Variability-Aware Thermal Simulation using CNNs

Hameedah Sultan
School of Information Technology
Indian Institute of Technology, New Delhi, India
Email: hameedah@cse.iitd.ac.in

Smruti R. Sarangi
Computer Science and Engineering
Indian Institute of Technology, New Delhi, India
Email: srsarangi@cse.iitd.ac.in

***Abstract*—With rising power densities in modern-day electronic systems, temperature has emerged as a fundamental design constraint. This has led to the advent of a range of thermal-aware design and runtime management techniques. However, such techniques are heavily dependent on a fast and accurate thermal modeling method. These methods need to account for manufacturing variability, that significantly impacts the chip's power and performance. Similarly, leakage power too contributes to a substantial portion of the total power. Thus a thermal modeling method can be accurate only if it is capable of incorporating the effects of process variation as well as leakage power.**

In this paper, we propose a simple and elegant residual convolutional neural network for thermal estimation in the presence of variability, which leverages the physics of heat transfer. Our approach is capable of modeling modern-day 3D chips with microchannels and incorporates accurate leakage power models. To enable ultra-fast thermal estimation, we implement our technique on a GPU. Our experiments show that our technique is orders of magnitude faster than the state-of-the-art with a similar, if not better, accuracy. The mean absolute error using our technique is 0.61°C, for a maximum temperature rise of 67.5°C (0.9%).

I. Introduction

In the last two decades, Moore's law-based scaling of electronic devices has led to a continuous increase in power densities in modern-day chips. This increase in power densities has led to temperature becoming a fundamental constraint in electronic design automation. Thus, thermal modeling has gained significant attention in the past few years. However, several challenges in thermal modeling have not been adequately addressed yet. Some of these challenges are:

1) With the advent of 3D ICs, the thermal problem has become much more severe. In a 3D IC, multiple layers are stacked on each other, resulting in poor heat transfer paths. To alleviate the hotspots in 3D ICs, microchannels have emerged as a promising solution [1]. However, they pose new thermal modeling challenges.

2) Leakage power can contribute to 30-50% of the total power in modern-day chips [2]. It has an exponential dependence on temperature. An increase in temperature increases the leakage power, which further increases the temperature. This results in a feedback loop. For any thermal modeling tool to have acceptable accuracy, it must capture the thermal effects of leakage power. Most existing techniques consider the effects of leakage power iteratively, but this multiplies the runtime by 3-10X.

3) With the continued miniaturization of chips, process variation has emerged as another significant design challenge. A chip designed by ignoring variation may actually fail after fabrication [3] or fail to meet the design specifications. There are several design techniques aimed at mitigating the effects of variation, such as frequency binning and body biasing [4]. However, to design such techniques, a fast and accurate thermal simulator is needed that can model the effects of variation in 2D as well as 3D chips.

This work has been partially sponsored by the Semiconductor Research Corporation (SRC). The first author is supported by the Visvesvaraya PhD Scheme, MeitY, Govt. of India, with awardee number MEITY-PHD-701.

Existing thermal modeling techniques fail to factor in the effects of these crucial contributors to temperature. This limits the accuracy of the models, which in turn affects their usability. Most existing thermal simulators are based on the finite-difference method. However, these methods are slow. Green's function-based methods are faster than these numerical approaches ([5], [6]). Although these methods can model leakage power, they use a linear model for the temperature dependence of leakage power, limiting their accuracy. Such methods do not model process variation or more complex geometries, such as chips with microchannels and TSVs. Hence, there is a further need for better thermal modeling methods.

To ameliorate such problems, learning-based techniques have been shown to have great potential in thermal estimation [7], [8], [9]. Most state-of-the-art techniques either use sensor measurements or data from performance counters to predict future temperatures. However, such empirical thermal data is often either unavailable, or available for a very limited region, or prone to processing delays and noise. Sadly, such learning-based techniques ignore the effects of leakage power and process variation. Moreover, such methods do not generalize well.

We build on convolutional neural networks (CNNs), which have played a prominent role in the advances in computer vision. CNNs are well-suited to image data since they capture the complex pixel dependencies very effectively with very few parameters. The power applied to any floorplan element affects its neighbors the most, while the elements far away are barely affected. The convolution operation inherently captures such dependencies. Thus, thermal data shares similar characteristics since the power and thermal matrices can be thought of as images. A convolution of the impulse response of the system with the power profile gives the temperature profile. Thus convolution is an inherent property of the system. Hence, we propose a novel convolutional neural network (CNN)-based architecture to estimate the leakage-aware temperature profile. Our approach is generic and can be used to model any type of chip. To demonstrate the effectiveness of our approach, we choose the most challenging layout: a 3D chip with microchannels that is affected by *process variation*. We also consider the effects of leakage power in our modeling. Due to the nature of parallel computations in a neural network, significant gains can be obtained by implementing such techniques on a GPU. Thus we implement our technique on a GPU along with tensor cores and demonstrate a significant speedup. We have validated our work using the open-source thermal modeling tool, 3D-ICE [1]. Our main contributions can be summarized as follows:

❶ We propose a novel CNN-based machine learning architecture for thermal estimation that leverages heat transfer physics to obtain accurate and efficient results. Our method generalizes well to a variety of floorplans with different die sizes, meaning that a trained CNN can be reused even when the floorplan changes.

❷ We use accurate models for leakage power as well as process

variation to obtain the final leakage aware full-chip temperature. To estimate this final temperature, our method relies on the power dissipation profile only.

❸ Our method is simple and lightweight, with very low complexity. The inferencing is done by simple convolution operations, which have very efficient implementations. The inferencing time is $0.1\ ms$ for a high resolution $64 \times 64 \times 4$ architecture.

❹ Due to the massively parallel nature of computations in a CNN, a significant speedup is obtained by implementing our technique on a GPU with tensor cores. We were able to reduce the training time from over 2 days to 15 minutes ($192\times$ speedup).

II. Related Work

1) Thermal Modeling in 2D and 3D ICs: Thermal modeling in 2D and 3D ICs has been widely studied [1], [10] and surveyed [11]. Most thermal modeling techniques are based on the finite difference or finite element methods. Such techniques are robust and can model a variety of layouts. However, these methods are slow. Moreover, most such techniques fail to account for important thermal effects, such as process variation and leakage power. Feng and Li [12] implement existing finite difference based techniques on a GPU using a two-step iterative method. Liu et al. [13] use a GPU accelerated GMRES solver. However, even after GPU acceleration, these techniques are still slow. If leakage power is modeled, the runtime will further deteriorate. Jaffari and Anis [3] studied the statistical temperature distribution in a die in the presence of variation. However, their technique is iterative, thereby slow and prone to convergence issues.

2) Machine Learning-based Thermal Modeling: Juan et al. [9] use an autoregressive framework to model the maximum temperature in a layer in the presence of variation as a function of the leakage power values. Their model is very simplistic and does not look at the temperature distribution. Siddhu and Panda [14] predict the temperature in 3D memories in the presence of leakage power using a linear regression-based model. They utilize the symmetry in the chip and a few sanity checks to greatly reduce the number of model parameters. They do not model process variation. Sadly, linear regression-based modeling requires a large number of parameters. Even after the $362\times$ parameter reduction, the number of parameters remains large for a relatively simple system. Additionally, our CNN based approach inherently takes care of the symmetries as well as the neighborhood dependencies since it relies on convolution operations.

Sridhar et al. [7] use a single-layer neural network (NN) for a 3D IC with microchannels using the current power and temperature as input to predict the temperature $500\ ms$ into the future. They remove the connections between neurons that are farther than a threshold to reduce computations. They implement the online part of their technique on a GPU. However, it is unclear how the initial temperatures are obtained at runtime and whether any error in the initial temperature can cause a cascade effect in the entire estimated temperature sequence. Furthermore, these works do not model leakage power or process variation. We show in our work that this method is slower than our approach and results in a much larger number of parameters; it also has a lower accuracy.

Abad et al. [15] use feature selection over a large set of performance counters to determine the best features to predict future temperatures. Zhang et al. [8] use a set of features to predict future temperatures in HPC environments. They classify these features into application features and physical features. The average error is under 3.7°C. Sadiqbatcha et al. [16] provide an LSTM (long short-term memory) based learning algorithm to estimate the temperature thermal data collected from IR cameras. However, all these techniques rely on the current temperature, which cannot be accurately and quickly obtained at runtime. There is often a significant time lag in getting these readings. Additionally, such measurements are prone to noise and need data preparation steps to be usable in actual runtime environments.

Performance-counter based methods cannot be used at design time. Additionally, these methods are very strong functions of micro-architectural features. In fact, other than the work by Sridhar et al. [7], none of the other works presented here have demonstrated the generalizability of their method.

III. Green's functions and CNNs

A. Convolutional Neural Network (CNN)

CNNs are a widely used class of neural networks that use layers to progressively learn patterns in the data. A CNN is best suited for image analysis since it can learn the spatial and temporal dependencies in the data using a set of filters that respond to a *receptive field*. Since the power and temperature maps in ICs are also images, a CNN based approach is well suited to capture the dependencies in the power matrix. **It can easily capture the neighborhood dependencies with a very few parameters.** This is especially important when modeling process variation since process variation exhibits strong spatial correlation. A CNN consists of the following basic layers:

1) Convolutional layer: The convolutional layer uses a kernel, which is slid over the input matrix, and the convolution of the kernel with the input image is computed. The resulting output is another matrix. The kernel is learned in the training phase. The resulting equation is as follows:

$$output = w \star input + b, \tag{1}$$

where $input$ and $output$ are the inputs and outputs of the convolutional layer, w and b are the weights and bias values respectively, learned in the training phase, and $\star$ is the convolution operator.

2) ReLu layer: In this layer, an element-wise activation function is applied to the input. The ReLu activation function computes $max(0, x)$ for each pixel x in the image.

3) Pooling layer: In this layer, a downsampling operation is done to reduce the dimensionality of the image.

4) Fully connected layer: Here, each neuron is connected to every neuron in the previous layer. The output from the previous layer is flattened and given to the fully connected layer, which computes its matrix multiplication with a weight matrix. The output is then passed through *softmax* activation to get the final output labels.

Although, a CNN is generally used for classification tasks, there is an entire body of work where it has been used for regression [17].

B. Green's Function-based Thermal Estimation

Green's function-based methods rely on obtaining the impulse response (also called the Green's function) for a unit power source. It can be calculated empirically, measured, or simulated. Next, the temperature profile at runtime is obtained by convolving the Green's function with the dynamic power profile:

$$\mathcal{T} = G \star P \tag{2}$$

where $\mathcal{T}$ is the temperature profile, G is the Green's function, and P is the power profile. Sarangi et al. [18] have shown that if the Green's function is modified to be leakage-aware, the leakage-aware temperature profile may be obtained by the same equation.

The convolutional layer in a CNN performs a similar operation. Comparing Equations 1 and 2, if the $input$ is the power profile, w is the leakage-aware Green's function, and $b = 0$, then the $output$

will be the leakage-aware thermal profile. Thus using a CNN, it is possible to *learn the Green's function*, instead of deriving or empirically obtaining it. The Green's function learned using CNNs is an adaptive kernel that adapts itself on the basis of the properties of the chip. This kernel can be progressively learned such that the fine irregularities in the thermal profile because of variation is modeled. This is particularly helpful in situations where it is prohibitively difficult to derive the leakage-aware Green's function, such as in ICs affected by process variation or having complex layouts.

The unique advantage of using this approach is that since the impulse response is independent of both the floorplan and the input power applied, **any floorplan and power dissipation profile can be used to train the CNN. The learned network can then be used to predict the thermal response corresponding to any other floorplan or power profile.** No other machine learning-based technique offers this advantage.

C. Motivation for a CNN based approach

Although Green's function approach can fairly quickly give the thermal profile for a 3D IC, there are several bottlenecks that reduce its applicability. Some of these are:

❶ The Green's function-based method cannot be trivially extended to complex chips, such as chips with microchannels.
❷ These methods cannot be used with a higher-order leakage model since the complexity makes the problem intractable. So the existing Green's function approaches assume a linear temperature dependence of leakage power, which may underestimate the temperature.
❸ A separate correction needs to be applied to such chips to account for the edge and corner effects, which increases the runtime.
❹ When process variation is considered, the temperature profile no longer remains symmetric. In such circumstances, a Green's function-based approach cannot be directly used or results in very large errors.
❺ If the chip's properties change, the CNN may adapt, depending on the variety of inputs given in the training process. For a regular Green's function-based approach, the Green's function will have to be obtained again, and its leakage aware version recomputed.
❻ Finally, the runtime implementation is much faster in a CNN based approach, especially with GPUs.

IV. A Learning based Thermal Estimation Framework Considering Variability

A. Overview

We first provide an overview of our approach.

❶ We model the power consumption in the presence of process variation using its statistical properties.
❷ We propose a novel CNN based architecture that is based on the physics of heat transfer.
❸ The inputs to step ❷ are the dynamic power dissipation profiles and the baseline leakage power profile, while the corresponding leakage-aware temperature maps are needed for training. The baseline leakage power maps considering process variation are obtained in step ❶. The leakage-aware thermal profile is obtained from 3D-ICE [1] iteratively, assuming a quadratic temperature-dependent leakage model. For greater accuracy, we could obtain the power and thermal training data using real hardware or open-source toolchains like HotSniper [19] (for chips that can be modeled in HotSpot [10]).
❹ We apply several machine learning optimizations to improve the accuracy and convergence of the proposed model. To speed up the training process, we implement our technique on a GPU.

We describe each of these in greater detail next.

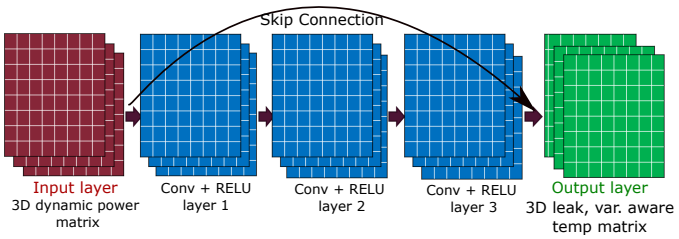

Fig. 1: 3D CNN Architecture

B. Modeling Process Variation

First, we need the baseline leakage power maps at the ambient temperature in the presence of variation. Process variation has two main components: inter-die (ΔX_{inter}) and intra-die component. The intra-die component has two sub-components: systematic variation (ΔX_{sys}), which is spatially correlated and occurs because of lithographic aberrations, and a random component (ΔX_{rand}) that occurs because of random dopant fluctuations and some other uncorrelated effects. The two prominent parameters affected by variability are the gate length and oxide thickness. Systematic variation is modeled by a multivariate Gaussian distribution, while random variation is modeled by a zero-mean Gaussian random variable. The chip is divided into tiny grids, and the spatial correlation is described by the covariance matrix of each parameter, assuming the properties remain the same within a grid. The random variation is described by its variance. The total variation considering all the components is given by Equation [3]:

$$\begin{aligned}\psi_X(i,i) &= \sigma^2_{\Delta X_{\text{inter}}} + \sigma^2_{\Delta X_{\text{sys}}} + \sigma^2_{\Delta X_{\text{rand}}} \\ \psi_X(i,j) &= cov(\Delta X_i, \Delta X_j) = \sigma^2_{\Delta X_{\text{inter}}} + \rho(d_{ij})\sigma^2_{\Delta X_{\text{sys}}}\end{aligned} \tag{3}$$

Here, ψ_X is the covariance matrix of parameter X (gate length or oxide thickness), i, j are grid points, d_{ij} is the distance between grid points i and j, ρ is the correlation function, and σ^2 represents the variance. The parameters here are obtained from ITRS data or empirical measurements in the literature. These equations give us the baseline leakage power at ambient temperature considering process variation, $P^{var}_{leak_0}$.

Next, we need to obtain the temperature-dependent leakage power maps in the presence of process variation. These are obtained by fitting a quadratic leakage model into the BSIM equation:

$$I_{leak} \propto v_T^2 * e^{\frac{V_{GS}-V_{th}-V_{off}}{\eta * v_T}}(1 - e^{\frac{-V_{DS}}{v_T}}) \tag{4}$$

where, v_T is the thermal voltage (kT/q), V_{th} is the threshold voltage, V_{off} is the offset voltage in the sub-threshold region, and η is a constant. Process variation results in a change in the threshold voltage via changes in the oxide thickness and gate length. Although I_{leak} has an exponential dependence on temperature, in the operating range of real ICs (40-80°C), a quadratic approximation results in less than a 0.5% error [11]. Thus, we arrive at the following equation:

$$\begin{aligned}I_{leak} &\propto (1 + \beta_1 \Delta T + \beta_2 \Delta T^2) e^{\beta_L \Delta L + \beta_{t_{ox}} \Delta t_{ox}} \\ P_{leak} &= (1 + \beta_1 \Delta T + \beta_2 \Delta T^2) P^{var}_{leak_0}\end{aligned} \tag{5}$$

where β represents the change of leakage power with temperature, $\beta_{t_{ox}}$ is a constant representing the variability in oxide thickness t_{ox} and β_L represents variability in gate length L, and $P^{var}_{leak_0}$ is the leakage power at ambient temperature considering variability. This gives us an expression for the temperature-dependent leakage.

C. Architecture of the CNN: CNNbasic

We first propose a basic CNN architecture, which assumes that each silicon layer in a stacked 3D-IC is independent of all other

layers. Our proposed CNN has five layers. Here, the input is a 2D power dissipation matrix. In the case of a 3D-IC, the power dissipation of each silicon layer is considered as a new sample. This input matrix is convolved with a weight matrix. The size of the weights is kept the same as the input size. To keep the output of the convolutional layer the same size as the input, zero padding is done around the edges of the input. A small bias is added to the input to account for non-idealities in the thermal profile and the random effects of process variation. Since machine learning approaches are purely mathematical in nature, we obtain negative outputs that are not practical. Thus we apply the ReLU activation to the output, where the negative output values are replaced by zero. We use two more identical convolutional layers. These layers help in progressively learning finer features of the thermal map, especially in the presence of process variation. These layers are optional and may be omitted with a minor loss of accuracy. The final output is obtained after this.

Although a traditional CNN has several more layers, such as max pooling and fully connected layers, we do not include these for two reasons: 1) we want the complexity of the architecture to be as low as possible. 2) Our goal here is to use a CNN for predicting the temperature values, which is essentially a regression task. We are not using the CNN for classification. Hence these layers are not necessary in our case.

D. Architecture of a more advanced CNN: CNN3D

The basic CNN architecture discussed above, while well-suited for 2D chips, fails to account for the interdependence between the various silicon layers of a 3D-IC, leading to higher errors. Thus we propose another architecture, *CNN3D*, which accounts for this dependence. This is achieved by adapting the Green's function-based method for 3D chips. Thus, we need a 3D weights matrix, each layer of which is convolved with the corresponding power dissipation matrix and added. We achieve this using a CNN as follows:

1) **Input layer:** Here, the input is a 3D power dissipation matrix. The z-axis corresponds to the silicon layers in the chip.
2) **Convolutional Layer 1:** The input matrix is convolved with a weights matrix, which has the same depth as the input. The weights of each chip layer along the depth are convolved with the corresponding layer of the input and added. Zero padding is done around the edges of the input to keep the output size the same. Finally, a ReLU activation is applied.

$$\begin{aligned}\mathcal{T}_{leak,1} = P_{in}(:,:,1) \star W_1(:,:,1) + P_{in}(:,:,2) \star W_1(:,:,2) \\ + P_{in}(:,:,3) \star W_1(:,:,3) + \ldots + b,\end{aligned} \tag{6}$$

where $\mathcal{T}_{leak,1}$ is the leakage-aware thermal profile for layer 1, P_{in} is the corresponding power dissipation map, and W_1 is the weight matrix for layer 1 and b is the bias.
3) **Convolutional Layers 2,3:** We use two more identical CNN layers to help in learning the finer features; these will take process variation into account. We have empirically observed that having three CNN layers gives us the best accuracy while keeping the complexity of the system in check.
4) **Skip Connections:** When using three convolutional layers, we added an identity block after the second convolutional layer that forwards the input power map to the input of the third convolutional layer. This skip connection greatly improves the convergence of the training algorithm, as well as the accuracy for unseen floorplans.
5) **Output layer:** Finally, we get the stacked temperature matrix of each chip layer, considering leakage power and process variation.

E. Data Collection

We obtained the training data using the open-source simulator 3D-ICE. We collect the temperature profiles for multiple widely varying floorplans of different sizes after discretizing them into a 64×64 grid. The dynamic power dissipation profile is obtained in some cases from the literature and in other cases by running McPAT [20]. A few more maps are synthetically generated by modifying the power consumption of each architectural unit randomly such that it stays within the maximum allowed power dissipation. A diverse training input set is key to predicting the temperature maps for new floorplans. The coefficients in Equation 5 are obtained by curve fitting (we derive the points from the BSIM equation, Equation 4). Next, we use 3D-ICE to compute the leakage-aware temperature profiles iteratively.

F. Data Normalization

Normalizing the input data improves the numerical stability of the training algorithm. In this work, we use a modified version of the *min-max* normalization technique, which works well for process variation. The min-max normalization is given by:

$$X_{out} = \frac{X_{in} - min(X_{in})}{max(X_{in}) - min(X_{in})} \tag{7}$$

where X_{in} is the input data and X_{out} is the normalized data. To generalize our method to a variety of floorplans, we divide X_{out} by the total area of a layer of the chip, A. Note that the temperature is correlated with the power density of the chip, rather than the absolute power values.

$$X'_{out} = \frac{X_{out}}{A} \tag{8}$$

G. Training of the CNN

Using the collected data, we train the CNN with the mean absolute error as the loss function. The weights and biases are updated in each training iteration using the ADAM optimizer [21]. We use an adaptive learning rate to obtain a balance between very long training times and an unstable training process. We choose an exponential decay function in which every few steps, the learning rate is reduced by a fixed *decay_rate*. This helps in quickly moving towards the minima (convergence). Once we are in the vicinity of the minima, a lower learning rate helps in attaining the minima without overshooting it. The input training data from multiple floorplans is presented in a *randomly shuffled order* to help the CNN learn from a diverse set of inputs. This helps in quicker training and prevents overfitting, so that our CNN can adapt to new unseen floorplans.

H. GPU Implementation

To further speed up our algorithm, we implement our technique on a Tesla Turing GPU with 320 tensor cores. These Turing tensor cores have been optimized for machine learning and artificial intelligence operations. They can perform large matrix computations, and achieve high degrees of parallelism. To port from a CPU to a GPU implementation, we set up a GPU build of our framework on the Tesla T4 machine and compile our code to enable GPU support.

V. Evaluation

A. Setup

We run all of our algorithms on Ubuntu 16.04. To generate the random power consumption maps, we use an R script. A C wrapper script is used to get the leakage-converged temperature values from the open-source tool 3D-ICE. These two components are generated on a Desktop PC running Ubuntu 16.04 having 16 GB of RAM. Finally, we implement our CNN algorithm in Tensorflow 2.0 using Python 3.6 on a Tesla T4 GPU having 320 tensor cores.

1) Geometry of the Chip: We consider a 3D chip with four active layers of silicon. Under each silicon layer, there is a microchannel layer, carrying water in it and a thermal interface layer. We assume that the number of active layers in the chip remains constant. This is generally the case. The chip is discretized into a high-resolution $64 \times 64 \times 4$ grid.

2) Dataset Generation: We use the Varius toolkit [22] to generate the baseline leakage power maps in the presence of process variation. To obtain the dynamic power maps, we assume one of the chip layers to be a processor layer while the remaining are memory layers with a uniform power dissipation (generated from a uniform random distribution within the typical power dissipation range). We choose the floorplan of the processors Alpha21264, Nehalem, Gainestown, and a modified version of it to generate the training data. These floorplans have different die sizes. The power profiles are discretized to a $64 \times 64 \times 4$ grid and normalized using Equation 8. We generate a set of 5000 dynamic power maps for the processor Alpha 21264, 2000 maps for Nehalem, 1000 maps for Gainestown, and 1000 maps for a modified version of the Gainestown floorplan. We get the dynamic power dissipation profiles from either open-source tools or McPAT and vary the power consumption of the architectural units within the typical power ranges to generate more power maps. Next, we run 3D-ICE iteratively to get the leakage aware temperature maps. Each of these maps is of dimension $64 \times 64 \times 4$. The ambient temperature is 45°C. To generate each leakage-converged temperature map, 3D-ICE took over $15\ s$ on a Desktop running Ubuntu 16.04. Thus it took 38 hours to generate the data.

3) Training: We split our data into three sets: training, validation, and test data. We shuffled the order of the inputs to randomize the inputs and avoid getting stuck in local minima. The weights were initialized to a random normal distribution with a standard deviation of 0.5, while the bias was initialized to a constant value ($= 0.5$). The mean absolute error is chosen as the loss function. We use Adam optimizer [21] with an adaptive learning rate, and an exponential decay function with a starting learning rate of 0.001. After every 200 iterations, the learning rate is reduced by a factor of 0.9.

B. Results

1) CNNbasic: This architecture works better for 2D chips. Using the 3-layer architecture, we are able to train the network in 4 minutes, using 12800 (3200*4) input power maps. The mean absolute error for training is 2.5°C (3.7%), while the mean absolute error for testing is 2.6°C (3.9%) when both training and testing are done on power profiles from the Alpha21264 floorplan. The maximum temperature is 112.5°C, which means that the temperature rise above the ambient temperature is 67.5°C. The time required to predict the temperature profile for one power profile is 0.025 ms.

2) CNN3D: A CNN with a single convolutional layer takes 10 minutes to train on a GPU, with a mean absolute error of 0.79°C and a validation error of 0.8°C. The test error is 0.81°C (1.2%). The time required to predict one $64 \times 64 \times 4$ temperature profile is $0.05\ ms$. For a 3-layer residual convolutional neural network on a GPU, our algorithm takes 15 minutes to complete the training process. The mean absolute error for training is 0.61°C for a maximum temperature of 112.5°C for the Alpha 21264 floorplan, while the test error is 0.62°C (0.9%). The time required for predicting one full thermal profile is $0.1\ ms$. We choose several combinations of floorplans for training and testing. We notice that using a mix of floorplans for training improves the accuracy for all data sets. The error lies between 0.12°C and 3.3°C (4.9%) when training and testing on multiple combinations of datasets. When the floorplan being tested is a part of the training data, the error lies within 1.1°C (1.6%) for all test cases. The results are summarized in Table I. If random shuffling of the input is not done, then the algorithm takes much longer to reach the minima, especially when training on a combination of multiple floorplans.

TABLE I: Different combinations used for training and testing

Train Floorplans	Test Floorplan	Test error (°C)
alpha	alpha	0.6K
gaine1,gaine2,alpha,nehalem	gaine1	0.81
gaine1,gaine2,alpha,nehalem	gaine2	0.52
gaine1,gaine2,alpha,nehalem	nehalem	0.32
gaine1,gaine2,alpha,nehalem	alpha	1.1
gaine1,alpha, nehalem	gaine2	1.1
gaine1,gaine2,nehalem	alpha	3.3
gaine1,gaine2,alpha	nehalem	2.5

These results show that our algorithm is able to train high-resolution thermal maps with a small training set and few learnable parameters. This is primarily because we have captured the physics of heat transfer using convolution operations, rather than taking a large amount of data and letting the learning algorithm work unanchored.

C. GPU vs CPU Implementation

We first implemented a single-layer CNN on a Desktop PC. The training time of the algorithm was over two days ($\sim$ 66 hours). We then implemented the same algorithm on a server PC with a GPU and tensor cores (configuration described in Section V-A). The training time of the algorithm was brought down to under 15 minutes for the same training set. Thus we obtained a $192\times$ speedup by implementing our algorithm on a GPU.

D. Generalization to new Floorplans

To see how well our algorithm generalizes to new floorplans that it did not see in the training phase, we use power maps from two floorplans for training and test on the third floorplan. The mean absolute error using this approach lies between 1°C and 3.3°C (4.9%), for a maximum temperature rise of 67.5°C (maximum temperature = 112.5°C). Thus our method is able to generalize for a variety of floorplans, with a small error. Note that **all of these floorplans have different dimensions**.

E. Effects of Process Variation

To quantify the effects of process variation on the design, we replace the leakage power at ambient temperature by its nominal values and obtain the leakage-converged temperature map from 3D-ICE. We observe that an error of up to 15.5°C is obtained for a maximum temperature rise of 50.6°C. The average error is 5.83°C. This is equivalent to an 11% error in temperature estimation. Thus ignoring process variation in modern-day chips can lead to under-provisioning of cooling resources, which may cause hotspots and reduce the lifetime and reliability of the chip. Figure 2 shows the errors in various test cases upon ignoring the effects of process variation.

F. Comparison with State-of-the-art Approaches

1) Finite Difference/Green's Function Methods: Finite difference methods such as 3D-ICE support the modeling of complex geometries such as chips with microchannels. However, the major drawback of these methods is the slow computational speed. Additionally, most such methods model leakage power by iterating through the leakage-temperature loop, which makes them slower. It is also not easy to implement such methods on a GPU. To generate our training data, 3D-ICE took about $11 - 30s$ per simulation depending on the

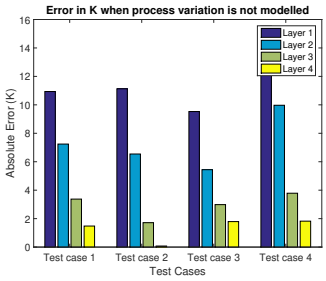

Fig. 2: Error upon ignoring process variation

number of iterations needed for convergence[1]. In comparison, our method takes 0.1 ms to compute the equivalent values, resulting in over $100,000\times$ speedup in inferencing once the longer training is completed. The accuracy of most thermal modeling methods is in the same range as ours. The open-source tool Hotspot [10] is unable to model the temperature distribution when a granularity of 64×64 is used[2]. When a power distribution granularity of 16×16 is used with a grid size of 64×64 (this is much coarser than the granularity modeled by us), HotSpot takes 2 minutes and 57 seconds to obtain the leakage converged temperature values. Green's function methods have been used to model 3D chips with microchannels, but without process variation [23]. However, such works include simple leakage models and require extra work to incorporate microchannels. The offline time of such an algorithm for a problem of the same granularity is 72 ms, and the online time is 15 ms (150X slower). Thus for finer granularities, our CNN-based method is a very promising solution.

2) Machine Learning Methods: There are a few learning-based thermal modeling methods that have tackled either 3D chips with microchannels or process variation, but not both together. Most works ignore leakage. Sridhar et al. [7] have used a single-layer neural network to model the temperature in a 3D chip with microchannels. They have not considered leakage power or microchannels. However, if we consider a four-layer 64X64 neural network, a single neuron in the hidden layer would have 16384 weights, and the entire single-layer neural network would have 268435456 weights. A similar number of weights would exist in a linear regression method [14]. Even after the proximity-based reduction in connections/symmetry-based reduction, the number of weights would still be huge. In comparison, in a CNN, for a single layer, we need 16384 weights in the complete network (plus four bias parameters). If the number of layers is increased, the number of parameters gets multiplied by the number of layers. This linear increase can be easily managed by modern computing systems. We implemented a similar NN-based method on an identical platform as ours and obtained a 0.3 ms simulation time for a single layer network with an error of over 2°C when training and testing on the same floorplan. Comparatively, our equivalent single-layer *CNN3D* method takes .05ms with an error under 1°C. Their method leads to high errors when testing on different floorplans.

VI. Conclusion

In this paper, we have proposed a CNN-based architecture for thermal simulation that is able to accurately predict the leakage aware thermal profile for complex IC geometries. It can account for the effects of process variation and is able to adapt to a variety of floorplans. Our algorithm takes the power profile only as its input. For further speedup, we have implemented it on a GPU. We have demonstrated our approach for a 4-layer 3D IC, where we obtained the temperature values for a $64\times64\times4$ grid in just 0.1 ms with a mean absolute error of 0.61°C (0.9%). As device scaling exacerbates the effects of process variation, our algorithm will increasingly prove to be very useful for modeling temperature.

[1]A small part of this time is overhead because of file operations, but still the time taken (11 s) is much higher than our algorithm (0.0001s)

[2]While HotSpot theoretically supports modeling of such a size, it was unable to read the input files we provided

References

[1] 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 *ICCAD, 2010*.

[2] S. Borkar, "Low power design challenges for the decade (invited talk)," in *ASPDAC, 2001*.

[3] J. Jaffari and M. Anis, "Statistical thermal profile considering process variations: Analysis and applications," *IEEE TCAD*, vol. 27, no. 6, pp. 1027–1040, 2008.

[4] S. Mittal, "A survey of architectural techniques for managing process variation," *CSUR*, vol. 48, no. 4, p. 54, 2016.

[5] H. Sultan and S. R. Sarangi, "A fast leakage aware thermal simulator for 3d chips," in *DATE 2017*.

[6] J.-H. Park, S. Shin, J. Christofferson, A. Shakouri, and S.-M. Kang, "Experimental validation of the power blurring method," in *SEMI-THERM*. IEEE, 2010, pp. 240–244.

[7] A. Sridhar, A. Vincenzi, M. Ruggiero, and D. Atienza, "Neural network-based thermal simulation of integrated circuits on GPUs," *IEEE TCAD*, vol. 31, no. 1, pp. 23–36, 2011.

[8] K. Zhang, A. Guliani, S. Ogrenci-Memik, G. Memik, K. Yoshii, R. Sankaran, and P. Beckman, "Machine learning-based temperature prediction for runtime thermal management across system components," *IEEE TPDS*, vol. 29, no. 2, pp. 405–419, 2017.

[9] D.-C. Juan, H. Zhou, D. Marculescu, and X. Li, "A learning-based autoregressive model for fast transient thermal analysis of chip-multiprocessors," in *ASPDAC 2012*.

[10] W. Huang, K. Skadron, S. Gurumurthi, R. J. Ribando, and M. R. Stan, "Differentiating the roles of IR measurement and simulation for power and temperature-aware design," in *ISPASS, 2009*.

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

[12] Z. Feng and P. Li, "Fast thermal analysis on GPU for 3D ICs with integrated microchannel cooling," *TVLSI*, vol. 21, no. 8.

[13] X.-X. Liu, Z. Liu, S. X.-D. Tan, and J. Gordon, "Full-chip thermal analysis of 3D ICs with liquid cooling by GPU-accelerated GMRES method," in *ISQED, 2012*, pp. 123–128.

[14] L. Siddhu and P. R. Panda, "PredictNcool: Leakage aware thermal management for 3D memories using a lightweight temperature predictor," *ACM TECS*, vol. 18, no. 5s, pp. 1–22, 2019.

[15] J. M. N. Abad and A. Soleimani, "Novel feature selection algorithm for thermal prediction model," *TVLSI*, vol. 26, no. 10, pp. 1831–1844, 2018.

[16] S. Sadiqbatcha, Y. Zhao, J. Zhang, H. Amrouch, J. Henkel, and S. X.-D. Tan, "Machine learning based online full-chip heatmap estimation," in *ASP-DAC, 2020*.

[17] S. Mahendran, H. Ali, and R. Vidal, "3D pose regression using convolutional neural networks," in *ICCV Workshops*, 2017, pp. 2174–2182.

[18] S. R. Sarangi, G. Ananthanarayanan, and M. Balakrishnan, "Lightsim: A leakage aware ultrafast temperature simulator," in *ASPDAC, 2014*.

[19] A. Pathania and J. Henkel, "Hotsniper: Sniper-based toolchain for many-core thermal simulations in open systems," *IEEE ESL*, vol. 11, no. 2, pp. 54–57, 2018.

[20] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *MICRO, 2009*.

[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[22] S. R. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas, "VARIUS: A model of process variation and resulting timing errors for microarchitects," *IEEE TSM*, vol. 21, no. 1, pp. 3–13, 2008.

[23] H. Sultan and S. R. Sarangi, "A fast leakage-aware Green's-function-based thermal simulator for 3-D chips," *TVLSI*, 2020.