denotes each net, and $\boldsymbol{v}$ represents the coordinates of cells or macros. WL denotes the differentiable wirelength approximation, and D is the density penalty to spread cells across the layout, with weight λ.

Given the netlist of a full-system DNN accelerator design, our goal is to find the legal positions of the movable standard cells and macros to minimize the wirelength while preserving the regularity inherent in spatial arrays. We augment the standard electrostatics-based placement with regularity-aware constraints to achieve this goal.

B. RegPlace Approach

Our overall algorithm flow is shown in Fig. 2. Starting with the netlist of a DNN accelerator design, we extract the standard cells of PEs using a graph convolutional network, then identify the regularity of spatial arrays. A prototype placement is created by a wirelength and density-driven placer, followed by incorporating a novel regularity penalty formulation as a soft constraint. This regularity-aware placement process iterates until it fails to improve the wirelength-density objective. Finally, legalization adjusts the placement to eliminate overlaps.

C. Spatial Array Extraction

DNN hardware designers can provide high-level descriptions to extract PE and spatial array information. However, in cases like Gemmini [3] with many parametric variations, they lack clear mapping for spatial arrays between RTL files and physical design files, necessitating additional annotation. We propose a GCN-based approach to automatically identify PEs and extract spatial array patterns from dataflow when designers do not provide this information.

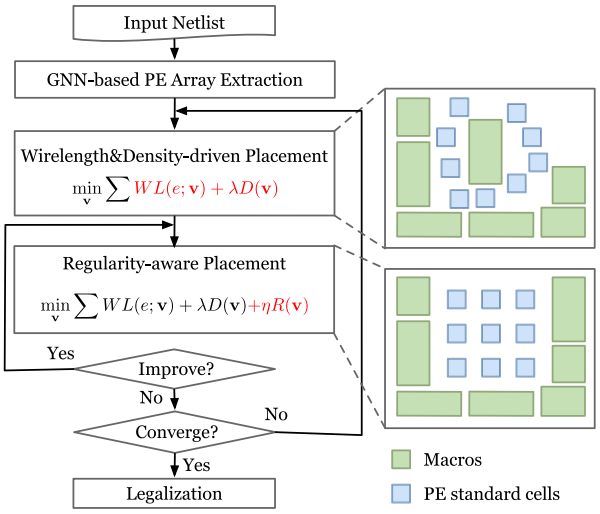


Fig. 2. Algorithm flow of RegPlace.

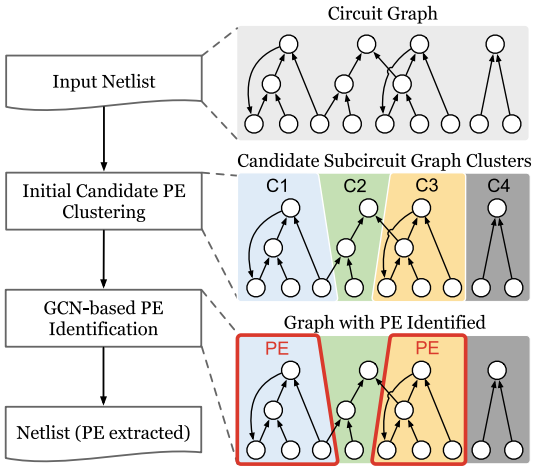


Fig. 3. GCN-based PE extraction flow.

The overview of our spatial array extraction method is depicted in Fig. 3. We first transform the input netlist from a hypergraph to a directed graph, then divide the graph into candidate PE graphs based on hierarchy information, and finally use GCN to perform graph classification.

We traverse the netlist hierarchy from bottom to top to identify PE candidates by comparing module size against predefined PE size thresholds. Specifically, we define PE candidates as modules containing between 200 and 20 000 standard cells, a range empirically determined based on typical PE implementations in modern DNN accelerators. The hierarchical traversal terminates when a module's size exceeds the upper threshold (indicating it is likely a larger functional block).

In the netlist-to-graph transformation, we first identify the source pin of each hypernet. For each remaining sink pin, we create an edge from the source pin to the sink pin, preserving the input/output information. The initial node feature is the one-hot code of the standard cell type.

After PE identification, we extract the PE connection relationships by analyzing the netlist's dataflow patterns. The spatial array identification process first finds PEs at corners and boundaries, then establishes their sequence, and finally assigns indices to each PE in the array.

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on February 03,2026 at 02:27:39 UTC from IEEE Xplore. Restrictions apply.

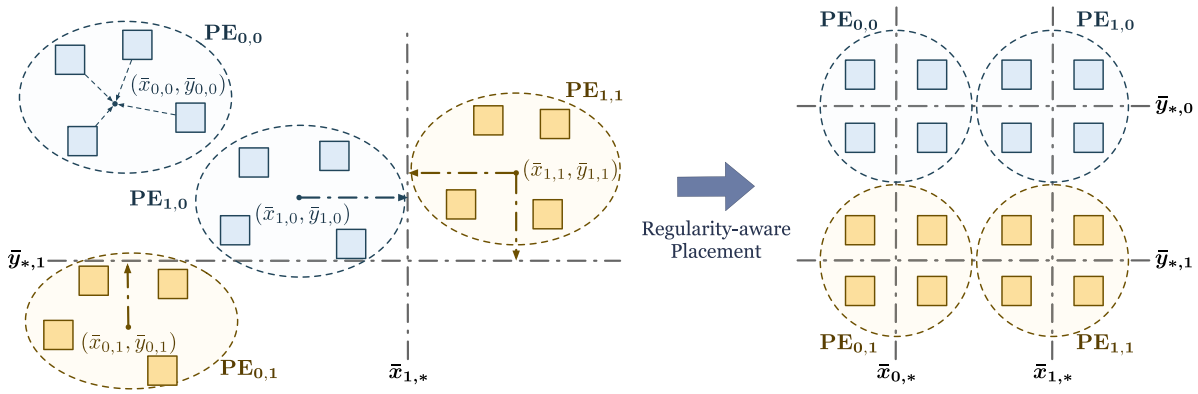


Fig. 4. Soft constraint for PE and spatial array.

D. Soft Constraints for PE and Spatial Array

We assume that standard cells belonging to the same PE module should be placed close together, and PEs in the same row or column should be placed roughly in a straight line. Unlike previous approaches using hard regularity constraints, we integrate soft regularity constraints during global placement, providing better solution quality and flexibility.

Overview of Soft Constraint: We implement a variance-based soft constraint, treating circuits with such constraints as soft macros whose shapes can vary according to standard cells' position distribution. By considering each PE and the entire spatial array as soft macros, the regularity is preserved while allowing adjustment according to the rest of the circuit design. Fig. 4 visualizes position changes after applying the soft constraint.

Soft Constraint for PE: To ensure that standard cells within a PE are placed close together, we use variance in x and y directions as the PE regularity penalty. For a PE in the ith column and jth row, the x-variance is calculated as

$$\sigma^2_{x_{i,j}} = \frac{1}{n_{i,j}} \sum_{k=1}^{n_{i,j}} \left(x^k_{i,j} - \bar{x}_{i,j} \right)^2 \tag{2}$$

where $\bar{x}_{i,j} = (1/n_{i,j}) \sum_{k=1}^{n_{i,j}} x^k_{i,j}$ and $n_{i,j}$ is the number of standard cells in that PE. The PE regularity term is

$$R_{pe} = \sum \sigma^2_{x_{i,j}} + \sigma^2_{y_{i,j}}. \tag{3}$$

Soft Constraint for Spatial Array: To prevent spatial array deformation, we apply variance penalties to each row and column of the spatial array. The variance penalty is calculated based on centers of PEs. This preserves the array's structure while accommodating control logic integration, unlike pseudo nets which can create overly dense PE placements. During placement, we dynamically adjust coefficients for x and y directions based on their respective variance magnitudes

$$\begin{aligned} R_a = \sum & w_{x_{i,*}} \sigma^2_{x_{i,*}} + w_{y_{i,*}} \sigma^2_{y_{i,*}} \\ & + w_{x_{*,j}} \sigma^2_{x_{*,j}} + w_{y_{*,j}} \sigma^2_{y_{*,j}} \end{aligned} \tag{4}$$

where $\sigma^2_{x_{i,*}}$ is the variance of the PE centers in the x-direction for the ith column. The weights (e.g., $w_{x_{i,*}}$) are adjusted based on their respective variance magnitudes. For instance, if the x-variance is larger, the x-weight is significantly smaller than the y-weight. Because if the x-weights and y-weights are set to be equal, it will only lead to overly dense PE placements.

Integration With Wirelength and Density: The regularity-based soft constraints guide the optimizer's search direction rather than being optimization goals themselves. We progressively increase the coefficient of regularity terms while keeping the density weight constant until the wirelength-density objective ceases to improve. The final objective function of RegPlace is

$$\min_{\boldsymbol{v}} \sum_{e \in E} WL(e; \boldsymbol{v}) + \lambda D(\boldsymbol{v}) + \eta_{pe} R_{pe}(\boldsymbol{v}) + \eta_a R_a(\boldsymbol{v}). \tag{5}$$

The regularity weights η_{pe} and η_a are adjusted dynamically based on original objective's response. When adding regularity terms improves the original objective, we increase the weights; conversely, when it hinders progress, we decrease them. We use a step size interval of $[1.01, 1.05]$ with an initial value of 0.001 as the configuration for both regularity weights, and each fixed density weight undergoes 6–50 iterations of regularity-aware placement.

E. Preconditioning

To ensure efficient convergence, we apply a Jacobi preconditioner to the gradients of our regularity terms, consistent with the approach used in DREAMPlace [15]. The preconditioners for PE regularity and array regularity are derived from their second derivatives, resulting in an overall preconditioner that balances wirelength, density, and regularity components.

III. Experimental Results

Our algorithm is implemented in C++/CUDA, built upon DREAMPlace [15], tested on a Linux machine with an Intel Xeon Platinum 8368 CPU and NVIDIA A800 GPU. We preprocess netlists to generate PE candidate graphs, which are subsequently analyzed via GCN for PE type identification.

Baseline Setup: We compare RegPlace against DREAMPlace 4.1.0 [16], array regularization method [5], DG-RePlAce [10]. For DREAMPlace, we disable Abacus legalization due to PDK numerical issues, retaining only greedy legalization. We also created a long-running version of DREAMPlace by changing "sub_iteration" from 1 to 2 for further comparison with RegPlace.

For array regularization [5], multiple regularizations or improper scheduling leads to solution divergence in our macro-heavy testcases, so we apply it only once at iteration 300. Our implementation calculates the bounding box of PE centers, matches diagonal corners to determine orientation, and equally divides the box to position PEs. For DG-RePlAce [10], we disable its dataflow-driven initial distribution due to clustering compatibility issues.

Benchmark Generation: Our testcases use Rocket Chip Coprocessor configurations from Gemmini [3] with spatial arrays of various sizes, datatypes, and dataflow types. In Table I, "B-f," "S-f," "H-f," and "8-int" represent brain-half precision float, single precision

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on February 03,2026 at 02:27:39 UTC from IEEE Xplore. Restrictions apply.

TABLE I
STATISTICS OF OUR BENCHMARK

Testcase	#Cells	#Nets	Datatypes	Dataflows	#PE Cells	#Cells / PE	Array Size
bf16	751504	765044	B-f	BOTH	191888	11993	(4,4)
bf16p	743786	757329	B-f	BOTH	195072	12192	(4,4)
bf168	1441217	1454757	B-f	BOTH	774976	12109	(8,8)
dual	1371681	1379352	B-f / 8-int	WS/WS	251136	2736 / 297	(8,8) / (16,16)
fp	818735	832284	S-f	BOTH	237120	14820	(4,4)
fp16	763113	776661	H-f	BOTH	201024	12564	(4,4)
gemmini	975826	989362	8-int	BOTH	279404	1090	(16,16)
infer	965642	979179	8-int	BOTH	281088	1098	(16,16)
inferb	1928346	1949767	8-int	BOTH	283904	1109	(16,16)
inferp	2108980	2130657	8-int	BOTH	1146880	1120	(32,32)
lean	761413	774953	8-int	WS	75776	296	(16,16)
train	1692279	1713972	S-f	BOTH	957376	14959	(8,8)

TABLE II
RESULT OF PE IDENTIFICATION

PE Type	#Graph	Recall	Precision
Not PE	561	0.979	0.965
int (WS)	154	1.000	1.000
int (BOTH)	538	1.000	1.000
float (WS)	20	1.000	0.909
float (BOTH)	58	0.655	0.792
Average	-	0.927	0.933

float, half precision float, and 8-bit integer respectively. "BOTH" indicates support for both weight and output stationary dataflows, while "WS" indicates weight stationary only.

GCN Identification Result: We consolidated our six PE types into five categories for GCN training: 1) Not PE; 2) Integer PE (WS); 3) Integer PE (BOTH); 4) Floating Point PE (WS); and 5) Floating Point PE (BOTH). Using the DGL framework with Adam optimizer (1e-4 learning rate, 200 epochs, batch size 50), we achieved high accuracy as shown in Table II. The PE identification process consumes negligible computational resources compared to placement runtime as shown in Fig. 5.

Overall Comparisons: Table III shows our RegPlace achieves a 9.9% HPWL reduction compared to default DREAMPlace, and 6.0% reduction compared to DREAMPlace 4.1.0 (long) with comparable iterations and runtime. The increased iterations are due to our regularity-aware approach with fixed density weight, which slows cell spreading but improves quality. With numerous macros and without I/O pin-based orientation constraints, array regularization [5] reduces HPWL in only limited test cases. Compared to DG-RePlAce, our approach achieves 5.2% HPWL improvement while maintaining similar runtime. While DG-RePlAce addresses PE adjacency, our method enables comprehensive global optimization. Post-placement metrics Table IV (generated by Innovus) show RegPlace achieves 5.3% reduction in routed wirelength compared to DREAMPlace 4.1.0 (long), with moderate congestion improvements.

Analysis and Visualization: RegPlace's runtime breakdown (Fig. 5) shows regularity extraction adds minimal overhead. Our method performs better for testcases with small PE sizes (gemmini, infer, lean) or small array dimensions (bf16, bf16p, fp, fp16), where regularity patterns are more sensitive to disruption from macros.

Visual comparisons (Figs. 6 and 7) demonstrate RegPlace's effectiveness. In lean, DREAMPlace allows macros to mix with and damage the spatial array's regularity, while RegPlace maintains a clear regular structure despite curved spreading forces. In dual, DREAMPlace's result shows a squeezed, truncated upper-right array and irregular lower-left array with rows crossing. RegPlace preserves clear regularity in both arrays, with rows in straight lines despite macro influence.

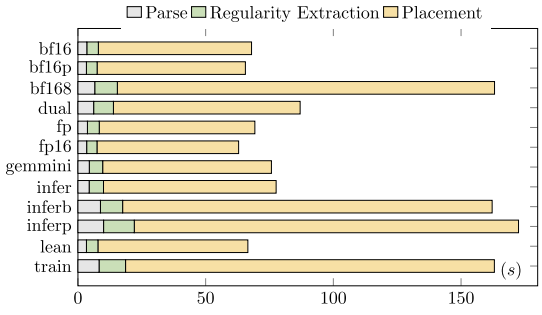


Fig. 5. Runtime breakdown of RegPlace.

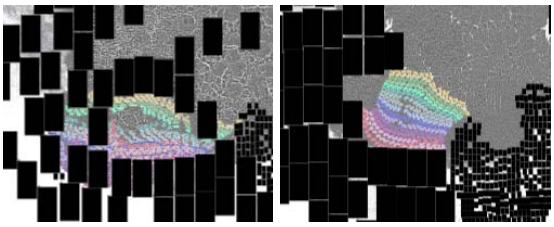


Fig. 6. Placement results of (a) DREAMPlace and (b) RegPlace (testcase "lean").

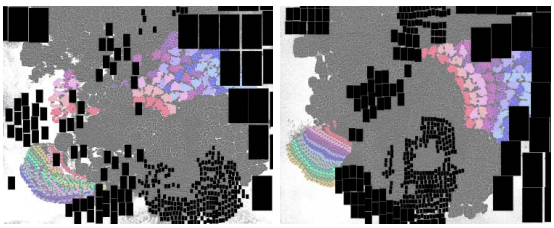


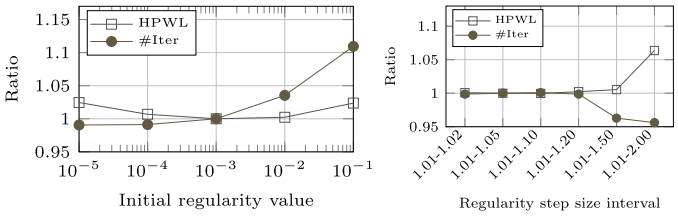
Fig. 7. Placement results of (a) DREAMPlace and (b) RegPlace (testcase "dual").

Regularity parameter analysis (Fig. 8) shows initial regularity values have a nonmonotonic effect on HPWL, initially improving then degrading performance at extreme values. This validates our viewpoint in the introduction that both ignoring regularity and overemphasizing regularity will lead to placement quality degradation. Iteration count increases with regularity values as PE cell constraints impede cell spreading. Regularity step size intervals show small intervals minimally impact HPWL and iterations, while large intervals accelerate weight growth but degrade HPWL despite fewer iterations.

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on February 03,2026 at 02:27:39 UTC from IEEE Xplore. Restrictions apply.

TABLE III
COMPARISON WITH DREAMPLACE 4.1.0, DREAMPLACE 4.1.0 (LONG), ARRAY REGULARIZATION (ONCE) AND DG-REPLACE

Testcase	DREAMPlace 4.1.0 [16]			DREAMPlace 4.1.0 [16](long)			Array Regularization [5]			DG-RePlAce [10]			RegPlace		
	#Iter	HPWL	time (s)	#Iter	HPWL	time (s)	#Iter	HPWL	time (s)	#Iter	HPWL	time (s)	#Iter	HPWL	time (s)
bf16	810	8.03E7	45.768	1567	7.54E7	55.504	754	8.08E7	49.871	1150	7.32E7	58.852	1543	7.04E7	63.461
bf16p	817	7.90E7	46.820	1573	7.98E7	56.396	744	7.84E7	49.023	1150	7.28E7	58.534	1511	7.02E7	61.378
bf168	924	1.12E8	84.211	1791	1.04E8	107.601	847	1.42E8	128.805	1320	1.07E8	147.557	1718	1.02E8	154.269
dual	938	1.17E8	79.285	1801	1.07E8	96.251	839	1.19E8	63.423	1320	1.14E8	92.394	1577	1.09E8	79.259
fp	839	8.59E7	51.674	1591	8.28E7	58.973	773	9.05E7	51.146	1200	8.09E7	64.339	1569	7.77E7	64.648
fp16	813	7.92E7	46.538	1595	7.81E7	55.548	764	8.15E7	52.626	1160	7.41E7	59.519	1530	7.18E7	58.873
gemmini	865	9.82E7	58.868	1691	9.93E7	75.831	815	1.07E8	64.873	1230	9.49E7	72.816	1555	8.95E7	70.511
infer	857	1.00E8	58.581	1789	9.96E7	79.630	804	1.16E8	61.050	1230	9.13E7	78.117	1475	8.97E7	71.984
inferb	1422	2.69E8	146.723	2801	2.62E8	219.300	1135	2.73E8	148.003	1900	2.84E8	144.545	1539	2.58E8	153.489
inferp	1116	1.93E8	141.373	2193	1.82E8	186.586	955	2.62E8	158.840	1540	2.02E8	187.782	1565	1.83E8	160.481
lean	852	8.66E7	47.806	1619	8.33E7	62.398	817	8.19E7	48.156	1170	7.84E7	68.964	1478	7.37E7	62.033
train	1027	1.56E8	129.297	1997	1.47E8	134.405	875	2.07E8	140.843	1410	1.52E8	141.622	1609	1.47E8	152.671
Avg Ratio	0.604	1.099	0.810	1.180	1.060	1.018	0.543	1.212	0.860	0.846	1.052	1.023	1.000	1.000	1.000



(a) (b)

Fig. 8. Impact of regularity parameters on placement quality and convergence: the influence of (a) initial regularity and (b) regularity step size interval on HPWL.

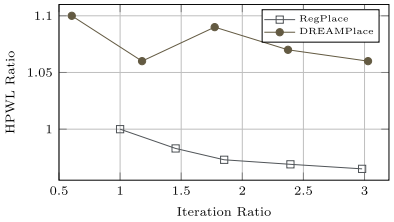


Fig. 9. Impact of iteration count on placement quality.

TABLE IV
POST-PLACEMENT METRICS COMPARISON WITH DREAMPLACE 4.1.0 (LONG)

Testcase	DREAMPlace 4.1.0 [16] (long)		RegPlace	
	WL(um)	Congestion (%)	WL (um)	Congestion (%)
bf16	2.00E+07	0.04	1.88E+07	0.03
bf16p	2.10E+07	0.03	1.91E+07	0.03
bf168	2.73E+07	0.02	2.69E+07	0.02
dual	2.76E+07	0.03	2.76E+07	0.03
fp	2.19E+07	0.04	2.09E+07	0.02
fp16	2.06E+07	0.05	1.90E+07	0.03
gemmini	2.56E+07	0.04	2.37E+07	0.02
infer	2.61E+07	0.02	2.41E+07	0.02
inferb	6.68E+07	5.15	6.58E+07	5.04
inferp	4.72E+07	0.05	4.68E+07	0.04
lean	2.23E+07	0.03	1.99E+07	0.01
train	3.87E+07	0.04	3.81E+07	0.03
Avg Ratio	1.053		1.000	

We examined adjusting sub_iterations (from 1 to 5) for RegPlace and DREAMPlace. Fig. 9 shows RegPlace's HPWL consistently decreases with iterations until stabilizing, while DREAMPlace's performance is less consistent, occasionally showing HPWL increases. RegPlace demonstrates more stable convergence and better HPWL results, highlighting the benefits of variance-based soft regularity constraints. Due to additional regularity computations, RegPlace's runtime is slightly longer than DREAMPlace's, as shown in Table III.

IV. CONCLUSION

Placement is a crucial step in VLSI physical design flow. Traditional nonlinear placers neglect the regularity of spatial arrays in DNN accelerators, leading to suboptimal solutions. We propose a new scenario for full-system DNN accelerators where regularity is important but not dominant due to the relatively small area percentage of spatial arrays. Our contributions include a spatial array extraction method utilizing netlist hierarchy and GCN, and a variance-based soft regularity constraint to guide the placement optimization. Experimental results demonstrate that compared to the state-of-the-art DREAMPlace, our approach achieves a 9.9% wirelength reduction with acceptable runtime overhead, and a 6% reduction compared to the long version within comparable runtime.

REFERENCES

[1] R. Venkatesan et al., "MAGNet: A modular accelerator generator for neural networks," in *Proc. ICCAD*, 2019, pp. 1–8.
[2] H. Ye, X. Zhang, Z. Huang, G. Chen, and D. Chen, "HybridDNN: A framework for high-performance hybrid DNN accelerator design and implementation," in *Proc. DAC*, 2020, pp. 1–6.
[3] H. Genc et al., "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration," in *Proc. DAC*, 2021, pp. 769–774.
[4] Y. Pu et al., "IncreMacro: Incremental macro placement refinement," in *Proc. ISPD*, 2024, pp. 169–176.
[5] D. Fang, B. Zhang, H. Hu, W. Li, B. Yuan, and J. Hu, "Global placement exploiting soft 2D regularity," in *Proc. ISPD*, 2022, pp. 203–210.
[6] H. Cai, S. Note, P. Six, and H. De Man, "A data path layout assembler for high performance DSP circuits," in *Proc. DAC*, 1991, pp. 306–311.
[7] X. Chen, J. Hu, and N. Xu, "Regularity-constrained floorplanning for multi-core processors," in *Proc. ISPD*, 2011, pp. 99–106.
[8] S. I. Ward et al., "Keep it straight: Teaching placement how to better handle designs with datapaths," in *Proc. ISPD*, 2012, pp. 79–86.
[9] S. Chou, M.-K. Hsu, and Y.-W. Chang, "Structure-aware placement for datapath-intensive circuit designs," in *Proc. DAC*, 2012, pp. 762–767.
[10] A. B. Kahng and Z. Wang, "DG-RePlAce: A dataflow-driven GPU-accelerated analytical global placement framework for machine learning accelerators," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 44, no. 2, pp. 696–708, Feb. 2025.
[11] R. X. Nijssen and J. A. Jess, "Two-dimensional datapath regularity extraction," in *Proc. IFIP Workshop Logic Archit. Synthesis*, 1996, pp. 110–117.
[12] H. Xiang, M. Cho, H. Ren, M. Ziegler, and R. Puri, "Network flow based datapath bit slicing," in *Proc. ISPD*, 2013, pp. 139–146.
[13] S. Ward, D. Ding, and D. Z. Pan, "PADE: A high-performance placer with automatic datapath extraction and evaluation through high dimensional data learning," in *Proc. DAC*, 2012, pp. 756–761.
[14] J. Lu et al., "ePlace: Electrostatics-based placement using fast fourier transform and Nesterov's method," *ACM Trans. Design Autom. Electron. Syst.*, vol. 20, no. 2, pp. 1–34, 2015.
[15] Y. Lin, S. Dhar, W. Li, H. Ren, B. Khailany, and D. Z. Pan, "DREAMPlace: Deep learning toolkit-enabled GPU acceleration for modern VLSI placement," in *Proc. DAC*, 2019, pp. 1–6.
[16] Y. Chen, Z. Wen, Y. Liang, and Y. Lin, "Stronger mixed-size placement backbone considering second-order information," in *Proc. ICCAD*, 2023, pp. 1–9.

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on February 03,2026 at 02:27:39 UTC from IEEE Xplore. Restrictions apply.