NEO Placer: ML-driven Thermal-aware Placement Considering Vertical Coupling in 3D ICs

Ming-Ru Wu * Norman Chang † Haiyang He † Akhilesh Kumar † Lang Lin † Jie Yang † Wenbo Xia † Jessica Yen † Yu-Chen Lin † and Jyh-Shing Roger Jang *

* Department of Computer Science and Information Engineering, National Taiwan University, Taiwan (R.O.C.)
† Ansys Inc.

***Abstract*—This paper presents the NEO (NEural Optimization-based) placer, a thermal-aware floorplanning/placement system for 3D ICs, by combining a set of novel methods with the powerful auto-gradient capability of modern machine learning (ML) frameworks. We employ a new pure neural network approach featuring the minimalist design in wirelengths and overlaps and the maximin (maximization of the minimum distance) and per-tile activation designs in temperatures. We discover that exact HPWL (half perimeter wire length) and overlap formulas can be used for auto gradients in ML without employing the differentiable weighted-average (WA) or log-sum-exp (LSE) approximation in most modern analytic placers. We also propose a new formula for the inter-block corner distance suitable for placing hotter blocks further apart to indirectly reduce the maximum temperature, T_{max}. The ML thermal solver based on parallel per-tile activation is applied for direct minimization of T_{max}. Our placer is extended to multiple chiplets for efficient analysis of vertical coupling in 3D ICs. Experimental results verify the validity of our NEO placer that outperforms the placer based on simulated annealing (SA) by reducing more than 5% in T_{max}.**

***Index Terms*—ML-driven, thermal-aware, floorplanning, placement, neural network, auto gradient, minimum distance, maximin, 3D IC**

I. Introduction

Thermal solvers (TS) play an important role in 3D IC designs by resolving temperature distributions and hot spots [1]–[6]. Fast ML-driven thermal analysis leveraging the linear superposition principle and per-tile activation scenarios has been developed to investigate peak temperatures and thermal gradients in 3D ICs. A reference model integrating a thermal solver and a thermal-aware placer (TP) is shown in Fig. 1 [7] [8]. TS generates thermal profiles for TP, and in turn, TP optimizes the power maps of heat sources for TS, where the ML thermal solver and the decay curve generator produce coarse-grained and local refinement thermal profiles, respectively. In view of the reference design, this paper aims at the thermal-aware placer with extensible applications for vertical coupling in 3D ICs.

A thermal-aware placer puts blocks on a chip to minimize the total wire length, overlap and peak temperatures [9]–[11] (see Fig. 2a). Taking the input information of (w, h, p) (width, height and power) and netlists of blocks, it generates optimized block positions (x, y) and rotations r with balanced temperature distributions. In comparison, a traditional placer

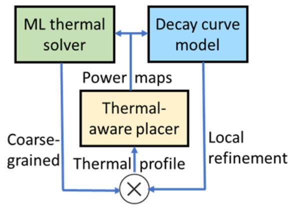

Fig. 1: A reference model consists of a thermal-aware placer (TP) generating power maps and a thermal solver (TS) having a coarse-grained ML TS with local refinement by a decay curve model.

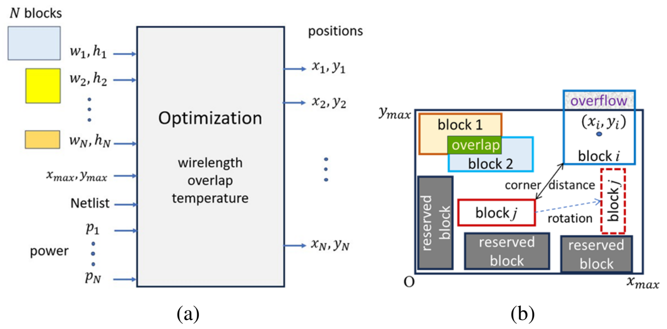

Fig. 2: (a) Thermal-aware floorplanning of N blocks. (b) Placement of blocks on a chiplet of size $x_{max} \times y_{max}$.

(see Fig. 2b) aims to find the best (x, y, r) [12]–[15] without considering the effect of powers p on temperature profiles.

Modern floorplanning and placement schemes can be roughly classified into two categories: analytic force-directed (AFD) and reinforcement learning (RL)-based. In the AFD scheme, differentiable approximations such as weighted-average [16] or log-sum-exp [17] are made on HPWL to facilitate computations of gradients at the (x, y) positions of blocks. Use of density fields similar to that in electrostatics leads to efficient force-directed analytic placers [18]–[28]. A simple fast thermal placer is developed in [29], using repulsive and attractive forces for hot and cold blocks like nuclear forces. In the RL-based scheme, blocks are placed in a grid world [30]–[32] or based on overlap-free floorplan representations such as sequence pairs (SP) [12] [33] [34]. Usually, massive computing resources are required due to the huge space of states and actions as well as the challenges in learning the pretrained neural network for deep RL.

Metrics for the optimization of thermal-aware floorplanning include HPWL, the overall area and overlaps as well as the peak temperature, T_{max}. Although analytic formulas of HPWL and overlaps are well-known [13] [15], temperature variations due to block powers have not yet been thoroughly characterized. It can be observed, however, that the peak temperature on a chiplet is mostly determined by the distances between hot blocks having high power densities [29]. Thus, in this paper, we maximize the minimum corner distance (see Fig. 2b) among hot blocks so as to place hotter blocks further apart. The success of the maximin criterion in areas such as game theory [35] and error-correcting maximum distance codes [36] can be expected to provide a new method for efficient thermal-aware floorplanning. On the other hand, we also employ our per-tile activation thermal solver [7] to directly compute T_{max} from temperature profiles based on linear superposition.

Neural networks (NN) [37] with gradient-based backpropagation [38] [39] are generalized differentiable machines that have greatly influenced fundamental ways of thinking in the scientific society [40] [41]. In NN, the strict-sense differentiability of Calculus has been extended to accommodate special (continuous) functions such as max, min, ReLU ($[\cdot]^+$) and absolute value ($|\cdot|$) functions. Thus, it is possible to design neural networks based on these extended differentiable functions of machine learning frameworks such as PyTorch [42] and Tensorflow [43]. Noting that the calculations of HPWL and overlaps involve these special functions only, we can therefore use PyTorch for precise parametric optimization of floorplanning.

Since netlists are hypergraphs [44], graph neural networks (GNN) [45] [46] or hypergraph neural networks (HNN) [47] have been frequently employed for feature extraction in floorplanning/placement [31] - [34]. Graph attention networks (GAT) [48] [49] with an interpretable structure [50] featuring direct correspondence of nodes and blocks are attractive for embedding in netlists.

In this paper, we propose the NEO placer by fully exploiting the auto-gradient capability of ML frameworks such as PyTorch with a set of novel design methods for thermal-aware floorplanning.

The contributions of this paper are described as follows.

1) We employ a pure neural network approach featuring exact gradients of HPWL and overlaps with auto gradient support of PyTorch, not having to use differentiable approximations for conventional AFD schemes.
2) While the force-directed scheme outputs block positions (x, y) only, we also generate block rotations (r) with the trick for auto gradients to achieve flexibility and functional enhancement.
3) We derive novel formulas for the corner distance $\sqrt{q_{i,j}}$ and minimum power-density-dependent distance $Q_{k,k}$ to fulfill the thermal-aware maximin criterion.
4) Our neural optimizer can be readily extended to thermal-aware floorplanning of 3D IC with vertical coupling.

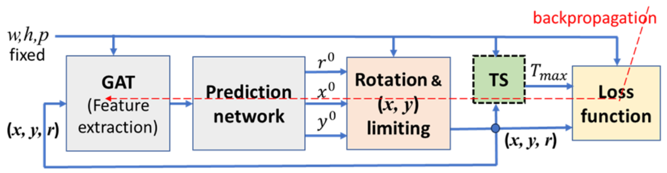

Fig. 3: System Architecture of our NEO placer.

5) In combination with the per-tile activation based thermal solver using the distributed heat transfer coefficient (HTC) [7], our placer offers a promising solution for optimization of temperature profiles.

We use the commercial thermal simulation tool [5] and ML thermal solver [7] to validate the temperature distributions for our 3D IC floorplanning results. Note that floorplanning and placement are used interchangeably in the context although floorplanning usually refers to the early stage circuit design with fewer but larger blocks.

II. System Architecture of NEO Placer

In Fig. 3, we show our feedback realization of the general thermal-aware floorplanning system.

At the first stage, a graph neural network extracts a feature vector f from the (fixed) widths, heights and powers (w, h, p), and current positions and rotations (x, y, r) of the N blocks, as well as the (fixed) netlist representing interconnection of blocks. We adopt the multi-head graph attention network (GAT) [48] for feature extraction since it is simple and interpretable: each block is associated with a node in GAT and the input netlist provides the basic connections of the nodes. We also insert links between successive nodes i and $mod(i, N)+1$, $i = 1, \ldots, N$, whichever missing in the netlist, chaining all the N nodes in a ring so that they are connected with degrees ≥ 2. Let $||$ denote concatenation, $\mathcal{M}$ the number of heads, $\mathcal{N}(i)$ the set of neighboring nodes for node i, and $\sigma(\cdot)$ the softmax function. The output at node i of GAT is the concatenation of features from head $m \in \{1, \ldots, \mathcal{M}\}$:

$$\mathbf{f}_i^{(t+1)} = \Big\Vert_{m=1}^{\mathcal{M}} \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{i,j}^m \mathbf{W}^m \mathbf{f}_j^{(t)}\right), \quad (1)$$

where $\alpha_{i,j}^m = \sigma(u_{i,j}^m)$ over $j \in \mathcal{N}(i)$; $u_{i,j}^m = \text{LeakyReLU}(\mathbf{a}^T[\mathbf{W}^m \mathbf{f}_i^{(t)} || \mathbf{W}^m \mathbf{f}_j^{(t)}])$; $\mathbf{a}$ and $\mathbf{W}^m$ are, respectively, the adjustable weighting vector and matrix.

Secondly, a prediction network (PN) converts the feature vector into a primitive estimate (x^0, y^0, r^0), where the sigmoid function $S(t) = \frac{1}{1+e^{-t}}$ is used by the end of PN to confine the output range to [0,1].

At the third stage, we design a rotation and (x, y) limiting module based on novel tricks for auto-gradients to compute block rotations and positions free from overflows.

At the final stage, we implement the loss function $\mathcal{L}$ in (2) for neural networks in terms of PyTorch functions with auto-gradients for backpropagation:

$$\mathcal{L} = HPWL + \lambda_o Overlap + \lambda_T L_T, \quad (2)$$

TABLE I: Gradients in PyTorch

no gradient	auto gradient
$argmax$	max
$argmin$	min
if $z < 0$ then $z = 0$	ReLU(z), $[z]^+$
if $z < 0$ then $z = -z$	abs(z), $\lvert z \rvert$

TABLE II: PyTorch functions for auto gradients in loss terms

Loss	related formula	max	min	ReLU	abs
HPWL	$\max \lvert A - B \rvert$	✓			✓
overlap	$[\min(B, D) - \max(A, C)]^+$	✓	✓	✓	
L_T	$\max([A - D]^+, [C - B]^+)$	✓		✓	
	Q_{min}		✓		

where we use the maximin criterion or per-tile activation for L_T, and the minimalist design for HPWL and Overlap using exact formulas.

The last two stages provide major novelty for our NEO placer and will be elaborated in Section III.

The thermal solver (TS) module in Fig. 3 is based on the parallel per-tile activation method [7] described in Sec. III-D.

III. Neural Optimization using Auto Gradients

A neural network is based on the backpropagation algorithm that computes gradients of the loss function to update its weights. A machine learning framework such as PyTorch or Tensorflow supports automatic calculations of gradients for built-in functions such as $\min, \max$, abs ($|\cdot|$), sgn and ReLU (rectified linear unit, $[\cdot]^+$), but has no gradient for discrete indices of elements ($argmax, argmin$) and arbitrary if clauses (see Table I). Since the calculations of HPWL, overlaps and minimum distances involve the operations of max, min, abs and ReLU, we can utilize the auto gradient support for built-in functions in PyTorch to achieve a compact design of thermal-aware floorplanning. Table II summarizes the special PyTorch functions used for auto gradients in (2), (9), (12) and (14).

A. Rotations and (x, y) Positions

Besides the (x, y) positions, our floorplanning system also supports block rotation r (see Fig. 3). For each block $i \in \{1, \ldots, N\}$ in Fig. 4a, $r_i = 1$ stands for rotation (i.e., swap(w_i, h_i)) and $r_i = -1$ for no rotation. However, generating the discrete indices r_i, $i = 1, \ldots, N$, for auto gradients is not simple and straightforward in PyTorch, as explained in the following.

The rotation & (x, y) limiting module in Fig. 3 uses the primitive rotation information $r_i^0 \in [0, 1]$ provided by the prediction network to perform the conditional swap operation: if $r_i^0 \geq 0.5$, then $swap(w_i, h_i)$. Direct implementations of this operation will encounter the 'no gradient' problem for the arbitrary 'if' clause in PyTorch.

Define

$$\begin{pmatrix} \tilde{w}_i \\ \tilde{h}_i \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 - r_i & 1 + r_i \\ 1 + r_i & 1 - r_i \end{pmatrix} \begin{pmatrix} w_i \\ h_i \end{pmatrix} \tag{3}$$

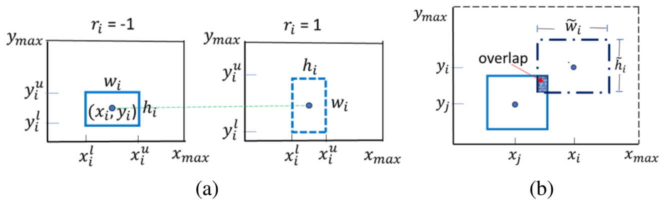

Fig. 4: (a) Configurations for rotation $r = \mp 1$ of block i. (b)The overlap area of block i and block j is a rectangle of width $\Theta\left(x_i^l, x_i^u, x_j^l, x_j^u\right)$ and height $\Theta\left(y_i^l, y_i^u, y_j^l, y_j^u\right)$.

so as to fit the conditional swap operation

$$\left(\tilde{w}_i, \tilde{h}_i\right) = \begin{cases} (w_i, h_i), & r_i = -1 \\ (h_i, w_i), & r_i = 1, \end{cases} \tag{4}$$

where $r_i = 1$ and $r_i = -1$ correspond to the conditions $r_i^0 \geq 0.5$ and $r_i^0 < 0.5$, respectively.

For block i (see Fig. 4a), we define the following rotation-dependent positions: $x_i^l = x_i - \frac{\tilde{w}_i}{2}$, $x_i^u = x_i + \frac{\tilde{w}_i}{2}$, $y_i^l = y_i - \frac{\tilde{h}_i}{2}$, and $y_i^u = y_i + \frac{\tilde{h}_i}{2}$, where

- (x_i^l, y_i^l) represents the lower left corner of block i;
- (x_i^u, y_i^u) denotes the upper right corner of block i;
- $(x_i, y_i) = (x_i^l + \frac{\tilde{w}_i}{2}, y_i^l + \frac{\tilde{h}_i}{2})$ is the center of block i.

We can bypass the 'no-gradient' problem by using the tricks for auto gradients of rotations, as summarized in the following three-step procedure:

1) To generate the conditions of r_i in (4), we need $r_i = s(r_i^0 - 0.5)$, where

$$s(z) = \begin{cases} -1, & z < 0 \\ 1, & z \geq 0 \end{cases} \tag{5}$$

so that $r_i = 1$ for $r_i^0 \geq 0.5$, and $r_i = -1$ for $r_i^0 < 0.5$. Noting that $s(z)$ resembles the sign function

$$sgn(z) = \frac{z}{|z|} = \begin{cases} -1, & z < 0 \\ 0, & z = 0 \\ 1, & z > 0, \end{cases} \tag{6}$$

we select $s(z)$ as the numerically stable function:

$$s(z) \doteq 2\frac{L[z]^+ + \epsilon}{L|z| + \epsilon} - 1, \tag{7}$$

accurate for $z \notin (-\epsilon, 0)$ with $0 < \epsilon \ll 1 \ll L$.

2) Use r_i to perform the conditional swap operation (3).

3) Based on the primitive (x_i^0, y_i^0) position from the prediction network in Fig. 3, where $x_i^0, y_i^0 \in [0, 1]$, we use the following (x, y) limiting operation to generate the lower left corner (x_i^l, y_i^l):

$$(x_i^l, y_i^l) = \left(x_i^0(x_{max} - \tilde{w}_i), y_i^0(y_{max} - \tilde{h}_i)\right), \tag{8}$$

so that $x_i^l \in [0, x_{max} - \tilde{w}_i]$ and $y_i^l \in [0, y_{max} - \tilde{h}_i]$, guaranteeing no overflow ($x_i^u \leq x_{max}$ and $y_i^u \leq y_{max}$) for each block $i \in \{1, \ldots, N\}$.

The above-mentioned three-step procedure constitutes the rotation & (x, y) limiting module in Fig. 3 and streamlines backpropagation operations of neural networks.

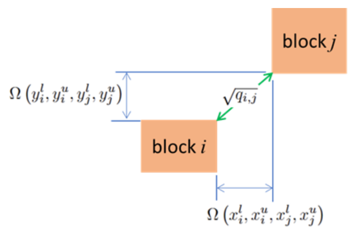
block j
$\Omega\left(y_i^l, y_i^u, y_j^l, y_j^u\right)$
$\sqrt{q_{i,j}}$
block i
$\Omega\left(x_i^l, x_i^u, x_j^l, x_j^u\right)$

Fig. 5: Corner distance $\sqrt{q_{i,j}}$.

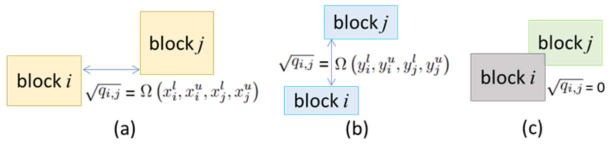
block i
block j
$\sqrt{q_{i,j}} = \Omega\left(x_i^l, x_i^u, x_j^l, x_j^u\right)$
(a)
block j
$\sqrt{q_{i,j}} = \Omega\left(y_i^l, y_i^u, y_j^l, y_j^u\right)$
block i
(b)
block j
block i
$\sqrt{q_{i,j}} = 0$
(c)

Fig. 6: Three special cases of corner distances $\sqrt{q_{i,j}}$ when blocks i and j have (a) a gap in x only, (b) a gap in y only, or (c) no gap (overlap).

B. Loss Functions using Exact HPWL and Overlap

1) HPWL: A netlist for a circuit of N blocks is a hypergraph consisting of N vertices (blocks), and a set of hyperedges E containing signal nets. From [13], we have

$$HPWL(\mathbf{v}) = \sum_{e \in E} \left(\max_{i,j \in e} |x_i - x_j| + \max_{i,j \in e} |y_i - y_j| \right), \quad (9)$$

where $\mathbf{v} = ((x_1, y_1), \ldots, (x_N, y_N))$, and (x_n, y_n) is the position for the center of block-n.

For PyTorch implementations, we can use the max and abs functions with auto gradients in Table I for exact computations of HPWL in (9) without having to make the commonly used weighted average (WA) approximation in HPWL for differentiability [18] [25].

2) Overlap: From [13], the total overlap area, $Overlap$, is obtained by accumulating all the overlap area of blocks i and j, denoted by $O_{i,j}$ (see Fig. 4b), as follows.

$$Overlap = \sum_{1 \le i < j \le N} O_{i,j}; \quad (10)$$

$$O_{i,j} = \Theta\left(x_i^l, x_i^u, x_j^l, x_j^u\right) \cdot \Theta\left(y_i^l, y_i^u, y_j^l, y_j^u\right); \quad (11)$$

$$\Theta(A, B, C, D) = [\min(B, D) - \max(A, C)]^+. \quad (12)$$

The min, max and ReLU functions in (12) are available for auto gradients in PyTorch (see Table I).

C. Loss Function L_T for Temperatures: Maximin Design

Since hot blocks having high power densities are the major sources of high temperatures, they need to be placed further apart from each other for reduction of the maximum temperature T_{max}. Therefore, maximization of the (power-dependent) minimum distance among hot blocks serves as a good criterion for minimization of T_{max}. While the conversion from powers of the blocks into chiplet temperatures is complicated, we employ this new criterion to attain efficient peak temperature reduction by deriving new formulas leading to our loss function for temperatures L_T.

A simple-minded choice of the inter-block distance is the Euclidean distance between the centers of block-i and block-j, $d_{i,j}$. However, $d_{i,j} > 0$ can occur even when blocks i and j overlap, i.e., $|x_i - x_j| < \frac{\tilde{w}_i + \tilde{w}_j}{2}$ and $|y_i - y_j| < \frac{\tilde{h}_i + \tilde{h}_j}{2}$.

As shown in Fig. 5, the distance $\sqrt{q_{i,j}}$ between the corners of block-i and block-j can be used to distinguish the above-mentioned overlap condition. We derive the squared distance between the corners of block-i and block-j as follows.

$$q_{i,j} = \Omega^2\left(x_i^l, x_i^u, x_j^l, x_j^u\right) + \Omega^2\left(y_i^l, y_i^u, y_j^l, y_j^u\right); \quad (13)$$

$$\Omega(A, B, C, D) = \max([A - D]^+, [C - B]^+). \quad (14)$$

Note that (14) involves max and ReLU functions that have auto gradients in PyTorch. Three special cases for corner distances are depicted in Fig. 6.

Consider a 3D IC formed by stacking K chiplets of size $x_{max} \times y_{max}$. Let k_i denote the index of the chiplet where block-i resides. We have $k_i \in \{1, \ldots, K\}$.

Denote the power density of block-i as $\rho_i = \frac{p_i}{A_i}$, where $A_i = w_i h_i$. A block is said to be hot if its power density is above the average power density $\rho_0 = \frac{\sum_{i=1}^N p_i}{K x_{max} y_{max}}$. The set of hot blocks on chiplet k, H_k, is given by $H_k = \{i: \ \rho_i \ge \rho_0, \ k_i = k\}$.

The effect of vertical coupling is reflected by the inter-chiplet distance, denoted as z_{k_i,k_j}. We have $z_{k_i,k_i} = 0$. The multi-chiplet squared distance between the corners of block i and block j is given as $q_{i,j} + z^2_{k_i,k_j}$, where $k_i = k_j$ and $k_i \ne k_j$, respectively, signify the intra and inter-chiplet cases. The effective multi-chiplet minimum corner distance is

$$Q_{k,\kappa} = \min_{\substack{i \in H_k \\ j \in H_\kappa \\ i \ne j}} \frac{q_{i,j} + z^2_{k_i,k_j}}{\mu_{i,j}}, \quad (15)$$

where $\mu_{i,j}$ is the power-density-dependent factor defined as

$$\mu_{i,j} = \frac{p_i + p_j}{\rho_0 (A_i + A_j)}. \quad (16)$$

It can be shown that $\mu_{i,j} \ge 1$ since $\rho_0 \le \frac{p_i}{A_i} \le \frac{p_i + p_j}{A_i + A_j} \le \frac{p_j}{A_j}$ for $\rho_i \le \rho_j, \quad i \in H_k, \ j \in H_\kappa$. Hotter blocks having higher power densities are associated with higher levels of $\mu_{i,j}$, which, in turn, reduce $\frac{q_{i,j} + z^2_{k_i,k_j}}{\mu_{i,j}}$, the effective squared corner distance between hot blocks i and j. Thus, hotter blocks are more likely to affect $Q_{k,\kappa}$ in (15). With the maximization of $Q_{k,\kappa}$, hotter blocks will be placed further apart globally as desired.

We define L_T, the multi-chiplet loss function for temperatures as an exponential function of intra and inter-chiplet terms in (15) in the following.

$$L_T = f(\hat{Q}_K), \quad (17)$$

where $\hat{Q}_K = (Q_{k,\kappa}, 1 \le \kappa \le k \le K)$, and $f(\mathbf{z})$ is a strictly decreasing function of $\mathbf{z}$. We select

$$L_T = \frac{1}{\Delta_K} \sum_{k=1}^{K} \sum_{\kappa=1}^{k} e^{-Q_{k,\kappa}} \quad (18)$$

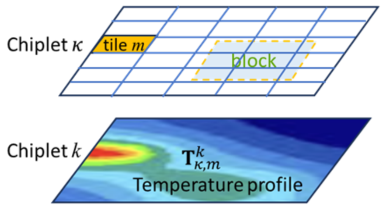

Fig. 7: The temperature profiles $\mathbf{T}^k_{\kappa,m}$ of resolution $W \times H$ pixels, for $m = 1, \ldots, M$, $k, \kappa = 1, \ldots, K$, are precomputed and serve as the base matrices for parallel superposition.

to allow a sufficient number, $\Delta_K = \frac{K(K+1)}{2}$, of update paths for backpropagation of gradients with respect to $Q_{k,\kappa}$.

D. Loss Function L_T for Temperatures: Per-Tile Activation

The maximin-based analysis for L_T in III-C uses the corner distances between hot blocks to indirectly reduce the peak temperature T_{max} through the loss function in (17) at the last stage of Fig. 3. We can indeed obtain a direct measure of T_{max} by employing the per-tile activation method [7] for the thermal solver (TS) in Fig. 3.

Let the chiplet area of size $x_{max} \times y_{max}$ be divided into $M = M_x M_y$ tiles, each having size $\frac{x_{max}}{M_x} \times \frac{y_{max}}{M_y}$.

Let $\mathbf{T}^k_{\kappa,m}$ denote the $W \times H$ rising temperature (above ambient) profile matrix of the k-th chiplet solely due to tile m of (unit) power P_0/M on the κ-th chiplet of power P_0. There are a total of MK^2 matrices for $\mathbf{T}^k_{\kappa,m}$ that are pretrained as the base matrices of linear interpolation.

To find the rising temperature profile $\mathbf{T}^k$ on the k-th chiplet, the per-tile activation method linearly interpolates all the precomputed $\mathbf{T}^k_{\kappa,m}$ (see Fig. 7 and (21)) in a highly parallel manner, suitable for high-performance GPU applications.

Let $(\hat{x}^l_m, \hat{y}^l_m)$ and $(\hat{x}^u_m, \hat{y}^u_m)$, respectively, denote the lower-left and upper-right corners of tile m on the k-th chiplet. The overlap area between block n and tile m on the k-th chiplet is given by

$$\hat{O}^k_{m,n} = \Theta\left(\hat{x}^l_m, \hat{x}^u_m, x^l_n, x^u_n\right) \Theta\left(\hat{y}^l_m, \hat{y}^u_m, y^l_n, y^u_n\right), \tag{19}$$

for $n \in B_k$, and $\hat{O}^k_{m,n} = 0$, $n \notin B_k$, where B_k is the set of blocks on the k-th chiplet.

The weight for tile m on the κ-th chiplet is computed by accumulating the powers from all the blocks overlapping with this tile as follows.

$$\beta_{\kappa,m} = \frac{M}{P_0} \sum_{n \in B_\kappa} p_n \frac{\hat{O}^\kappa_{m,n}}{w_n h_n}. \tag{20}$$

Note that the area of block n is equal to the sum of overlap areas between block n and each of the M tiles on a chiplet: $w_n h_n = \sum_{m=1}^{M} \hat{O}^\kappa_{m,n}$, $n \in B_\kappa$.

We can therefore compute the overall $W \times H$ rising temperature profile (matrix) on the k-th chiplet by linear interpolation:

$$\mathbf{T}^k = \sum_{\kappa=1}^{K} \sum_{m=1}^{M} \beta_{\kappa,m} \mathbf{T}^k_{\kappa,m}. \tag{21}$$

Let $T^k_{i,j}$ denote the (i,j)-th element of $\mathbf{T}^k$. We minimize the maximum temperatures across multiple chiplets by defining the loss function for temperatures as

$$L_T = T_0 + \max_{k=1,\ldots,K} T^k_{max}, \tag{22}$$

where T_0 is the ambient temperature and $T^k_{max} = \max_{\substack{i=1,\ldots,W \\ j=1,\ldots,H}} T^k_{i,j}$.

Note that only the special PyTorch function max is used in (22). Also, the matrices $\mathbf{T}^k$ and $\mathbf{T}^k_{\kappa,m}$ with uniform sampling in (21) can be replaced by data structures associated with non-uniform sampling [5] [7] in practice.

Modeling of vertical coupling in 3D ICs for the inter-chiplet terms $\mathbf{T}^k_{\kappa,m}$, $\kappa \neq k$, for $K = 2$ in (21) is described below.

Given the stacked chiplet configurations, a global run with 200 μm × 200 μm tile resolution will be performed using the thermal simulator [5] to extract the distributed heat transfer coefficient (HTC) at the interface between the top and bottom chiplets. Assuming a linear relationship between temperature change and small variations in HTC, an assumption validated through experiments, we begin with initial temperature predictions for the top and bottom chiplets, denoted as T_1 and T_2, respectively, obtained from the machine learning solver.

Using the extracted distributed HTC (HTC_{dist}) and the reference HTC used in the training data (HTC_{ref}), we compute a scaling factor as: scaling_factor = HTC_{ref} / HTC_{dist}.

The predicted temperatures T_1 and T_2 are then scaled elementwise using this factor to produce updated temperatures: $T_1' = T_1 \times$ scaling_factor; $T_2' = T_2 \times$ scaling_factor. (If necessary, an additional uniform global_scaling_factor, empirically determined, is applied to T_1' and T_2'. By default, this global scaling factor is set to 1.0.)

IV. Experimental Results

We examine the performance of our NEO placer by considering block numbers $N = 10, 20, 30, 40,$ and 50 for $K = 2$ chiplets, each of which is associated with $N/2$ blocks. We set $\lambda_o = 100$ and $\lambda_T = 670$ initially to balance the numerical ranges of HPWL, overlap, and L_T. The heat transfer coefficient is set to be 700 $\frac{mW}{\mu m^2 \cdot {}^\circ K}$ and the inter-chiplet distance $z_{1,2} = 0.16$ mm. To fully exploit the auto gradient capability, the special functions, max, min, ReLU, and abs, are designed to operate on the 'tensor' data structure in PyTorch. The experiments are carried out on a personal computer with an Intel Core i7 CPU and an NVIDIA GeForce RTX 4090 card.

Owing to the inherent property of neural networks based on gradients that attain local minima, multiple runs help reduce loss functions. For each block number N, we run 30 experiments with random initialization, each of which takes 3000 iterations. The best result corresponding to the lowest loss L is then selected as the output of our NEO placer.

For the purpose of performance comparison, we also developed a placer based on simulation annealing (SA) with the sequence pair (SP) representation. The run time for the SA placer is set to be approximately equal to that of our placer so as to provide a fair comparison.

TABLE III: Thermal-aware Floorplanning Results

circuit	Simulated annealing					NEO				
	WL	Temperature (C)			Runtime	WL	Temperature (C)			Runtime
	(mm)	Max	Min	Range	(s)	(mm)	Max	Min	Range	(s)
n10	53	75.67	53.11	22.56	121	53	73.03	55.3	17.73	125
n20	169	77.41	58.4	19.01	245	151	74.92	61.29	13.63	243
n30	280	74.15	60.31	13.84	365	231.5	71.55	64.4	7.15	362
n40	488	80.47	64.28	16.19	492	437	74.59	69.1	5.31	482
n50	812	82.36	65.05	17.31	619	567	75.05	68.71	6.34	605

TABLE IV: Floorplanning Results with $\lambda_T = 0$

circuit	WL	Temperature (C)			Runtime
	(mm)	Max	Min	Range	(s)
n10	47	78.46	49.77	28.69	122
n20	156	81.27	59.16	22.11	241
n30	236	73.07	61.59	11.48	360
n40	410	77.25	66.72	10.53	482
n50	579	79.11	66.89	12.22	603

A. Comparison with simulated annealing (SA)

We compare the experimental results of our NEO placer with that of SA in Table III. We observe that the peak temperatures, T_{max}, of NEO are $73.03°C$ and $75.05°C$ for $N = 10$ and $N = 50$, respectively, whereas those of SA have higher values of $75.67°C$ and $82.36°C$. Our NEO placer excels SA by 5.3% in T_{max} on average and achieves smaller wire lengths (WL) and lower temperature ranges, $T_{max} - T_{min}$, in general.

In Fig. 8, we illustrate the power maps and temperature profiles for SA and NEO, respectively. Although the lower right hot blocks (in crimson) on top and bottom chiplets based on SA overlap in Fig. 8 (a) and (c), our NEO placer is able to scatter hot blocks on two chiplets, as shown in Fig. 8 (b) and (d). As a result, a hot spot emerges from the lower right corner in Fig. 8 (e) for SA, but the hot spots are found further away from each other in Fig. 8 (f) for NEO, thereby lowering T_{max}.

B. Ablation study

We conduct the ablation study by setting $\lambda_T = 0$ in (2) to examine the performance of our NEO placer without temperature optimization in Table IV. As compared to Table III, for $N = 10$ and $N = 50$ blocks, the maximum temperatures are elevated from $73.03°C$ to $78.46°C$ and from $75.05°C$ to $79.11°C$, respectively, while the temperature ranges from $17.73°C$ to $28.69°C$ and from $6.34°C$ to $12.22°C$. The effects of temperature optimization in our placer are prominent in lowering T_{max} and reducing temperature variations.

V. Conclusions

We have carried out a pure neural network approach for thermal-aware floorplanning. The proposed NEO placer is shown to be efficient in reducing the maximum temperature for 3D ICs with vertical coupling. We will examine the synergistic operations of the per-tile activation-based ML thermal solver, the decay curve model, and our thermal-aware NEO placer in further details to gain more insight into the complete reference model in Fig. 1.

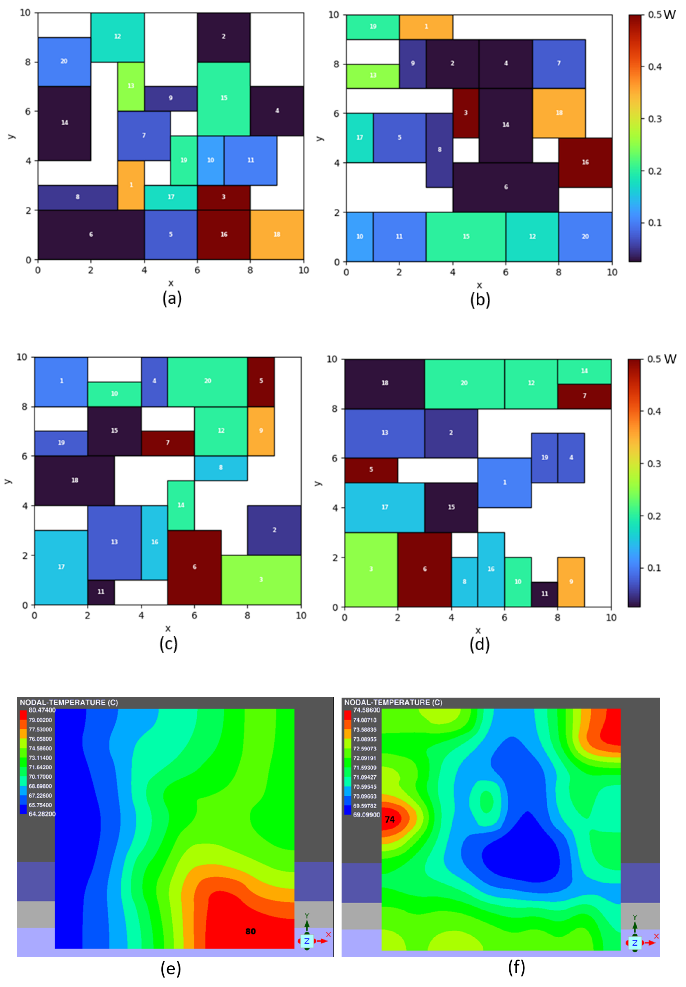

Fig. 8: Thermal results for simulated annealing and NEO placer with $N = 40$ blocks: Top chiplet power maps of 20 blocks: (a) SA and (b) NEO; Bottom chiplet power maps of 20 blocks: (c) SA and (d) NEO; Temperature profiles on the top chiplet: (e) SA and (f) NEO.

References

[1] H. Sultan, A. Chauhan, and S. R. Sarangi, “A survey of chip-level thermal simulators,” *ACM Comput. Surv.*, 2019.

[2] P Li, et al., ”IC thermal simulation and modeling via efficient multigrid-based approaches,” *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 2006.

[3] W. Huang, et al., ”HotSpot: a compact thermal modeling methodology for early-stage VLSI design,” *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, 2006.

[4] Z. Liu et al., ”DeepOHeat: operator learning-based ultra-fast thermal simulation in 3D-IC design,” *ACM/IEEE Design Automation Conf. (DAC)*, 2023.

[5] *RedHawk-SC-Electrothermal User Manual*. San Jose, CA, USA: Ansys Inc., 2024.

[6] R. Ranade, et al., ”A thermal machine learning solver for chip simulation,” *ACM/IEEE Workshop on Machine Learning for CAD (MLCAD)*, 2022.

[7] H. He, et al., ”Parallel per-tile activation with linear superposition of thermal response for solving arbitrary power pattern in 3DIC thermal simulation,” *ACM/IEEE Symposium on Machine Learning for CAD (MLCAD)*, 2024.

[8] H. He, et al., ”Invited paper: solving fine-grained static 3DIC thermal with ML thermal solver enhanced with decay curve characterization,” *IEEE/ACM Internat. Conf. Computer Aided Design (ICCAD)*, 2023.

[9] J. Cong, et al., ”Thermal-aware 3D IC placement via transformation,” *Asia and South Pacific Design Automation Conference*, 2007.

[10] D. Al Saleh, et al., ”P* admissible thermal-aware matrix floorplanner for 3D ICs,” *IEEE Internat. System-on-Chip Conf. (SOCC)*, 2023.

[11] Y. Safari, et al., ”Thermal simulator for advanced packaging and chiplet-based systems,” *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, 2025.

[12] C.J. Alpert, D.P. Mehta and S.S. Sapatnekar, *Handbook of Algorithms for Physical Design Automation*, CRC Press, 2008.

[13] L.T. Wang, Y.W. Chang and K.T. Cheng, *Electronic Design Automation: Synthesis, Verification, and Test*, Morgan Kaufmann, 2009.

[14] A.B. Kahng, J. Lienig, I.L. Markov and J. Hu, *VLSI Physical Design: From Graph Partitioning to Timing Closure*, Springer, 2022.

[15] I. L. Markov, et al., ”Progress and challenges in VLSI placement research,” *Proceedings of the IEEE*, 2015.

[16] M. -K. Hsu, Y. -W. Chang and V. Balabanov, ”TSV-aware analytical placement for 3D IC designs,” *ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2011.

[17] W. C. Naylor, R. Donelly and L. Sha, *Non-linear optimization system and method for wire length and delay optimization for an automatic electric circuit placer*, U.S. Patent 6 301 693, 2001.

[18] A. B. Kahng and Q. Wang, “Implementation and extensibility of an analytic placer,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 2005.

[19] T. -C. Chen, et al., ”NTUplace3: an analytical placer for large-scale mixed-size designs With preplaced blocks and density constraints,” *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 2008.

[20] J. Lu, et al., “ePlace: electrostatics-based placement using fast fourier transform and Nesterov‘s method,” *ACM Trans. Design Automation of Electronic Systems*, 2015.

[21] J. Lu, et al., “ePlace-MS: electrostatics-based placement for mixed-size circuits,” *IEEE Trans. Computer-Aided Design of Integrated Circuits Systems*, 2015.

[22] J. Lu, et al., “ePlace-3D: electrostatics based placement for 3D-ICs,” *International Symposium on Physical Design (ISPD)*, 2016.

[23] C. -K. Cheng, et al., ”RePlAce: advancing solution quality and routability validation in global placement,” *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 2019.

[24] J.M. Lin, et. al., ”Thermal-Aware fixed-outline floorplanning using analytical models with thermal-force modulation,” *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, 2021.

[25] Y. Lin, et al., ”DREAMPlace: deep learning toolkit-enabled GPU acceleration for modern VLSI placement,” *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 2021.

[26] P. Liao, et al., ”DREAMPlace 4.0: timing-driven placement With momentum-based net weighting and lagrangian-based refinement,” *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 2023.

[27] L. Liu, et al., “Xplace: an extremely fast and extensible placement framework,” *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[28] A. B. Kahng, et al., ”DG-RePlAce: A dataflow-driven GPU-accelerated analytical global placement framework for machine learning accelerators,” *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 2025.

[29] X. Liu, et al., ”ThePlace: Thermal-aware placement with operator learning-based ultra-fast simulator,” *Asia and South Pacific Design Automation Conference (ASPDAC)*, 2025.

[30] A. Mirhoseini, et al., ”Chip placement with deep reinforcement learning,” arXiv:2004.10746, 2020.

[31] A. Mirhoseini, et al., ”A graph placement methodology for fast chip design,” *Nature*, 2021.

[32] S. Yue, el al., ”Scalability and Generalization of Circuit Training for Chip Floorplanning,” *International Symposium on Physical Design*, 2022.

[33] W. Guan, et al., ”Thermal-aware fixed-outline 3-D IC floorplanning: an end-to-end learning-based approach,” *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, 2023.

[34] W. Guan, et al., ”ATT-TA: a cooperative multiagent deep reinforcement learning approach for TSV assignment in 3-D ICs,” *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, 2023.

[35] M. Maschler, E. Solan & S. Zamir, *Game Theory*, 2nd ed., Cambridge University Press, 2020.

[36] T.K. Moon, *Error Correction Coding*: *Mathematical Methods and Algorithms*, Wiley, 2005.

[37] R. Lippmann, ”An introduction to computing with neural nets,” *IEEE ASSP Magazine*, 1987.

[38] D.E. Rumelhart, G.E. Hinton and R.J. Williams, ”Learning representations by back-propagating errors,” *Nature*, 1986.

[39] A.G. Baydin, et al., ”Automatic differentiation in machine learning: a survey”, *Journal of Machine Learning Research*, 2018.

[40] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, MIT Press, 2016.

[41] K. Murphy, *Probabilistic Machine Learning: An Introduction*, Summit Valley Press, 2022.

[42] A. Paszke, et al., ”PyTorch: An imperative style, high-performance deep learning library,” *arXiv:1912.01703*, 2019.

[43] M. Abadi, et al., ”TensorFlow: A system for large-scale machine learning,” *arXiv:1605.08695*, 2016.

[44] X. Ouvrard, ”Hypergraphs: an introduction and review,” *arXiv: 2002.05014*, 2002.

[45] Z. Wu, et al., ”A comprehensive survey on graph neural networks,” *IEEE Trans. Neural Networks and Learning Systems*, 2021.

[46] B. Khemani, et al. ”A review of graph neural networks: concepts, architectures, techniques, challenges, datasets, applications, and future directions,” *J. Big Data*, 2024.

[47] S. Kim, et al., “A survey on hypergraph neural networks: an in-depth and step-by-step guide,” *ACM SIGKDD Conf. Knowledge Discovery and Data Mining (KDD)*, 2024.

[48] P. Veličković, et al., “Graph attention networks,” *Internat. Conf. Learning Representations (ICLR)*, 2018.

[49] S. Bai, F. Zhang and P.H.S. Torr, “Hypergraph convolution and hypergraph attention,” *Pattern Recognition*, 2024.

[50] W. Samek, et al., ”Explaining deep neural networks and beyond: a review of methods and applications,” *Proceedings of the IEEE*, 2021.