ASP-DAC 2024
29th Asia and South Pacific Design Automation Conference

SoC-Tuner: An Importance-guided Exploration Framework for DNN-targeting SoC Design

Shixin Chen, Su Zheng, Chen Bai, Wenqian Zhao,
Shuo Yin, Yang Bai, Bei Yu

Department of Computer Science and Engineering,
The Chinese University of Hong Kong

Jan. 23, 2024

Outline

1. Introduction
2. Problem Formulation
3. SoC-Tuner Framework
4. Experiments & Analysis
5. Conclusion

Introduction

Introduction: DNNs & DNNs Deployment

DNNs of Various Applications

DNNs' excellent capability comes with the cost of massive computation, so it is necessary to design excellent devices for deployment.

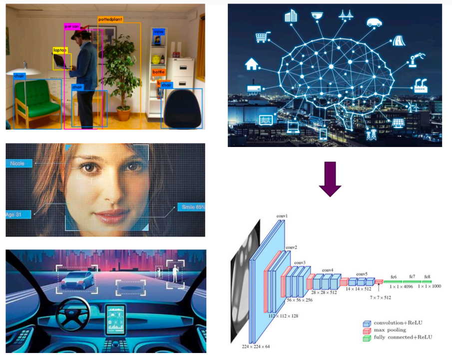
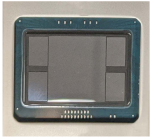


Fig.1 Various Applications using DNNs

(a) CPU (Intel Xeon)

(b) FPGA (Xilinx U280)

(c) GPU (Nvidia 3090)



(d) SoC (Google TPU v4)

Fig.2 Deployment Devices for DNNs

DNNs Accelerators

Lots of accelerators are proposed to accelerate the computation of DNNs. The core idea is to minimize the data movement and improve computation parallelism.

- Eyeriss[1]
- FlexFlow[2]
- MAGNet[3]

[1] Yu-Hsin Chen, Joel Emer, and Vivienne Sze (2016). "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks". In: *ACM SIGARCH computer architecture news* 44.3, pp. 367–379.

[2] Wenyan Lu et al. (2017). "Flexflow: A flexible dataflow accelerator architecture for convolutional neural networks". In: *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, pp. 553–564.

[3] Rangharajan Venkatesan et al. (2019). "Magnet: A modular accelerator generator for neural networks". In: *Proc. ICCAD*. IEEE, pp. 1–8.

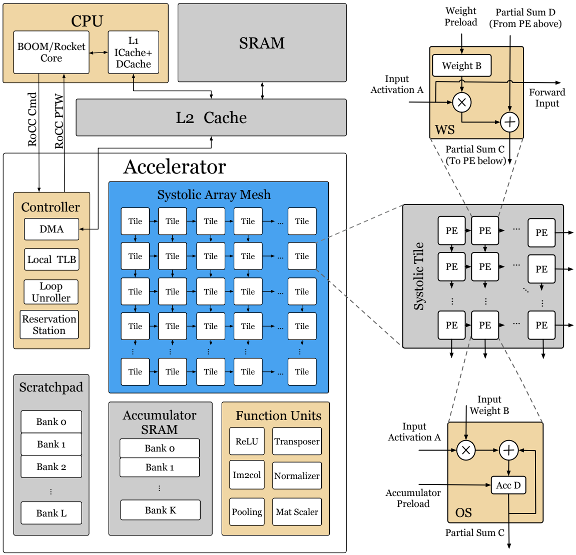


Fig.3 An SoC Architecture Containing a DNN Accelerator

Designing Challenges:

- Previous works mainly focus on the accelerator, ignoring the complicated interactions between modules.
- Huge design space from component parameters makes it necessary to design an efficient exploration strategy.
- Inaccurate analytical models and simplified simulation tools may mislead design direction.

Challenge: Huge design space from component parameters

Table: Selected Design Space Parameters from the SoC

Modules	Components	Descriptions	Candidate Values
CPU & L2 Cache	HostCore	Various Host CPU core	core1, core2, core3
	L2Bank	Entries of L2 cache banks	1, 2, 4
	L2Way	Entries of L2 cache way	4, 8, 16
	L2Capa	Capacity of each L2 cache bank	128, 256, 512
Systolic Array	Tilerow/col	Dimension of the tile in mesh	1, 2, 4, 8
	Meshrow/col	Dimension of the mesh in systolic array	8, 16, 32, 64
	Dataflow	Dataflow mode of systolic array	WS, OS, BOTH
	InputType	Bit width of input datetype	8, 16, 32
	AccType	Bit width of accumulator datatype	8, 16, 32
	OutType	Bit width of output datetype	8, 20, 32
Accelerator Memory	SpBank	Banks of scratchpad memory	4, 8, 16,32
	SpCapa	Entries of each scratchpad bank	64, 128, 256, 512
	AccBank	Banks of accumulator memory	1, 2 ,4 ,8
	AccCapa	Entries of each accumulator bank	64, 128, 256, 512
Accelerator Controller	LdQueue	Entries of the `Lord` instruction queue	2, 4, 8, 16
	StQueue	Entries of the `Store` instruction queue	2, 4, 8, 16
	ExQueue	Entries of the `Execute` instruction queue	2, 4, 8, 16
	LdRes	Entries of the `Lord` instruction in ROB	2, 4, 8, 16
	StRes	Entries of the `Store` instruction in ROB	2, 4, 8, 16
	ExRes	Entries of the `Execute` instruction in ROB	2, 4, 8, 16
Communication	MemReq	Number of memory requests in-flight	16, 32, 64
	DMABus	Width of DMA bus	32, 64, 128
	DMABytes	Number of bytes in DMA bus	32, 64, 128
	TLBSize	Size of TLB page	4, 8, 16

Problem Formulation

Problem Formulation

Notation

- $\mathbf{x}$: a combination of feature parameters is denoted as a design point of SoC
- $\mathcal{X}$: all design points make up the entire design space
- $\mathbf{y} = \mathbb{F}(\mathbf{x}) = \{f_1(\mathbf{x}), \cdots, f_{\mathbf{m}}(\mathbf{x})\}$: evaluation metrics reported by VLSI tools, like chip area, power consumption, and latency of DNNs computation.

SoC Design Problem

Given $\mathcal{X}$, the SoC design problem is to find optimal $\mathbf{x}$ that minimizes chip area and power consumption constraints, and latency of DNNs computation.

Pareto Optimal Set

For an optimization problem, an objective $\mathbf{y} = \mathbb{F}(\mathbf{x})$, an n-dimensional vector, is said to be dominated by $\mathbf{y}' = \mathbb{F}(\mathbf{x}')$ if

$$\begin{aligned} &\forall i \in [1, n], \mathbb{F}_i(\mathbf{x}) \leq \mathbb{F}_i(\mathbf{x}'); \\ &\exists j \in [1, n], \mathbb{F}_j(\mathbf{x}) < \mathbb{F}_j(\mathbf{x}'). \end{aligned} \tag{1}$$

In this way, we denote $\mathbf{x}' \succcurlyeq \mathbf{x}$. A set of design points that are not dominated by any other points form the *Pareto optimal set*. In the Pareto optimal set, a design point can not be optimized without sacrificing other objectives.

SoC Design Space Exploration

We define the SoC design space exploration as to find a subset $\mathcal{X}^* \in \mathcal{X}$, whose corresponding metrics $\mathcal{Y}^*$ form the Pareto optimal set. Hence,
$\mathcal{Y}^* = \{\mathbf{y}' | \mathbf{y}' \not\preccurlyeq \mathbf{y}, \forall \mathbf{y} \in \mathcal{Y}\}, \mathcal{X}^* = \{\mathbf{x} | \mathbb{F}(\mathbf{x}) \in \mathcal{Y}^*, \forall \mathbf{x} \in \mathcal{X}\}$

SoC-Tuner Framework

SoC-Tuner: Overview of Framework

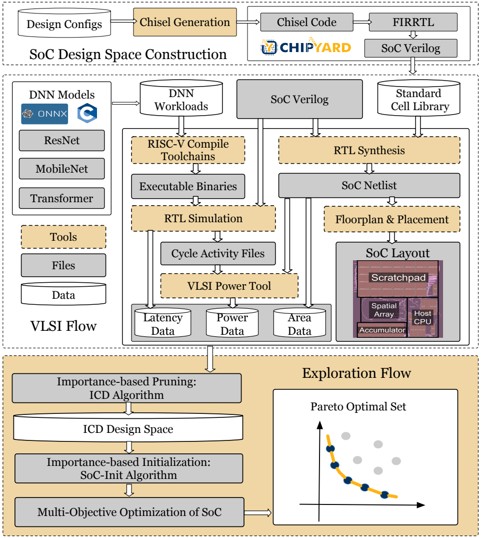


Fig.6 The overall flow of the proposed SoC-Explorer framework

SoC Design Space Construction

- Instead of using analytical models or simplified simulation tools, our exploration goal is to find actual RTL-level SoC design.
- Chipyard[4] is an open-source SoC generation framework written in Chisel language, which can generate RTL-level design that can be fabricated.
- We use Python to implement a Chisel Generation Tool, which can automatically take in design points and generate Chisel-based SoC design.

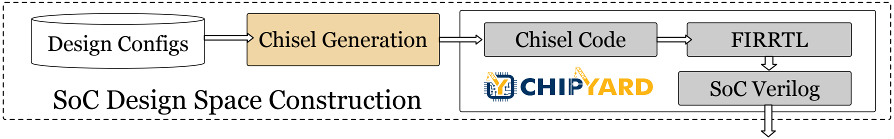


Fig.7 SoC Design Space Construction

[4] Alon Amid et al. (2020). "Chipyard: Integrated design, simulation, and implementation framework for custom SoCs". In: *IEEE Micro* 40.4, pp. 10–21.

VLSI-Flow

Through VLSI tools, we can get accurate metrics to guide the design of SoC.

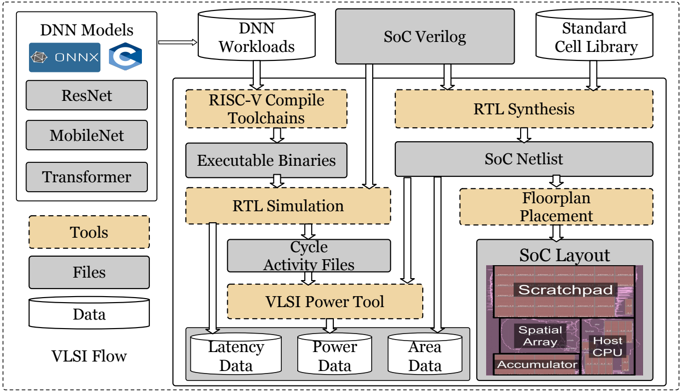


Fig.8 VLSI Flow to Get Metrics of SoC

Exploration of SoC-Tuner

- Space Pruning: ICD Algorithm
- Efficient Sampling: SoC-Init Algorithm
- Exploration Optimization: Information-guided Bayesian Algorithm

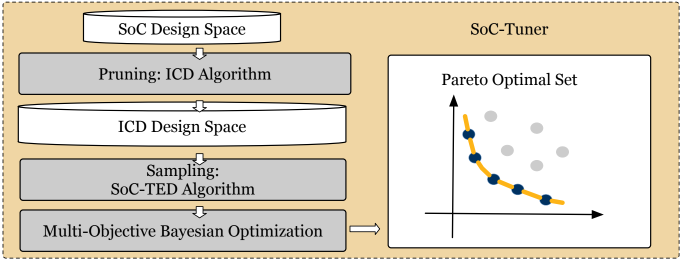


Fig.9 Backbone of SoC-Tuner

Pruning Space

We utilize ICD results to prune the design space as follows

$$\text{if } \mathbf{v}_i < \mathbf{v}_{threshold}, \text{ then } \forall \mathbf{x} \in \mathcal{X}, \mathbf{x}_i = medium(\{\mathbf{x}_i^1, \cdots, \mathbf{x}_i^j\}), \tag{2}$$

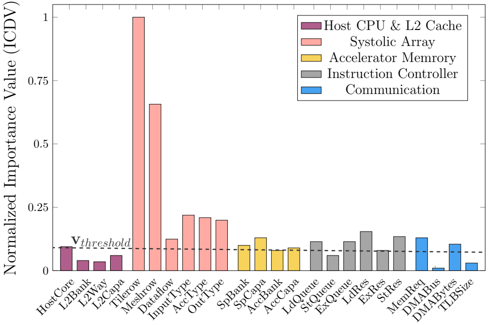


Fig.10 Importance Analysis Given by ICD Algorithm

- The more important feature will have more influence on the metrics of the SoC.
- We use the Inter-Cluster Distance (ICD) to analyze parameter importance.

$$v_i = \frac{\sum_{p,q} ||m_p, m_q||_2}{C_2^{|M|}}, p, q \in \{1, 2, ..., t_i\}, \tag{3}$$

ICD Space

We utilize ICD results to transform the original space into ICD space, where similar design points will move closer and different design points will move further.

$$\mathcal{X}' = \{\mathbf{v} \odot \mathbf{x}, \forall \mathbf{x} \in \mathcal{X}\}, \tag{4}$$

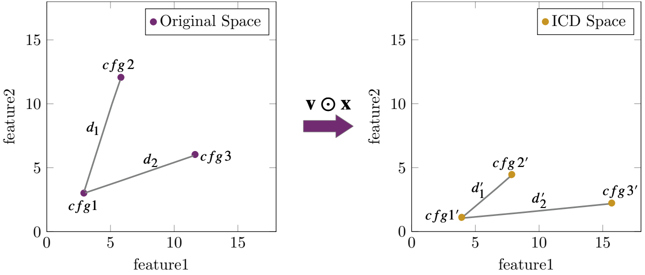


Fig.11 A toy example with 2 features shows the transformation from the original space to the ICD space.

SoC-Tuner: SoC-Init Algorithm

- We adapt TED (Yu, Bi, and Tresp 2006) method into the SoC design problem and form the SoC-Init Algorithm.
- SoC-Init algorithm will sample the most representative design points that scatter in the whole design space base on features.

Algorithm 2 `SoC-Init`$(\mathcal{X}, u, b, \boldsymbol{v}, v_{th})$

Input: $(\mathcal{X}, u, b, \boldsymbol{v}, v_{th})$, where $\mathcal{X}$ is the un-sampled design space, μ is the normalization coefficient, b is the number of configurations we will sample, $\boldsymbol{v}$ is the ICD feature vector from Algorithm 1.

Output: $\mathcal{Z}$, the sampled set with $|\mathcal{Z}| = b$.

$\mathcal{Z} = \emptyset$; if $v_i < v_{th}$, then $\forall \boldsymbol{x} \in \mathcal{X}, \boldsymbol{x}_i = \text{medium}(\{\boldsymbol{x}_i^1, \cdots, \boldsymbol{x}_i^j\})$;

$\mathcal{X}' = \{\boldsymbol{v} \odot \boldsymbol{x}, \forall \boldsymbol{x} \in \mathcal{X}\}$;

$\boldsymbol{K} = \{\Phi(\boldsymbol{x}_i', \boldsymbol{x}_j') | \boldsymbol{x}_i', \boldsymbol{x}_j' \in \mathcal{X}'\}$; ▷ $\Phi(.)$ is Euclidean distance.

for $i \in \{1, 2, \cdots, b\}$ **do**

  $\boldsymbol{z} = \text{argmax}_{\boldsymbol{x}' \in \mathcal{X}'} = \dfrac{||\boldsymbol{K}_{\boldsymbol{x}'}||^2}{\Phi(\boldsymbol{x}', \boldsymbol{x}') + \mu}$; ▷ $\boldsymbol{K}_{\boldsymbol{x}'}$ and $\Phi(\boldsymbol{x}', \boldsymbol{x}')$ are $\boldsymbol{x}'$'s corresponding column and diagonal entry in $\boldsymbol{K}$.

  $\mathcal{Z} = \mathcal{Z} \bigcup \{\boldsymbol{z}\}$;

  $\boldsymbol{K} = \boldsymbol{K} - \dfrac{\boldsymbol{K}_{\boldsymbol{z}} \boldsymbol{K}_{\boldsymbol{z}}^\top}{\Phi(\boldsymbol{z}, \boldsymbol{z}) + \mu}$;

end for

return $\mathcal{Z}$;

Gaussian Process

$$\mathbb{F} = [f(\mathbf{x}'_1), f(\mathbf{x}'_2), \cdots, f(\mathbf{x}'_n))]^T \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{K}_{\mathcal{X}'\mathcal{X}'|\boldsymbol{\theta}}), \tag{5}$$

where $\mathbf{K}_{\mathcal{X}'\mathcal{X}'|\boldsymbol{\theta}}$ is the intra-covariance matrix among all feature vectors and can be computed via $[\mathbf{K}_{\mathcal{X}'\mathcal{X}'|\boldsymbol{\theta}}]_{ij} = k_{\boldsymbol{\theta}(\mathbf{x}'_i, \mathbf{x}'_j)}$, and Gaussian noise $\mathcal{N}(f(\mathbf{x}'), \sigma_e^2)$ is to model uncertainties of GP models.

Bayesian Models

Given a newly sampled feature vector $\mathbf{x}'_*$, the predictive joint distribution f_* based on $\mathbf{y}$ can be calculated according to Equation 6.

$$f_*|\mathbf{y} \sim \mathcal{N}(\begin{bmatrix} \boldsymbol{\mu} \\ \mu_* \end{bmatrix}, \begin{bmatrix} \mathbf{K}_{\mathcal{X}'\mathcal{X}'|\boldsymbol{\theta}} + \sigma_e^2\mathbf{I} & \mathbf{K}_{\mathcal{X}'\mathbf{x}'_*|\boldsymbol{\theta}} \\ \mathbf{K}_{\mathbf{x}'_*\mathcal{X}'|\boldsymbol{\theta}} & k_{\mathbf{x}'_*\mathbf{x}'_*|\boldsymbol{\theta}} \end{bmatrix}). \tag{6}$$

Information-guided Optimization

We attempt to maximize the information gained about the Pareto optimal set $\mathcal{Y}^*$. The information gain-based acquisition function $I(\mathbf{x}')$ can be expressed as with entropy H(.), We use

$$
\begin{aligned}
I(\mathbf{x}') &= I(\{\mathbf{x}', \mathbf{y}\}, \mathcal{Y}^* | \mathcal{X}') && (7)\\
&= H(\mathcal{Y}^* | \mathcal{X}') - \mathbb{E}_{\mathbf{y}}[H(\mathcal{Y}^* | \mathcal{X}' \cup \{\mathbf{x}', \mathbf{y}\})]. && (8)\\
\mathbf{x}^* &= argmin_{\mathbf{x}'} I(\mathbf{x}') && (9)
\end{aligned}
$$

To conclude, we can define **SoC-Tuner Algorithm** as the overall BO algorithm of SoC-Tuner:

$$\mathbf{x}^* \leftarrow \textbf{SoC-Tuner}(\mathcal{X}', \mathcal{Y}^*, \boldsymbol{\theta}), \tag{10}$$

Experiment

Experiment: Baseline & Setting

Experiment Setitng

In the setting of SoC-Tuner, we set $n = 30$ for the ICD algorithm, $\mathbf{v}_{th} = 0.12$ for pruning design space, $u = 0.1$ and $b = 20$ for SOC-Init algorithm, $T = 120$ for SoC-Tuner.

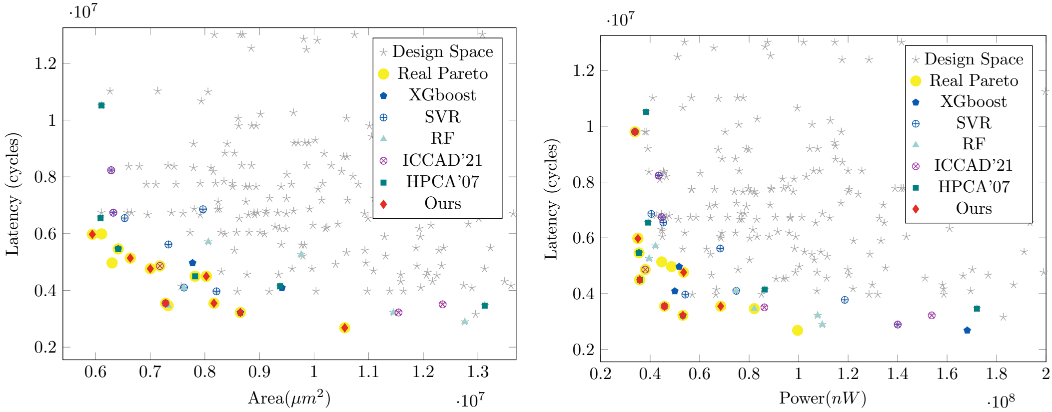
Baselines

- XGboost
- Support Vector Regression (SVR)
- Random Forest (RF)
- ICCAD'21[5]
- HPCA'07[6]

[5] Chen Bai et al. (2021). "BOOM-Explorer: RISC-V BOOM Microarchitecture Design Space Exploration Framework". In: *Proc. ICCAD*, pp. 1–9. DOI: 10.1109/ICCAD51958.2021.9643455.

[6] Benjamin C Lee and David M Brooks (2007). "Illustrative design space studies with microarchitectural regression models". In: *Proc. HPCA*. IEEE, pp. 340–351.

Experiment: Pareto Optimal Set



(a) The learned Pareto optimal set (inference latency *v.s.* chip area)

(b) The learned Pareto optimal set (inference latency *v.s.* power consumption)

Fig.13 The Pareto optimality set given by SoC-Tuner (ResNet50)

ADRS

$$ADRS(\Gamma, \Omega) = \frac{1}{|\Gamma|} \sum_{\gamma \in \Gamma, \omega \in \Omega} min \quad f(\gamma, \omega), \tag{11}$$

where f is the Euclidean distance function. Γ is the real Pareto optimality set and Ω is the learned Pareto optimality set.

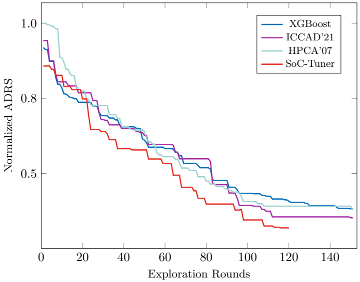


Fig.14 The ADRS curves of different exploration methods

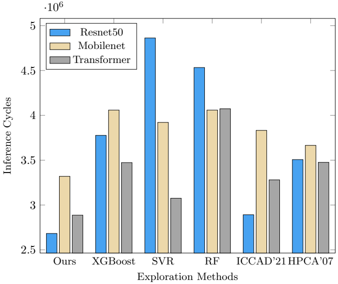


Fig.15 The inference cycles of the optimal SoC designs given by various methods

Table: The optimal SoC design explored by SoC-Tuner

Components	Values	Components	Values
HostCore	core1	AccBank	8
L2Bank	1	AccCapa	128
L2Way	8	LdQueue	4
L2Capa	512	StQueue	8
Tilerow/col	4	ExQueue	8
Meshrow/col	8	LdRes	8
Dataflow	OS	StRes	8
InputType	8	ExRes	8
AccType	16	MemReq	16
OutType	20	DMABus	64
SpBank	8	DMABytes	128
SpCapa	256	TLBSize	4

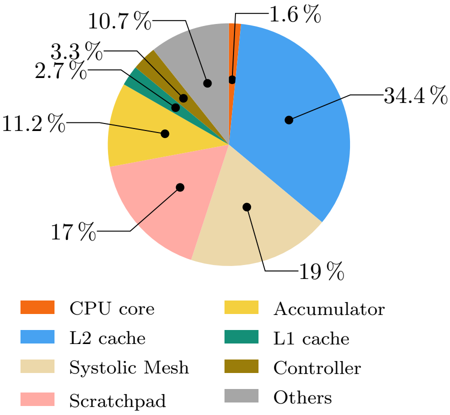


Fig.16 The area breakdown of the optimal SoC

Why SoC-Tuner Effective?

- We thoroughly consider various SoC components that influence DNN computations and construct a huge design space to avoid insufficient evaluation of overall DNN inference.
- We employ actual very-large-scale-integration (VLSI) flow to evaluate multiple metrics, which achieves more accurate modeling of SoC than simplified analytical tools.
- We propose an importance-based analysis to prune the design space, a sampling algorithm to select the most representative initialization points, and an information-guided multi-objective optimization method to balance multiple design metrics of SoC design.
- Experimental results demonstrated the efficiency and effectiveness of our framework on various benchmarks compared to some state-of-the-art methods.

Q & A

THANK YOU!