Accelerating IC Thermal Simulation Data Generation via Block Krylov and Operator Action

Hong Wang, Wenkai Yang, Jie Wang[†], Huanshuo Dong, Zijie Geng, Zhen Huang,
Depeng Xie, Zhezheng Hao, Hande Dong
University of Science and Technology of China, Hefei, Anhui, China. Email: wanghong1700@mail.ustc.edu.cn

Abstract—Recent advances in data-driven approaches, such as neural operators (NOs), have shown substantial efficacy in reducing the solution time for integrated circuit (IC) thermal simulations. However, a limitation of these approaches is requiring a large amount of high-fidelity training data, such as chip parameters and temperature distributions, thereby incurring significant computational costs. To address this challenge, we propose a novel algorithm for the generation of IC thermal simulation data, named block Krylov and operator action (BlocKOA), which simultaneously accelerates the data generation process and enhances the precision of generated data. BlocKOA is specifically designed for IC applications. Initially, we use the block Krylov algorithm based on the structure of the heat equation to quickly obtain a few basic solutions. Then we combine them to get numerous temperature distributions that satisfy the physical constraints. Finally, we apply heat operators on these functions to determine the heat source distributions, efficiently generating precise data points. Theoretical analysis shows that the time complexity of BlocKOA is one order lower than the existing method. Experimental results further validate its efficiency, showing that BlocKOA achieves a 420-fold speedup in generating thermal simulation data for 5000 chips with varying physical parameters and IC structures. Even with just 4% of the generation time, data-driven approaches trained on the data generated by BlocKOA exhibits comparable performance to that using the existing method.

Index Terms—neural operator, IC thermal simulation, data generation

I. INTRODUCTION AND RELATED WORK

The temperature distribution of an integrated circuit (IC) directly affects its performance, reliability, and lifespan [1]. As a result, thermal optimization has become an essential step in the design process, involving numerous numerical thermal simulations [2]. Traditionally, thermal simulations rely heavily on computationally intensive methods [3]. To reduce simulation time and enhance the efficiency of thermal optimization, recent studies have explored data-driven approaches to solve heat equations [4], [5]. One prominent method involves neural operators (NOs) [6], [7], which can be trained on pre-generated datasets as surrogate models. During practical applications, they only require a straightforward forward pass to predict IC temperature distributions which takes only several milliseconds [8]. This is significantly faster than numerical partial differential equation (PDE) solvers, thermal resistance models, and other conventional algorithms.

However, the high computational cost of generating training datasets, which comprise chip structures and temperature dis-

tributions, presents significant challenges for these data-driven algorithms. First, chip applications often involve various chip types, and training a neural network for a specific type of IC requires a large volume of training data [9]. For example, training DeepOHeat typically requires thousands of temperature distributions under different power and IC structures [10]. Dataset generation can range from several to thousands of hours. Secondly, acquiring temperature distributions as training labels poses a further obstacle. Data generation often relies on finite element method (FEM) [3], demonstrated in Figure 1, involves solving large linear systems, with a high computational complexity that accounts for over 95% of the data generation time [11]. Thirdly, solving large linear systems often involves iterative methods such as conjugate gradient (CG) [12]. However, due to the presence of termination conditions, these methods inevitably introduce errors, which can degrade the neural network's performance [13]. Increasing solution accuracy significantly raises computational costs [14]. Furthermore, unlike other domains, IC heat equation parameters are closely tied to the characteristics of the chip, resulting in linear systems with specific structures [3]. Existing algorithms fail to exploit these structures, resulting in redundant computations. Altogether, these challenges in data generation significantly hinder the practical application of data-driven algorithms in IC thermal simulations [9].

Several studies have achieved meaningful progress. [15], [16] proposed architectures that preserve conservation laws to enhance data efficiency. [17] reduced dataset generation costs by leveraging correlations between linear systems. However, these advances mainly focused on optimizing the solution algorithms without fundamentally changing the generation approach. [18] introduced the operator action method to reduce the generation time, but it did not consider the specific structure of IC heat equations, limiting its direct applicability.

In this work, we introduce BlocKOA (Block Krylov and Operator Action), an efficient method for generating IC thermal simulation data. Initially, we use the block Krylov algorithm to simultaneously solve a small number of linear systems with different thermal parameters and quickly generate a set of temperature distributions that reflect the real IC design as basic solutions. These basic solutions are then suitably combined to satisfy the heat equation conditions and generate IC temperature distributions. Finally, we perform the action of heat operators on the temperature distributions to derive other physical parameters. The key insight behind BlocKOA

[†] Corresponding author.

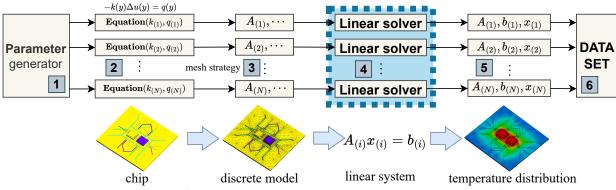


Fig. 1: The typical generation process of the thermal simulation dataset: 1. Produce a collection of random parameters derived from chips 2. Generate the relevant chips using these parameters 3. Discretize the chips using the FEM 4. Solve linear systems 5. Acquire solutions for the linear systems and convert them into temperature distributions 6. Compile the data into a dataset.

is that it reduces the redundant computation in generating basic solutions via block Krylov and avoids solving linear systems through operator action. Our contributions are as follows:

- 1. A novel dataset generation algorithm for IC thermal simulation that employs block Krylov and operator action to quickly produce large-scale data.
- 2. Theoretical analysis shows BlocKOA achieves higher precision at a lower cost, ensuring accuracy and efficiency.
- 3. Extensive experiments demonstrate that BlocKOA substantially reduces generation time, accelerating it by up to 420 times.

II. BACKGROUND

A. Mathematical Form of IC Thermal Simulation

We focuses on the generation of steady-state IC thermal simulation datasets, and the equation can be expressed as [19]:

$$-k(\mathbf{y})\Delta u(\mathbf{y}) = q(\mathbf{y}). \tag{1}$$

Here, Δ is the Laplacian; u represents the temperature distribution; k is related to IC thermal conductivity and floorplan; And q is the power density (heat generation rate) [20].

IC thermal simulations usually consider uniform boundary shapes like rectangular prisms and involve various boundary conditions [3]. We discuss the 3 most common types [21]: **Dirichlet**: $u=u_0$, modeling contact with objects at a fixed temperature. **Robin**: For simulating convective heat transfer at boundaries: $k\frac{\partial u}{\partial n} + h(u - u_\infty) = 0$. Here h is the convective heat transfer coefficient (HTC) and u_∞ is the ambient temperature. **Mixed**: Combine Dirichlet and Robin conditions.

B. Discretization for the Heat Equation

IC thermal simulation datasets are obtained by solving the corresponding heat equation, typically using discretization methods like the finite element method (FEM) [22]. These numerical methods transform the PDEs to linear systems [23]. Realistic thermal simulations, which involve more complex boundary conditions [1], require finer grid resolutions. Consequently, the matrix dimension in the resulting linear systems can increase dramatically, from 10^3 to 10^7 or more, leading to substantial computational costs in dataset generation.

Unlike other PDE datasets, the parameters in the heat equation are constrained by the specific chip structure. For example, the thermal conductivity parameter k determines A, but k cannot be generated randomly and usually depends on the specific chip structure. However, the number of publicly available chip structures is limited, leading to linear systems with the same A in the dataset. Independently solving these problems with existing algorithms can cause significant redundancy. We introduce a block algorithm to simultaneously perform Krylov subspace iterations for linear systems with the same A. By avoiding redundant computations, thereby significantly reducing the computational overhead.

C. Details of the Dataset

The heat eqution is discretized on a $N_g \times N_g \times N_g$ uniform grid $\Omega = \{(i_1/N_g, i_2/N_g, i_3/N_g) \mid i_1, i_2, i_3 = 0, 1, \dots, N_g\}$. Therefore, the dimension of the matrix A obtained from the discretized heat equation is N_g^3 . Based on the grid Ω , we generate a dataset with features $F_l = (k_l(\Omega), q_l(\Omega))$ and target $T_l = u_l(\Omega)$, where $l = 1, 2, \dots, N_{\text{data}}$. In existing data generation methods, the solution u is obtained by solving the equation with given k, q, and boundary conditions. The function k, directly related to the chip's materials and component floorplan, is limited due to the scarcity of publicly available chip designs—typically, a dataset with 10^3 samples may have only 5-100 different chip floorplans [8]. In contrast, q can be generated in many ways (e.g., random constants and Gaussian random fields consistent with chip structures), leading to distinct u values for each sample [17].

D. Direct Solution Method

Existing generation typically employs direct solution method [3], [10]. This method randomly generates q, inputs its discretized form \boldsymbol{b} into the linear system, and solves \boldsymbol{x} accordingly. This process requires solving large linear systems independently for each heat equation, which is time-consuming and becomes a bottleneck in real application.

III. METHOD

As shown in Figure 2, unlike the direct solution method, our BlocKOA method first generates the basic solutions using

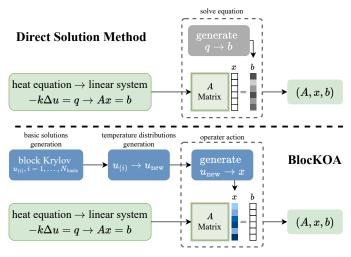


Fig. 2: Overview of the model architecture: the process of the existing direct solution method and our BlocKOA method.

the block Krylov algorithm, then combines them to produce feasible temperature distributions $u_{\rm new}$. Finally, BlocKOA inputs their discretized form x into linear systems and calculates b accordingly. Both methods produce data that comply with their respective heat equation constraints. However, BlocKOA leverages the chip structure to reduce redundant computations. It also avoids the high computational costs and termination errors typically encountered in solving large linear systems.

A. Basic Solutions Generation

The BlocKOA method randomly generates a set of IC thermal parameter distributions (e.g., $N_{\text{basis}} = 50$), like the direct solution method. Subsequently, the corresponding linear systems are solved to obtain a small set of solutions $x_{(i)}$:

$$-k_{(i)}(\boldsymbol{y})\Delta u_{(i)}(\boldsymbol{y}) = q_{(i)}(\boldsymbol{y}) \to \boldsymbol{A}_{(i)}\boldsymbol{x}_{(i)} = \boldsymbol{b}_{(i)}, \qquad (2)$$

where $i=1,\ldots,N_{\rm basis}$. If the direct solution method is used, this step requires solving $N_{\rm basis}$ linear systems independently. Given the limited number of publicly available chip component floorplans (e.g., $N_k=5$), we set $N_{\rm basis}$ as a multiple of N_k , denoted as $\eta=N_{\rm basis}/N_k=10$. This means that the resulting linear systems contain repeated matrices A:

$$\mathbf{A}_{(\eta j+1)} = \mathbf{A}_{(\eta j+2)} = \dots = \mathbf{A}_{(\eta (j+1))}, \ j = 0, \dots, N_k - 1.$$
 (3)

To improve computational efficiency, we reorganize Eq. (2). Then we apply the block Krylov algorithm to solve the linear systems with the same coefficient matrix A simultaneously:

$$\mathbf{A}_{(\eta j+1)}\mathbf{X}_{(j)} = \mathbf{B}_{(j)}, \quad j = 0, \dots, N_k - 1,$$
 (4)

where $X_{(j)} = [x_{(\eta j+1)} \mid x_{(\eta j+2)} \mid \cdots \mid x_{(\eta (j+1))}]$ and $B_{(j)} = [b_{(\eta j+1)} \mid b_{(\eta j+2)} \mid \cdots \mid b_{(\eta (j+1))}]$. The block algorithm (e.g., block CG) combines multiple linear systems into a block system [12], effectively exploiting the shared structure of A and the similarities in the computational process. By simultaneously handling multiple right-hand sides, it reduces the size of the required Krylov subspace compared to existing method that solve them independently, thereby decreasing computational costs. Finally $x_{(i)}$ are converted into the basis solutions $u_{(i)}$ according to the discretization form.

B. Temperature Distributions Generation

We apply a Gaussian distribution to randomly weight and normalize the previously obtained basis solutions $u_{(i)}$ to obtain a large number of diverse solution functions that satisfy physical constraints. Specifically, we introduce a noise element ϵ (e.g., the normal distribution) that maintains the boundary conditions unaltered, resulting in a new temperature distribution function u_{new} :

$$u_{\text{new}}(\boldsymbol{y}) = \sum_{i=1}^{N_{\text{basis}}} \alpha_i \cdot u_{(i)}(\boldsymbol{y}) + \epsilon, \qquad \alpha_i = \frac{\mu_i}{\sum_{j=1}^{N_{\text{basis}}} \mu_j}, \quad (5)$$

where $\mu_i \sim N(0,1)$, $i=1,2,...,N_{\rm basis}$. This method of weighting ensures that the newly formulated temperature distributions $u_{\rm new}$ comply with the heat equation. The incorporation of noise helps to enhance the complexity and generalization ability of the generated dataset. Through this method, we are able to generate a large number of physically meaningful temperature distributions with minimal cost.

C. Operator Action

The differential operator $-k\Delta$ in the heat equation (1) represents a mapping within the Sobolev space. This operator maps functions to other functions, as described below:

$$-k(\mathbf{y})\Delta : \mathcal{U} \to \mathcal{Q}, \quad u(\mathbf{y}) \mapsto q(\mathbf{y}).$$
 (6)

Here \mathcal{U} and \mathcal{Q} represent the Sobolev spaces of the temperature distribution and power density function, respectively. The operator action represents the action of $-k\Delta$ on u. We obtain q by applying the operator to the temperature distributions u_{new} generated in the previous step.

In practice, when applying the operator to a function, we discretize the differential equation. The operator $-k\Delta$ is represented as a linear transformation A, where u is represented by the vector x, and q is represented by the vector b. In direct solution methods, this process is transformed into solving a large linear system by computing x from A and b. However, in BlockOA, the operator action is directly represented as a matrix-vector multiplication $Ax \mapsto b$, which avoids solving the linear system. For the same problem, the computational cost of a single matrix-vector multiplication is significantly lower than that of solving the corresponding linear system, and no additional errors are introduced during the computation. Therefore, BlockOA offers greater speed and higher precision.

IV. THEORETICAL ANALYSIS

A. Computational Complexity Analysis

1) Direct Solution Method: The primary computational cost in numerical thermal simulation arises from solving the corresponding linear systems, with the Krylov subspace algorithm (e.g., CG) being among the most commonly used approaches [3], [24]. The most computationally intensive components are the matrix-vector multiplication and the orthogonalization process [25]. Assuming a matrix of dimension n and iteration count $j=1,2,\ldots,m$, where m denotes the final dimension of the Krylov subspace. The complexity per iteration is primarily determined by $O(n^2)$ for matrix-vector multiplications and O(jn) for orthogonalization. The total complexity

across m iterations can be approximated by $O(mn^2)$ for matrix-vector products and $O(m^2n)$ for orthogonalization, yielding an overall cost of approximately $O(mn^2+m^2n)$. Furthermore, practical complexity is influenced by matrix sparsity. For a dataset with $N_{\rm data}$ data points, the computational complexity can be by $O(mn^2N_{\rm data}+m^2nN_{\rm data})$.

2) BlocKOA Method: Let $N_{\rm basis}$ represent the number of basis solutions, which is typically much smaller than the dataset size $N_{\rm data}$ (e.g., $N_{\rm basis}=50$ and $N_{\rm data}=5\times10^3$). The BlocKOA method consists of 3 steps: 1. The time complexity for "Basic Solution Generation" can be expressed as $O(mn^2N_{\rm basis}+m^2nN_{\rm basis})$. 2. Constructing the temperature distributions from the basic solutions involves only matrix and vector additions, which contribute negligible computational cost. 3. The computational cost for the operator action is equivalent to a single matrix-vector multiplication, with a complexity of $O(n^2)$ for dense matrices. Specifically, matrix sparsity affects this cost. For a dataset with $N_{\rm data}$ data points, this part incurs a computational cost of $O(n^2N_{\rm data})$.

Consequently, the overall complexity of BlocKOA can be approximated by $O(n^2N_{\rm data}+mn^2N_{\rm basis}+m^2nN_{\rm basis})$. Compared to the direct solution method, BlocKOA is computationally advantageous because $N_{\rm basis}$ is significantly smaller than $N_{\rm data}$. Additionally, since m is of the same order as n (typically between n/20 to n/5 in experiments), our approach theoretically provides an approximate speedup of $O(m)\approx O(n)$, which corresponds to an increase in speed by one order. The block algorithm can be used to significantly reduce the number of iterations of "Basic Solution Generation" by sharing the Krylov subspace. Typically m can be reduced to m/10-m/2 in experiments. Therefore, using the block algorithm can speed up our algorithm by 2-10 times as a whole.

B. Error Analysis

When constructing a thermal simulation dataset, errors stem mainly from three sources: 1. PDE modeling error; 2. grid discretization error; 3. linear system error. Our analysis focuses on the third component, assuming the first two are negligible.

Iterative methods (i.e., CG) generate approximate solutions that improve with each iteration. Let error be $e_m = \| \boldsymbol{x}_m - \boldsymbol{x} \|$, where m is the number of iterations, x_m is the solution obtained at the m-th iteration, x is the true solution. e_m is related to the condition number κ of \boldsymbol{A} and the initial error e_0 [14]: $e_m \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^m e_0$. Reducing e_m requires increasing m, that will increase the computational cost.

It is generally required that the error of the dataset be significantly lower than that of the data-driven algorithm. For instance, in algorithms like [10], a final error range of $1\mathrm{E}{-2}$ to $1\mathrm{E}{-5}$ is typical, ideally with relative dataset errors is lower than $1\mathrm{E}{-7}$ to maintain training accuracy. For direct solution method, higher accuracy requires more iterations.

In BlocKOA, applying the operator to the generated temperature distributions essentially involves matrix-vector multiplication. The precision of this operation is governed by the machine epsilon of floating-point operations, generally yielding error is lower than 1E-16. Achieving this high level of precision is impossible with the direct solution method.

V. EXPERIMENT

A. Experimental Details

1) IC Problem Details: We consider the thermal simulation for 3D-ICs. Specifically, we discuss a rectangular chip measuring $10 \,\mathrm{mm} \times 10 \,\mathrm{mm} \times 0.51 \,\mathrm{mm}$. The chip consists of three device layers, each of 0.15 mm thickness, namely: 1. the topmost core layer; 2. two L2 Cache layers with identical structures. Beneath each device layer lies a 0.02 mm-thick thermal interface material (TIM) layer (comprising bumps, redistribution layers, pads, and underfills). The address and data buses between the L2 Cache and core layer modules are interconnected using through-silicon vias, which are made of copper. The thermal conductivity values used in the simulation are 150 W/mK for silicon, 413 W/mK for copper, and an equivalent thermal conductivity of 40 W/mK for the TIM layer. In the core layer, we place 156 active blocks. 6 of them are randomly selected as high-power modules, with power densities randomly assigned between 3 and $6 \,\mathrm{W/mm^2}$. The remaining 150 modules have power densities randomly assigned between 0.5 and 1 W/mm². Each L2 Cache layer consists of two L2 Caches, with power densities randomly assigned between 0.02 and $0.04 \,\mathrm{W/mm^2}$.

We consider 3 boundary conditions: 1. Dirichlet: All boundary temperatures are fixed at 50°C. 2. Robin: All boundaries have a HTC of 3×10^4 W/m²K. 3. Mixed: The top and bottom surfaces are Robin boundaries with HTC at 3×10^4 W/m²K, and the other four sides are fixed at 50°C. We consider 5 mesh resolutions, corresponding to matrix dimensions ranging from 6.0×10^4 to 2.7×10^5 . These meshes are obtained by dividing them using the open-source tool Gmesh [26]. Each dataset considers N_k (e.g. $N_k = 5$) different component floorplans, which are manually designed to mimic real chips rather than being randomly generated. For each component floorplan, several reasonable power distributions are generated using the above random strategy. Then we use the open-source finite element IC thermal simulation tool, manchester thermal analyzer (MTA) [24], to obtain the linear systems. Finally, we use the professional linear system solver library PETSc to obtain the chip temperature distributions [27].

- 2) Baselines Details: The main time expense of the existing direct solution method is solving linear systems composed of large sparse symmetric matrices [23]. In all experiments, We do not consider the time of the finite element numerical discretization process. We use the direct solution method based on the CG algorithm as our baseline, utilizing PETSc 3.19 [27]. PETSc is a state-of-the-art linear solver library and serves as the underlying solver for many professional thermal simulation software packages (e.g., DEAL.II [28], FEniCS [29], OpenFOAM [30], MTA [23]).
- 3) Experimental Environment: The data generation process is performed on an Intel i7-13700KF CPU, and the neural operator training is performed on a RTX 3090 GPU.

TABLE I: Comparison of data generation time (in seconds) for BlocKOA and CG under different IC scenarios. BlocKOA achieves a machine precision error of 1E-16. The first row lists IC boundary conditions, while TIME1 and TIME2 in the third row represent the total data generation time and operator action time, respectively. Other parameters in the third row represent CG errors (relative residual norm). Dim represents the matrix dimension.

	DIRICHLET				ROBIN				MIXED									
DIM	BLOG	CKOA		C	G		BLOG	CKOA		C	G		BLOG	CKOA		C	3	
	TIME1	TIME2	1E-3	1E-5	1E-7	1E-9	TIME1	TIME2	1E-3	1E-5	1E-7	1E-9	TIME1	TIME2	1E-3	1E-5	1E-7	1E-9
6.0×10^{4}	2.1E1	1.4E0	2.2E3	3.5E3	4.5E3	5.7E3	2.4E1	1.6E0	2.2E3	3.8E3	5.4E3	5.9E3	2.0E1	1.3E0	2.4E3	3.8E3	5.7E3	7.4E3
8.5×10^{4}	3.8E1	2.1E0	3.3E3	5.6E3	7.8E3	1.0E4	4.5E1	3.0E0	3.9E3	6.2E3	7.5E3	9.0E3	3.5E1	1.8E0	3.8E3	6.3E3	7.9E3	9.5E3
1.0×10^{5}	5.2E1	2.7E1	5.1E3	8.1E3	1.2E4	1.6E4	5.6E1	3.2E0	5.8E3	9.6E3	1.2E4	1.5E4	5.0E1	3.2E0	6.1E3	9.7E3	1.3E4	1.5E4
1.3×10^{5}	7.0E1	2.3E1	7.5E3	1.3E4	1.8E4	2.2E4	8.5E1	4.1E0	8.7E3	1.5E4	1.9E4	2.4E4	6.8E1	3.8E0	9.9E3	1.5E4	2.3E4	2.9E4
2.7×10^{5}	2.7E2	4.2E1	3.4E4	5.1E4	6.6E4	8.4E4	2.9E2	6.1E0	3.8E4	6.0E4	8.5E4	1.0E5	2.7E2	7.0E0	4.3E4	6.5E4	9.1E4	1.1E5

TABLE II: Comparison of data generation time (in seconds) and training results (RMSE) across different models (test data is generated by CG). The first row lists the IC boundary conditions, the second row lists the training data generation times and neural operators, the first column list the methods used for training data generation and N_{data} is in brackets.

Метнор		DIRICHLE	Т		ROBIN			MIXED		
METHOD	TIME(s)	FNO	DEEPONET	TIME(s)	FNO	DEEPONET	TIME(s)	FNO	DEEPONET	
CG (500)	1.64E3	2.41E-3	6.44E-4	1.50E3	8.07E-3	1.20E-2	1.54E3	8.52E-3	8.01E-3	
CG (1000)	3.28E3	2.13E-3	3.41E-4	3.01E3	7.39E-3	8.31E-2	3.08E3	7.32E-3	6.84E-3	
BLOCKOA (500)	5.24E1	5.61E-3	1.68E-3	5.62E1	1.23E-2	1.35E-2	4.85E1	1.49E-2	1.48E-2	
BLOCKOA (2000)	5.33E1	2.17E-3	3.64E-4	5.70E1	6.38E-3	7.59E-3	4.93E1	8.31E-3	5.49E-3	
BLOCKOA (5000)	5.44E1	1.85E-3	3.12E-4	5.82E1	6.14E-3	6.74E-3	5.04E1	7.12E-3	5.38E-3	

B. Comparative Experiments with Direct Solution Method

We tested the data generation time for generating $N_{\rm data}=5\times 10^3$ data points at different accuracies as shown in Table I. BlocKOA method consistently demonstrates remarkable acceleration compared to the CG method across datasets of all experiments. We set the number of generated basis solutions $N_{\rm basis}=50$ and the number of different chip component floorplans $N_k=5$. The noise ϵ is uniform randomly assigned within the range of -0.01 to 0.01 at each grid point.

First, the experimental results show that compared to the CG method, the BlocKOA method achieves a speedup of approximately 420 times in terms of the total time, while the time for operator actions can be accelerated by up to approximately 1.7×10^4 times for the matrix dimension of 2.7×10^5 . The data generation time in BlocKOA is divided into two components: basis solutions generation and operator action. Experimental results show that compared to the total time (TIME1), the operator action time (TIME2) is negligible, with basis solutions generation being the main part of BlocKOA's generation time. This is because operator action essentially involves a single matrix-vector multiplication, which has a minimal computational cost.

Second, during the CG algorithm's process of solving linear systems, as the accuracy requirement increases, the solution time significantly increases. For instance, when the accuracy of the CG algorithm is improved from 1E-3 to 1E-9, the time increases by 160% to 220%. In contrast, our BlocKOA method achieves a precision of 1E-16. This indicates that improving the accuracy of the CG algorithm comes with expensive computational costs, while our algorithm guarantees data accuracy at machine precision through operator actions.

Moreover, as the matrix dimensions increase, the acceleration ratio of BlocKOA compared to CG grows steadily. For instance, under the Dirichlet boundary condition, the total time speedup rises from approximately 370 to 420 when the dimension increases from 6.0×10^4 to 2.7×10^5 . This is

due to BlocKOA's computational complexity, which is one order lower, O(n), compared to the direct solution method, where n denotes the matrix dimension. This further supports the theoretical analysis in Section IV-A, highlighting the effectiveness of our method in high resolution.

Notably, the time required by BlocKOA to generate different quantities $N_{\rm data}$ of training instances remains relatively constant. The primary computational cost of BlocKOA lies in generating the basis solutions. This time depends on the number $N_{\rm basis}$ of basis solutions generated and is independent of $N_{\rm data}$. This characteristic implies that, given a fixed $N_{\rm basis}$, BlocKOA can generate an arbitrarily large amount of training data at minimal additional cost.

C. Data Validity Experiments

In these experiments, to validate the effectiveness of the data generated by BlocKOA, we focused on testing two widely recognized and extensively used NOs for IC thermal simulation: 1. FNO [6], [8], 2. DeepONet [7], [10]. We set the number of generated basis solutions $N_{\rm basis}=50$ and the number of different chip component floorplans $N_k=5$. The noise ϵ is uniform randomly assigned within the range of -0.01 to 0.01 at each grid point. Both models were evaluated using matrices of dimension 1.0×10^5 , and neural operator test data is generated by CG. The CG sets the accuracy to $1\mathrm{E}{-9}$ (relative residual norm). The results are shown in Table II.

The generation time of BlocKOA is significantly lower than that of the direct solution method with the CG method for all experiments. For instance, BlocKOA's computation time for generating 5000 data points is approximately $\frac{1}{50}$ of CG's time for generating 1000 data points. For fair comparison, all NOs test sets in our experiments were generated using the CG method. Consequently, when using datasets of equal size, models trained on BlocKOA-generated data initially showed slightly inferior performance compared to those trained on CG-generated data. However, BlocKOA's superior generation

TABLE III: Comparison of data generation time (in seconds) between BlocKOA and CG across different number of generated data N_{data} . The first row lists N_{data} , and the first column lists the methods used for data generation.

N_{data}	100	200	500	1000	2000	3000	6000	8000	10000	20000
CG	3.08E2	6.17E2	1.54E3	3.08E3	6.17E3	9.25E3	1.85E4	2.47E4	3.08E4	6.17E4
BLOCKOA	4.82E1	4.82E1	4.85E1	4.93E1	4.93E1	4.99E1	5.04E1	5.10E1	5.23E1	5.90E1

TABLE IV: Performance comparison of BlocKOA under varying number of generated basis solutions N_{basis} . The first row lists N_{basis} , the first column lists evaluation metrics: data generation time (seconds) and neural operator error (RMSE), and the second column presents baseline results from CG.

	CG	10	20	30	50	80	100
TIME (s)	3.08E3	9.98E0	2.02E1	3.03E1	5.04E1	8.08E1	1.02E2
FNO	7.32E-3	5.38E-1	4.34E-2	9.92E-3	7.12E-3	7.10E-3	7.07E-3
DEEPONET	6.84E-3	2.36E-1	3.29E-2	6.42E-3	5.38E-3	5.32E-3	5.28E-3

TABLE V: Performance comparison of BlocKOA method under varying N_k . The first row lists N_k , the first column lists evaluation metrics: data generation time (seconds) and neural operator performance (RMSE), and the second column presents baseline results from CG.

	CG	$N_k = 1$	$N_k = 2$	$N_k = 5$	$N_k = 10$
TIME (s)	3.08E3	3.88E1	4.29E1	5.04E1	8.39E1
FNO	7.32E-3	4.73E-1	3.24E-2	7.12E-3	7.08E-3
DEEPONET	6.84E-3	2.51E-1	2.75E-2	5.38E-3	5.23E-3

speed enables the creation of substantially larger datasets, which ultimately yield better model performance. A compelling example under mixed boundary conditions shows that BlocKOA can generate $10\times$ more data points in just $\frac{1}{25}$ of CG's computation time, while achieving significantly lower training errors (17% reduction for FNO and 32% reduction for DeepONet). These results robustly validate the effectiveness of our data generation method.

D. Parameter Analysis Experiments

This section examines the impact of four critical algorithmic parameters on experimental results: 1. $N_{\rm data}$: The number of generated data; 2. N_k : The number of distinct chip component floorplans; 3. $N_{\rm basis}$: The number of generated basis solutions; 4. ϵ : Noise element generation method. All experiments address chip thermal simulation problems with mixed boundary conditions, employing a matrix dimension of 1.0×10^5 . For consistent evaluation, we generate all neural operator test datasets using the CG method with $N_k=5$. CG sets the accuracy to $1\mathrm{E}{-9}$ (relative residual norm).

1) Analysis of $N_{\rm data}$: We set $N_{\rm basis}=50$, $N_k=5$, with ϵ uniformly randomized within (-0.01,0.01). The experimental results are presented in Table III. First, regardless of the $N_{\rm data}$ value, BlocKOA demonstrates significantly lower computational time compared to CG. Second, as $N_{\rm data}$ increases, BlocKOA's advantage becomes more pronounced. For instance, when $N_{\rm data}=100$, CG requires 6.4 times more computation time than BlocKOA. This ratio escalates to 1000 times when $N_{\rm data}=20000$. This behavior occurs because the CG method solves each problem independently, resulting in linear growth of computation time with increasing $N_{\rm data}$. In contrast, BlocKOA's primary computational overhead lies in

TABLE VI: Performance comparison of BlocKOA under different noise ϵ generation method. The first row lists ϵ generation methods, the first column lists evaluation metrics: data generation time (seconds) and neural operator error (RMSE), the second column presents results from CG.

	00	N- N	Ut	NIFORM RANDOM	GAUSSIAN RANDOM		
	CG	No Noise	(-0.005,0.005)	(-0.01,0.01)	(-0.02, 0.02)	$\sigma = 0.002$	$\sigma = 0.01$
TIME (s)	3.08E3	5.03E1	5.03E1	5.04E1	5.04E1	5.03E1	5.05E1
FNO	7.32E-3	7.97E-3	7.38E-3	7.12E-3	7.32E-3	7.39E-3	7.08E-3
DEEPONET	6.84E-3	7.29E-3	6.18E-3	5.38E-3	6.46E-3	6.31E-3	5.32E-3

TABLE VII: Comparison of dataset generation time (in seconds) with different BlocKOA settings.

	BLOCKOA	W/O BLOCK KRYLOV	W/O OPERATOR ACTION	W/O ALL
TIME (s)	5.20E1	1.25E2	6.83E3	1.64E4

the basis solution generation phase, which is independent of $N_{\rm data}$. $N_{\rm data}$ only affects the operator action phase, causing merely marginal increases in computation time.

- 2) Analysis of N_k : We set BlocKOA with $N_{\rm data}=5000$ and CG with $N_{\rm data}=2000$, while maintaining $N_{\rm basis}=50$ and ϵ uniformly randomized within (-0.01,0.01). Results are shown in Table V. First, N_k directly affects BlocKOA's solution time, with larger values leading to longer computation. This occurs because with fixed $N_{\rm basis}$, increasing N_k reduces the number of linear systems solved simultaneously in the block method, thereby decreasing the exploitable redundancy and increasing computation time. Second, N_k influences the quality of BlocKOA-generated data. Test dataset was generated using CG with $N_k=5$. Results show that overly small N_k values yield lower training accuracy, while values exceeding 5 produce nearly identical training outcomes. This aligns with expectations: insufficient N_k limits floorplan diversity, while excessive values introduce irrelevant information.
- 3) Analysis of N_{basis} : We set BlocKOA with $N_{\rm data} = 5000$ and CG with $N_{\rm data} = 2000$, while maintaining $N_k = 5$ and ϵ uniformly randomized within (-0.01, 0.01). Results are presented in Table IV. First, BlocKOA's data generation time increases linearly with $N_{\rm basis}$, consistent with our theoretical analysis in Section IV-A. The primary computational cost resides in the basis solution generation phase, which scales directly with $N_{\rm basis}$. Second, $N_{\rm basis}$ affects data quality. Insufficient basis solutions reduce data diversity and problem representation capability, ultimately degrading model performance. The impact becomes negligible when $N_{\rm basis} > 50$, indicating adequate diversity has been achieved.
- 4) Analysis of ϵ : We set BlocKOA with $N_{\rm data}=5000$, CG with $N_{\rm data}=2000$, $N_{\rm basis}=50$, and $N_k=5$. Results are shown in Table VI, where "No noise" denotes absence of additive noise, and Gaussian random represents zero-mean normal distribution with standard deviation σ . Results demonstrate that appropriate noise addition improves the training

effectiveness of BlocKOA-generated data. However, excessive noise proves detrimental.

E. Ablation Experiments

As shown in Table VII, we consider the dataset generation time of the BlocKOA method after removing different modules. We set the number of generated basis solutions $N_{\rm basis}=50$ and the number of different chip designs $N_k=5$. The noise ϵ is randomly assigned within the range of -0.01to 0.01 at each grid point. The matrix dimension is 1.0×10^5 , $N_{\rm data} = 5 \times 10^3$, and the boundary condition is Dirichlet. We replace block CG with standard CG for "w/o block Krylov". The block algorithm achieves a 2.4 times speedup in computation ("w/o block Krylov" is approximately equivalent to [18]). Additionally, our experiments reveal that the average number of iterations for CG is 980, whereas block CG requires only 224 iterations. This suggests that this acceleration stems from the block algorithm's shared Krylov subspace, which minimizes redundant computations. Moreover, BlocKOA is 130 times faster than the "w/o operator action". In summary, these experiments demonstrate the critical importance of block Krylov and operator action in the efficiency of BlocKOA.

VI. CONCLUSION

This paper presents the BlocKOA algorithm. To our knowledge, this is the first attempt to accelerate IC thermal simulation dataset generation. The BlocKOA ensures speed and accuracy, alleviating a significant obstacle to the development of data-driven algorithms in the field of IC thermal simulation.

REFERENCES

- [1] H. Delaram, A. Dastfan, and M. Norouzi, "Optimal thermal placement and loss estimation for power electronic modules," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, 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 Computing Surveys (CSUR), vol. 52, no. 2, pp. 1–35, 2019.
- [4] A. Kumar, N. Chang, D. Geb, H. He, S. Pan, J. Wen, S. Asgari, M. Abarham, and C. Ortiz, "MI-based fast on-chip transient thermal simulation for heterogeneous 2.5 d/3d ic designs," in 2022 International Symposium on VLSI Design, Automation and Test (VLSI-DAT). IEEE, 2022, pp. 1–8.
- [5] 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 2020 36th Semiconductor Thermal Measurement, Modeling & Management Symposium (SEMI-THERM). IEEE, 2020, pp. 65–75.
- [6] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, "Fourier neural operator for parametric partial differential equations," arXiv preprint arXiv:2010.08895, 2020.
- [7] L. Lu, P. Jin, and G. E. Karniadakis, "Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators," arXiv preprint arXiv:1910.03193, 2019.
- [8] M. Wang, Y. Cheng, W. Zeng, Z. Lu, V. F. Pavlidis, and W. W. Xing, "Aro: Autoregressive operator learning for transferable and multi-fidelity 3d-ic thermal analysis with active learning."
- [9] Z. Hao, S. Liu, Y. Zhang, C. Ying, Y. Feng, H. Su, and J. Zhu, "Physics-informed machine learning: A survey on problems, methods and applications," arXiv preprint arXiv:2211.08064, 2022.
- [10] Z. Liu, Y. Li, J. Hu, X. Yu, S. Shiau, X. Ai, Z. Zeng, and Z. Zhang, "Deepoheat: operator learning-based ultra-fast thermal simulation in 3dic design," in 2023 60th ACM/IEEE Design Automation Conference (DAC). IEEE, 2023, pp. 1–6.
- [11] B. Szabó and I. Babuška, "Finite element analysis: Method, verification and validation," 2021.
- [12] G. H. Golub and C. F. Van Loan, *Matrix computations*. JHU press, 2013.
- [13] L. Lu, X. Meng, S. Cai, Z. Mao, S. Goswami, Z. Zhang, and G. E. Karniadakis, "A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data," Computer Methods in Applied Mechanics and Engineering, vol. 393, p. 114778, 2022.
- [14] H. A. Van der Vorst, Iterative Krylov methods for large linear systems. Cambridge University Press, 2003, no. 13.
- [15] J. Brandstetter, R. v. d. Berg, M. Welling, and J. K. Gupta, "Clifford neural layers for pde modeling," arXiv preprint arXiv:2209.04934, 2022.
- [16] N. Liu, Y. Yu, H. You, and N. Tatikola, "Ino: Invariant neural operators for learning complex physical systems with momentum conservation," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2023, pp. 6822–6838.
- [17] H. Wang, Z. Hao, J. Wang, Z. Geng, Z. Wang, B. Li, and F. Wu, "Accelerating data generation for neural operators via krylov subspace recycling," arXiv preprint arXiv:2401.09516, 2024.
- [18] H. Dong, H. Wang, H. Liu, J. Luo, and J. Wang, "Accelerating pde data generation via differential operator action in solution space," arXiv preprint arXiv:2402.05957, 2024.
- [19] P. Li, L. T. Pileggi, M. Asheghi, and R. Chandra, "Ic thermal simulation and modeling via efficient multigrid-based approaches," *IEEE Transac*tions on Computer-Aided Design of Integrated Circuits and Systems, vol. 25, no. 9, pp. 1763–1776, 2006.
- [20] Y. Hua, Z.-Q. Wang, X.-Y. Yuan, Y. B. Li, W.-T. Wu, and N. Aubry, "Estimation of steady-state temperature field in multichip modules using deep convolutional neural network," *Thermal Science and Engineering Progress*, vol. 40, p. 101755, 2023.
- [21] J. W. Thomas, Numerical partial differential equations: finite difference methods. Springer Science & Business Media, 2013, vol. 22.
- [22] J. C. Strikwerda, Finite difference schemes and partial differential equations. SIAM, 2004.

- [23] T. J. Hughes, *The finite element method: linear static and dynamic finite element analysis.* Courier Corporation, 2012.
- [24] S. Ladenheim, Y.-C. Chen, M. Mihajlović, and V. F. Pavlidis, "The mta: An advanced and versatile thermal simulator for integrated systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 12, pp. 3123–3136, 2018.
- [25] J. Liesen and Z. Strakos, Krylov subspace methods: principles and analysis. Numerical Mathematics and Scie, 2013.
- [26] C. Geuzaine and J.-F. Remacle, "Gmsh: A 3-d finite element mesh generator with built-in pre-and post-processing facilities," *International journal for numerical methods in engineering*, vol. 79, no. 11, pp. 1309– 1331, 2009.
- [27] S. Balay, S. Abhyankar, M. F. Adams, S. Benson, J. Brown, P. Brune, K. Buschelman, E. M. Constantinescu, L. Dalcin, A. Dener, V. Eijkhout, J. Faibussowitsch, W. D. Gropp, V. Hapla, T. Isaac, P. Jolivet, D. Karpeev, D. Kaushik, M. G. Knepley, F. Kong, S. Kruger, D. A. May, L. C. McInnes, R. T. Mills, L. Mitchell, T. Munson, J. E. Roman, K. Rupp, P. Sanan, J. Sarich, B. F. Smith, S. Zampini, H. Zhang, H. Zhang, and J. Zhang, "PETSc Web page," https://petsc.org/, 2024. [Online]. Available: https://petsc.org/
- [28] D. Arndt, W. Bangerth, B. Blais, T. C. Clevenger, M. Fehling, A. V. Grayver, T. Heister, L. Heltai, M. Kronbichler, M. Maier et al., "The deal. ii library, version 9.2," *Journal of Numerical Mathematics*, vol. 28, no. 3, pp. 131–146, 2020.
- [29] M. Alnæs, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M. E. Rognes, and G. N. Wells, "The fenics project version 1.5," *Archive of numerical software*, vol. 3, no. 100, 2015.
- [30] H. Jasak, "Openfoam: Open source cfd in research and industry," International journal of naval architecture and ocean engineering, vol. 1, no. 2, pp. 89–94, 2009.