DeepOHeat: Operator Learning-based Ultra-fast Thermal Simulation in 3D-IC Design

Ziyue Liu[1], Yixing Li[2], Jing Hu[3], Xinling Yu[1], Shinyu Shiau[3], Xin Ai[3], Zhiyu Zeng[2] and Zheng Zhang[1]
[1]University of California at Santa Barbara, Santa Barbara, CA. Email: {ziyueliu, xyu644, zzhang01}@ucsb.edu
[2] Cadence Design Systems, Austin, TX. Email: {yixingli, zzeng}@cadence.com
[3] Cadence Design Systems, San Jose, CA. Email: {jinghu, shinyu, nathanai}@cadence.com

2023 60th ACM/IEEE Design Automation Conference (DAC) | 979-8-3503-2348-1/23/$31.00 ©2023 IEEE | DOI: 10.1109/DAC56929.2023.10247998

***Abstract*—Thermal issue is a major concern in 3D integrated circuit (IC) design. Thermal optimization of 3D IC often requires massive expensive PDE simulations. Neural network-based thermal prediction models can perform real-time prediction for many unseen new designs. However, existing works either solve 2D temperature fields only or do not generalize well to new designs with unseen design configurations (e.g., heat sources and boundary conditions). In this paper, *for the first time*, we propose DeepOHeat, a physics-aware operator learning framework to predict the temperature field of a family of heat equations with multiple parametric or non-parametric design configurations. This framework learns a functional map from the function space of multiple key PDE configurations (e.g., boundary conditions, power maps, heat transfer coefficients) to the function space of the corresponding solution (i.e., temperature fields), enabling fast thermal analysis and optimization by changing key design configurations (rather than just some parameters). We test DeepOHeat on some industrial design cases and compare it against Celsius 3D from Cadence Design Systems. Our results show that, for the unseen testing cases, a well-trained DeepOHeat can produce accurate results with $1000\times$ to $300000\times$ speedup.**

***Index Terms*—3D IC, thermal simulation, operator learning, deep learning**

I. Introduction and Related Work

The increasing transistor density on a silicon chip has led to high power and heat density. The excessive heat can affect the normal performance, reliability, and lifespan of semiconductor chips. Due to the multiple stacked active silicon layers, 3D IC design suffers from much higher power density [1]–[3]. Meanwhile, the increased complexity of 3D chips introduces extra design configurations and system parameters and hence prolongs the design cycle. Consequently, chip thermal optimization, which provides the optimal thermal-aware floorplan at an early stage, has become an important step in the 3D IC design flow. Detailed and fast thermal simulators are needed in various thermal-aware design optimization tools.

Discretization-based PDE solvers, such as finite-element and finite-difference methods, have been widely used for 3D chip thermal analysis. The finite-element method (FEM), though computationally expensive, provides the best accuracy and flexibility [3], and is mostly used in commercial solvers such as Celsius, ANSYS, and COMSOL. The finite-difference methods (FDM) are simpler to implement and are widely used in open-source solvers [4]–[6]. These thermal simulators provide accurate temperature estimations but cost extensive computational resources. Once a new design is generated, designers need to re-run many simulations to optimize the design case, which can be unaffordable for complicated tasks. Some surrogate models have been developed to reduce the cost of thermal prediction. For instance, model-order reduction [7], [8] can accelerate each time-domain simulation via reducing the number of state variables in a dynamic system. Data-driven regression methods [9], [10] can model the dependence on certain design parameters in a specified range, but the training step often needs massive high-resolution PDE simulation data. Neither technique can capture the dependence of the temperature field on key PDE configurations (e.g., boundary conditions, non-parametric heat source configurations).

Neural network-based methods can perform real-time predictions for unseen data. Several data-driven [11]–[13] and physics-informed neural network-based (PINN) methods [14], [15] have been proposed. However, these existing works either fail to solve 3D full-chip temperature fields, lack generalization to different PDE configurations, or need to be combined with traditional solvers or additional computations. For example, the data-driven method in [11] needs to be combined with a coarse thermal profile obtained by a traditional FEM-based method. The ML-based transient thermal solver in [12] needs to be combined with convolution operations. The autoencoder-decoder-based methods in [13], [14] are not applicable to 3D volumetric power maps. The PINN-based approach in [15] only takes input from geometric parameters rather than general configurations, such as boundary conditions and power maps.

Paper Contributions. We propose the DeepOHeat framework, which leverages recent advances in operator learning, as an end-to-end thermal solver for ultra-fast 3D chip thermal prediction under various (both parametric and non-parametric) PDE configurations. Our contributions are as follows:

- *For the first time*, an end-to-end operator learning-based 3D IC thermal simulator is proposed to solve a family of heat equations under various PDE configurations.
- We propose a modular approach that encodes the PDE configurations of 3D IC designs, including arbitrarily stacked cuboidal geometry, individually defined boundary conditions, 2D/3D power maps, and full-chip flexible material conductivity distribution.
- The proposed DeepOHeat achieves $1000\times$ to $300000\times$ speed up with satisfactory accuracy when compared against Celsius 3D, a FEM-based commercial solver.

II. Background: Thermal simulation in 3D IC

Here we provide a brief overview of thermal simulation in the context of 3D IC design. Thermal simulation aims to predict the temperature field of a given object (chip) S by solving the heat conduction PDE globally. The 3D governing PDE is written as

$$\frac{\partial}{\partial y_1}\left(k\frac{\partial T}{\partial y_1}\right)+\frac{\partial}{\partial y_2}\left(k\frac{\partial T}{\partial y_2}\right)+\frac{\partial}{\partial y_3}\left(k\frac{\partial T}{\partial y_3}\right)+q_V=\rho c_p\frac{\partial T}{\partial t}, \tag{1}$$

where T and q_V represent the temperature and the rate of internally generated energy per unit volume at any spatial-temporal location $(y_1, y_2, y_3, t) \equiv (\boldsymbol{y}, t)$. Here k, ρ, c_p are material-specific properties of S denoting material conductivity, mass density, and heat capacity, respectively.

We focus on the static temperature field for isotropic materials (i.e., k_{y_1}=k_{y_2}=k_{y_3}=k), and simplify (1) by setting $\frac{dT}{dt}=0$:

$$k\cdot\nabla^2 T+q_V=0, \tag{2}$$

in which ∇^2 stands for the laplacian operator. We then solve (2), with appropriately defined boundary conditions in the context of 3D IC design, for various chip designs to find the optimal design by thresholding the temperature field.

III. Modular Chip Configurations for Thermal Analysis

Without loss of generality, we model the geometry of a chip as single or multiple stacked rectangular cuboid(s) as shown in Fig. 1. For each cuboid, its temperature field depends on some key design configurations which include, but are not limited to, material/geometric parameters.

The first family of design configurations is the boundary condition (BC) for each individual surface that is exposed to the environment. We consider the following types of BCs:

- **Dirichlet**: the temperature field on a surface is fixed as q_d:

$$T=q_d. \tag{3}$$

- **Neumann**: the temperature flux on a surface is fixed

$$-k\frac{\partial T}{\partial y_i}=q_n, \tag{4}$$

  where q_n represents the local heat flux density at the surface.
- **Adiabatic**: a special case of Neumann BC when q_n is 0 everywhere. This indicates a perfectly insulated surface.
- **Convection**: also known as Newton BC. This BC corresponds to a balance between heat conduction and convection in the same direction at the surface:

$$-k\cdot\frac{\partial T}{\partial y_i}=h(T-T_{\text{amb}}). \tag{5}$$

  Here h and T_{amb} stand for the heat transfer coefficient at the surface and the ambient temperature.

The second family of key design configurations are the locations and intensity of external/internal heat sources. This work considers the following two types of heat sources:

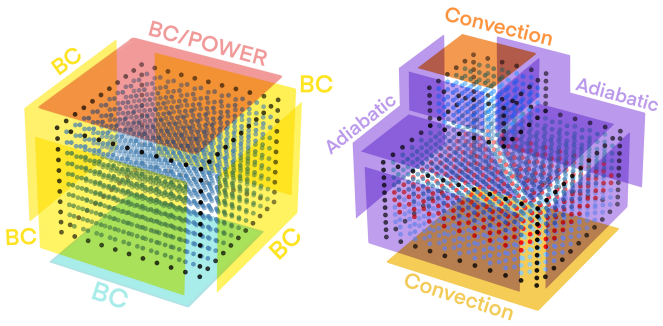

Fig. 1: Schematic figures of chip designs in thermal simulation. The left one shows a general single cuboid chip model, of which the right one is a concrete implementation.

- **Surface/2D power**: defined by the Neumann BC (4) when q_n is positive somewhere. Such q_n is referred to as a surface/2D power map.
- **Volumetric/3D power**: defined by the heat equation (2) when q_V is positive somewhere. Such q_V is referred to as a volumetric/3D power map.

We now present the thermal chip designs by several independent modular configurations as shown in Fig. 1. The left figure shows a general single cuboid chip with different BCs defined on each surface. The BC for the top surface also defines a 2D power map. The uniform blue color for the dots inside the cuboid indicates homogeneously distributed conductivity without any internal heat source. As a comparison, the right figure indicates a concrete implementation. In this model, we have volumetric power shown as the red dots in the middle layer of the bottom cuboid with adiabatic BCs on all side surfaces and convection BCs on the top and bottom surfaces. The different colors applied to the convection surfaces and the internal blue dots indicate different heat transfer coefficients and inhomogeneously distributed conductivity.

The above design configurations can change the PDE structure and temperature field of a 3D IC significantly. Many of them are described as functions instead of parameters, and they cannot be handled by traditional machine learning techniques.

IV. The DeepOHeat framework

Now we present DeepOHeat: a self-supervised operator learning-based neural thermal solver enabling ultra-fast thermal prediction. DeepOHeat takes functions that characterize key design configurations (e.g., power maps, boundary conditions, domain of interest) rather than just material or geometric parameters as inputs to predict temperature fields in real time. The key ideas of DeepOHeat are shown in Fig. 2.

A. Learning the Solution Dependence on Multiple PDE Configurations via a Multi-input DeepONet

For succinct notations, we denote the heat equation of interest (2) in the following general format

$$\mathcal{N}(\boldsymbol{s}(\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_k)(\boldsymbol{y}))=0. \tag{6}$$

Here $\mathcal{N}$ is a symbolic representation of the simplified heat equation (2). We denote the temperature field, i.e., the solution

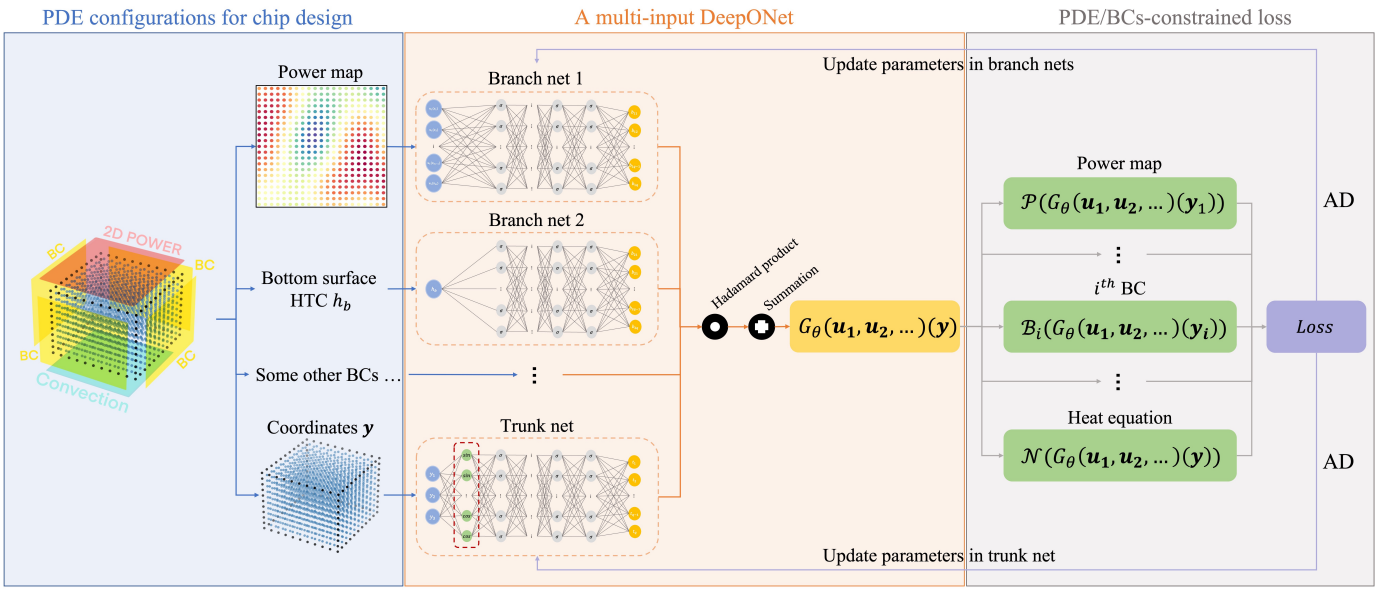

Fig. 2: The proposed DeepOHeat framework.

function of this PDE, as s. A concrete temperature field is determined by a certain chip design specified by various configurations such as a power map and BCs. We present in general k design configurations of interest (i.e., PDE configurations), both parametric and non-parametric, as $u_1, u_2, \ldots, u_k$. Given specific PDE configurations $\{u_i\}_{i=1}^k$, the temperature field on the domain of interest is then evaluated on the corresponding spatial coordinates y, yielding the final formal representation as $s(u_1, u_2, \ldots, u_k)(y)$.

To avoid any potential confusion, we emphasize that each u_i, $i = 1, 2, \ldots, k$, no matter which representation form it uses, is represented as a **function** instead of a parameter in DeepOHeat. Therefore, DeepOHeat is designed to learn a functional map G_θ (θ denote all the neural network parameters in DeepOHeat, i.e., weights and bias) that maps the function space spanned by the PDE configurations $\{u_i\}_{i=1}^k$, denoted by $\mathcal{U} : \mathcal{U}_1 \times \mathcal{U}_2 \times \cdots \times \mathcal{U}_k$, to the corresponding function space $\mathcal{S}$ spanned by its temperature field $s(u_1, u_2, \ldots, u_k)(y)$, i.e.,

$$G_\theta : \mathcal{U} \to \mathcal{S}. \tag{7}$$

Such a map means that, a well-trained DeepOHeat is capable of accurately predicting the temperature field given any unseen design drawn from the same PDE configurations space $\mathcal{U}$. To learn this functional map, we leverage recent works in operator learning, DeepONets [16] and multi-input DeepONets [17].

Encoding Design Configurations as Input Functions of DeepOHeat. We consider the general case that k PDE configurations are considered. Correspondingly, we will have k different input functions. For the i^{th} configuration, we consider a random sample $u_i^{(j)}$ drawn from its function space $\mathcal{U}_i$. This function (e.g., a 2D power map) is identified by its values on fixed locations $(x_1, x_2, \ldots, x_m)$ (e.g., some grid points of a surface), and is then fed as an m-dimensional vector into the i^{th} sub-network block, namely the i^{th} "branch net" [16]. Repeating this process for all k design configurations, we then have k different input functions and the corresponding branch nets. All these configurations are from a certain design thus share the same domain of interest. we then input all the coordinates sampled from this simulation domain into another sub-network, namely the "trunk net". To effectively learn the high-frequency information of the temperature field, we also apply a Fourier features mapping [18] to the first layer of the trunk net, which is shown inside the dashed red box in the trunk net part of Fig. 2.

Example. We consider the example shown in the left part of Fig. 2. We see that for this single-cuboid chip, we define a 2D power map on the top surface. The power map that can have an arbitrary layout of heat sources, is with no doubt a non-parametric function. We identify this 2D power map by its values on equispaced grid points, which naturally form a two-dimensional matrix as shown in Fig. 2. We then flatten this matrix to a vector and feed it into the first branch net. If we consider a 3D power map, everything will be exactly the same except it will be identified by its values on three-dimensional equispaced grid points. Meanwhile, we define a convection BC on the bottom surface of the chip with a uniform HTC distribution of value h_b. In this case, the HTC on the bottom surface can be seen as a constant function therefore only one grid point is needed to identify this configuration. We then input h_b into the second branch net. Note that h_b should still be regarded as a function that has a parametric format instead of a parameter. If the surface has an inhomogeneous HTC distribution, one can simply encode it similarly as we encode a 2D power map. For the side surfaces of the chip, other BCs are defined accordingly and encoded as other DeepOHeat inputs or just fixed invariant configurations.

With k defined PDE configuration inputs and the domain coordinates, we have in total k branch nets and one trunk net, each of which outputs a q-dimensional feature vector. We then follow the ideas in [17] to combine all these output features via Hadamard (element-wise) product and then sum up the

resulting vector to a scalar output that represents the predicted temperature field, denoted as $T = G_{\boldsymbol{\theta}}(\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_k)(\boldsymbol{y})$.

B. Training DeepOHeat via Physics-Informed Loss

Now we explain how to train the DeepOHeat network. According to [16], a DeepONet is generally trained via a data-driven approach, in which data triplets $(\boldsymbol{y}, \{\boldsymbol{u}_i\}_{i=1}^k, \boldsymbol{s})$ need to be collected via massive runs of numerical simulation. For relatively complicated chip designs, a single FEM simulation might cost hours or even days to complete. Therefore, large-scale data collection is practically prohibitive in this context. Instead, we follow the idea from a recent approach [19], which leveraged the ideas from physics-informed neural networks (PINNs) [20] to train a single-input DeepONet for solving parametric PDEs. We extend their work to handle multi-input scenarios as shown on the right of Fig. 2.

Again we consider the aforementioned general case where k chip design configurations are considered. For the i^{th} configuration $\boldsymbol{u}_i$, we first index all the coordinates that are located in its designated regions, such as a boundary surface, denoted as $\boldsymbol{y}_i$. Then on $\boldsymbol{y}_i$, we impose a physics constraint $\mathcal{L}_i$. If $\boldsymbol{u}_i$ represents a power map, we denote $\mathcal{L}_i$ as

$$\mathcal{L}_i = \left\| \mathcal{P}\left(G_{\boldsymbol{\theta}}(\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_k)(\boldsymbol{y}_i)\right) \right\|. \tag{8}$$

For a 2D power map, $\mathcal{P}$ is a symbolic representation of the Neumann BC (4). For a 3D power map, $\mathcal{P}$ will represent the heat equation (2) with non-zero q_V. If $\boldsymbol{u}_i$ represents a general BC, such as convection or Dirichlet BC, we denote $\mathcal{L}_i$ as

$$\mathcal{L}_i = \left\| \mathcal{B}_i\left(G_{\boldsymbol{\theta}}(\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_k)(\boldsymbol{y}_i)\right) \right\|, \tag{9}$$

where $\mathcal{B}_i$ denotes the formulation of the corresponding BC. For the entire domain of interest, we impose the PDE constraint, except for the region where a 3D power map is imposed, as

$$\mathcal{L}_r = \left\| \mathcal{N}\left(G_{\boldsymbol{\theta}}(\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_k)(\boldsymbol{y})\right) \right\|. \tag{10}$$

We then obtain the total loss as

$$\mathcal{L}_{\text{total}} = \mathcal{L}_r + \sum_{i=1}^{k} \mathcal{L}_i. \tag{11}$$

We train DeepOHeat by minimizing the total loss via gradient descent based on automatic differentiation algorithms [21].

V. Experiments

In this section, we present two implementations of the proposed DeepOHeat and compare our results with Celsius 3D, a state-of-the-art numerical solver for 3D chip thermal analysis from Cadence Design Systems. Our results demonstrate that, for any unseen designs, a well-trained DeepOHeat is capable of producing satisfactory results with at least 1000$\times$ speedup.

A. 2D Power Map Configuration on The Top Surface

As the power map controls the heat generation in a certain chip design, the prediction performance of DeepOHeat on unseen new power maps are of major interest. For illustration, here we focus solely on optimizing a 2D power map by training a single-input DeepOHeat.

1) Problem setup: We consider a $21 \times 21 \times 11$ mesh grid-based single-cuboid geometry which represents a 1mm $\times$ 1mm $\times$ 0.5mm chip in practice. This geometry is similar to the one shown in the left of Fig. 1 and has in total of 4851 grid points. We define a 2D power map on the top surface, in which a one-unit power corresponds to a $0.00625(mW)$ power in real-world settings. We define Adiabatic BC on all side surfaces and convection BC on the bottom surface with HTC $= 500W/(m^2K)$ and $T_{\text{amb}} = 298.15(K)$. A homogeneous thermal conductivity $k = 0.1W/(mK)$ is assigned to the entire domain and no volumetric power is applied.

2) Generating training power maps: We sample all the training power maps from a two-dimensional standard Gaussian random field (GRF) with the length scale parameter equal to 0.3. The length scale controls the smoothness of the sampled functions. We choose 0.3 in this example to generate relatively smooth power maps as shown on the left of Fig. 4. One can also tune this parameter to generate training power maps similar to those in specific optimization tasks. Corresponding to our 21×21 mesh grids on the top surface, we identify each power map by its values on these coordinates formatted as a matrix of the same size. We then flatten these matrices to vectors of length 441 as the input of the branch net.

3) DeepOHeat settings: In this example, we use a 9-layer branch net with 256 neurons per layer combined with a 6-layer trunk net with 128 neurons per layer. The first layer of the trunk net is a Fourier features mapping [18] where its coefficients are sampled from a normal distribution with zero mean and 2π standard deviation. The input dimensions of the branch net and the trunk net are 441 and 3, which correspond to the dimensions of the encoded power map and the 3D spatial coordinates, respectively. The output dimensions of the two sub-networks are both 128. We set all the activation functions as the "Swish" function proposed by Ramachandra et al. [22]. We find in experiments that Swish yields relatively better results compared to other popular activation functions used in PINNs, such as Sine and Tanh.

4) Training settings: We train this DeepOHeat by 10000 iterations to guarantee convergence, which takes 10 hours on a single Tesla V100 GPU. In each iteration, 50 input functions are sampled from the given GRF and fed into the branch net. For each function, the 4851 mesh grid points of the entire simulation domain are fed into the trunk net. We therefore have a 242550×441 input for the branch net and a 242550×3 input for the trunk net. We choose the initial learning rate as 1e-3 and decay the learning rate by $0.9\times$ every 500 iterations.

5) Test settings: We aim to compare the predicted temperature fields with Celsius 3D element-wisely on unseen new power maps. There exists a minor discrepancy between the power maps of Celsius 3D and DeepOHeat. As shown in the middle of Fig. 4, the power maps in Celsius 3D are tile-based, different from the grid-based ones in DeepOHeat. To accommodate these realistic power maps used in Celsius 3D, we interpolate the 20×20 tile-based power maps to 21×21 grid-based power maps, as shown in the middle and right of Fig. 4. Such a transformation not only enables DeepOHeat

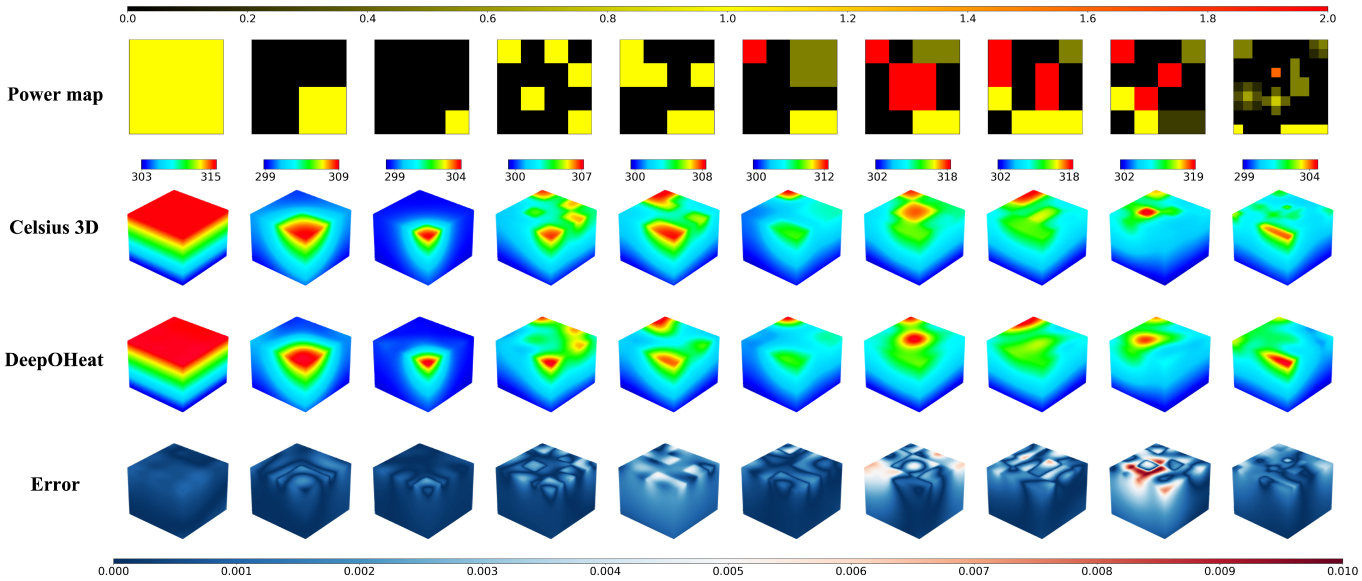

Fig. 3: Predicted temperature fields for different 2D power maps defined on the top surface.

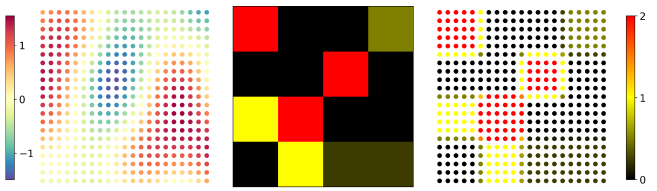

Fig. 4: Left: a power map for training; Middle and right: test power maps for Celsius 3D and for DeepOHeat, respectively.

TABLE I: Mean and peak errors for all power maps

	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}
MAPE (%)	0.03	0.03	0.02	0.05	0.14	0.04	0.13	0.07	0.16	0.08
PAPE (%)	0.10	0.20	0.24	0.38	0.52	0.49	0.71	0.66	1.00	0.40

to accept almost the same realistic power maps as in Celsius 3D but also smooths out these discretely defined power maps. As the ones we used for training are all continuous functions, using the smoothed rather than discrete power maps for testing, in a heuristics sense, would have lower generalization errors.

6) Results: All the results of this example are shown in Fig. 3. From the left to the right, we gradually increase the complexity of the unseen test power maps. We simply refer them to as $p_1, p_2, \ldots, p_{10}$ and report their mean absolute percentage errors (MAPEs) and peak absolute percentage errors (PAPEs) in Table. I. We see that DeepOHeat is capable of predicting the temperature fields for all these unseen test power maps with satisfactory accuracy. Moreover, we want to highlight the strong generalization power of DeepOHeat. All these power maps, though most of which are composed of heat blocks, are quite different from the training power maps. Specifically, the last power map p_{10} can be seen as a very wiggly function in this context. We see that p_{10} has multiple small-sized heat sources and one of them is also given a relatively large power. For such a complicated power map, DeepOHeat still yields satisfactory predictions at the most part of the domain, except for mildly overestimated temperatures

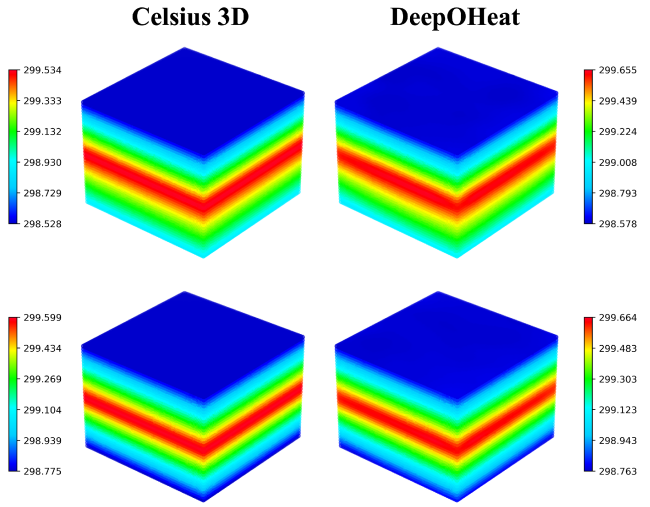

Fig. 5: Temperature fields under different HTC configurations.

at the regions between those small-sized heat sources.

7) Speedup: In this example, Celsius 3D costs approximately 5min for a single simulation on an Intel Xeon Gold 6148 CPU. The post-training prediction time for DeepOHeat is 0.1s on the same CPU and 0.001s on a Tesla V100 GPU, which correspond to a 3000× and 300000× speedup, respectively. For a larger-scale or more complicated design, the computational cost for FEM-based solvers will rapidly increase while remaining unchanged for DeepOHeat. We expect more significant speed-up in realistic thermal optimization tasks.

B. HTC Configurations on Both Top and Bottom Surfaces

DeepOHeat can predict the thermal behaviors influenced by multiple design configurations. To demonstrate this, we build a dual-input DeepOHeat to predict the temperature field of a 3D IC influenced by the HTCs on two surfaces simultaneously. In this example, we avoid introducing detailed settings instead focus on those that are different from the previous example.

We consider a similar cuboid chip geometry with the size of 1mm × 1mm × 0.55mm but all 7000 points are randomly

sampled inside (on) the entire domain. We don't use mesh because we don't have mesh-based encoding for this example. We define convection BCs for both top and bottom surfaces and assume HTCs are constantly distributed. We define a single-layer uniform volumetric power with a thickness of 0.05mm and the value of $0.000625(W)$. The settings for side surfaces and thermal conductivity are the same as before.

In each iteration, we sample 20 i.i.d samples uniformly from a squared area $[333.33, 1000] \times [333.33, 1000]$ (W/m^2K), corresponding to 20 different HTCs for both two surfaces. For each sampled HTC tuple, we randomly draw a new set of coordinates from the simulation domain. Combining these, we have two 140000×1 inputs for the two branch nets and a 140000×3 input for the trunk net.

In this example, we use relatively simpler networks for the two branch nets, each of which contains 5 fully-connected layers with only 20 neurons per layer. The trunk net still has 6 layers with 128 neurons per layer and a Fourier features mapping defined in the first layer with a π standard deviation this time. The output dimensions for all sub-networks are 50.

After training DeepOHeat for 5000 iterations (about 2 hours), we evaluate its performance on some unseen values sampled from the same 2D region. For example, we pick two sets of HTCs, $(1000, 333.33)$ and $(500, 500)$, as the test cases and show the corresponding results in each row of Fig. 5. Although different HTCs make only slight differences, DeepOHeat still yields accurate predictions in both cases. As shown by the color bars in Fig. 5, the differences in the predicted maximal and minimal temperatures between Celsius 3D and DeepOHeat are within $0.1(K)$. In the first case where HTC $= 1000$ on the top surface and HTC $= 333.33$ on the bottom surface (first row in Fig. 5), the MAPE and PAPE of DeepOHeat are 0.032% and 0.043%. In the second case where HTC $= 500$ on both two surfaces (second row in Fig. 5), the MAPE and PAPE of DeepOHeat are 0.011% and 0.025%.

Celsius 3D costs around 2min for a single simulation on the aforementioned CPU. The runtime for DeepOHeat remains unchanged. Therefore the speed up in this example is $1200\times$ and $120000\times$ on CPU and GPU, respectively.

VI. Conclusion

In this work, *for the first time*, we have introduced a physics-aware operator learning framework, named DeepOHeat, to perform ultra-fast 3D chip thermal prediction under multiple chip design configurations. We have proposed a modular chip thermal model to encode various chip geometries, power maps, and boundary conditions. We have applied a physics-informed multi-input DeepONet to seamlessly solve a family of heat equations that take multiple BCs and the power map as input configurations with no data supervision required. The experiments on two specific tasks show that a well-trained DeepOHeat can predict the temperature fields on unseen new chip designs with high accuracy while no noticeable simulation time is required. In the future, we will further investigate how DeepOHeat performs in more complicated geometries and in optimizing 3D power maps.

References

[1] H. Delaram, A. Dastfan, and M. Norouzi, "Optimal thermal placement and loss estimation for power electronic modules," *IEEE Trans. Comp., Packag. and Manufacturing Tech.*, vol. 8, no. 2, pp. 236–243, 2018.

[2] K. Cao, J. Zhou, T. Wei, M. Chen, S. Hu, and K. Li, "A survey of optimization techniques for thermal-aware 3D processors," *Journal of Systems Architecture*, vol. 97, pp. 397–415, 2019.

[3] H. Sultan, A. Chauhan, and S. R. Sarangi, "A survey of chip-level thermal simulators," *ACM Comput. Surv.*, vol. 52, no. 2, pp. 1–35, 2019.

[4] A. Sridhar, A. Vincenzi, M. Ruggiero, T. Brunschwiler, and D. Atienza, "3D-ICE: Fast compact transient thermal modeling for 3D ICs with inter-tier liquid cooling," in *Proc. ICCAD*, 2010, pp. 463–470.

[5] P. Li, L. T. Pileggi, M. Asheghi, and R. Chandra, "Efficient full-chip thermal modeling and analysis," in *ICCAD*, 2004, pp. 319–326.

[6] ——, "IC thermal simulation and modeling via efficient multigrid-based approaches," *IEEE Trans. CAD Integr. Circuits Syst.*, vol. 25, no. 9, pp. 1763–1776, 2006.

[7] T.-Y. Wang and C. C.-P. Chen, "SPICE-compatible thermal simulation with lumped circuit modeling for thermal reliability analysis based on modeling order reduction," in *International Symposium on Signals, Circuits and Systems*, 2004, pp. 357–362.

[8] J. Xie and M. Swaminathan, "System-level thermal modeling using nonconformal domain decomposition and model-order reduction," *IEEE Trans. CPMT*, vol. 4, no. 1, pp. 66–76, 2013.

[9] S. K. Samal, S. Panth, K. Samadi, M. Saeidi, Y. Du, and S. K. Lim, "Adaptive regression-based thermal modeling and optimization for monolithic 3-D ICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 10, pp. 1707–1720, 2016.

[10] S. K. Samal, S. Panth, K. Samadi, M. Saedi, Y. Du, and S. K. Lim, "Fast and accurate thermal modeling and optimization for monolithic 3D ICs," in *Proc. Design Automation Conference*, 2014, pp. 1–6.

[11] J. Wen, S. Pan, N. Chang, W.-T. Chuang, W. Xia, D. Zhu, A. Kumar, E.-C. Yang, K. Srinivasan, and Y.-S. Li, "Dnn-based fast static on-chip thermal solver," in *Semiconductor Thermal Measurement, Modeling & Management Symposium*, 2020, pp. 65–75.

[12] A. Kumar, N. Chang, D. Geb, H. He, S. Pan, J. Wen, S. Asgari, M. Abarham, and C. Ortiz, "Ml-based fast on-chip transient thermal simulation for heterogeneous 2.5 d/3D IC designs," in *International Symposium on VLSI Design, Automation and Test*, 2022, pp. 1–8.

[13] R. Ranade, H. He, J. Pathak, N. Chang, A. Kumar, and J. Wen, "A thermal machine learning solver for chip simulation," in *ACM/IEEE Workshop on Machine Learning for CAD*, 2022, pp. 111–117.

[14] H. He and J. Pathak, "An unsupervised learning approach to solving heat equations on chip based on auto encoder and image gradient," *arXiv preprint arXiv:2007.09684*, 2020.

[15] O. Hennigh, S. Narasimhan, M. A. Nabian, A. Subramaniam, K. Tangsali, Z. Fang, M. Rietmann, W. Byeon, and S. Choudhry, "Nvidia simnet™: An ai-accelerated multi-physics simulation framework," in *Int. Conf. Computational Science*. Springer, 2021, pp. 447–461.

[16] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis, "Learning nonlinear operators via deeponet based on the universal approximation theorem of operators," *Nature Machine Intelligence*, vol. 3, no. 3, pp. 218–229, 2021.

[17] P. Jin, S. Meng, and L. Lu, "Mionet: Learning multiple-input operators via tensor product," *arXiv preprint arXiv:2202.06137*, 2022.

[18] M. Tancik, P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. Barron, and R. Ng, "Fourier features let networks learn high frequency functions in low dimensional domains," *Advances in Neural Information Processing Systems*, vol. 33, pp. 7537–7547, 2020.

[19] S. Wang, H. Wang, and P. Perdikaris, "Learning the solution operator of parametric partial differential equations with physics-informed deeponets," *Science advances*, vol. 7, no. 40, p. eabi8605.

[20] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational physics*, vol. 378, pp. 686–707, 2019.

[21] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: a survey," *Journal of Marchine Learning Research*, vol. 18, pp. 1–43, 2018.

[22] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *arXiv preprint arXiv:1710.05941*, 2017.