CompoundEye: A 0.24-4.17 TOPS Scalable Multi-Node DNN Processor for Image Recognition

Xiaobai Chen, Qiurun Hu, Fu Xiao and Jieming Yin*
School of Computer Science, Nanjing University of Posts and Telecommunications, China
Email: {chenxb86, 1220045017, xiaof, jieming.yin}@njupt.edu.cn, *Corresponding Author

***Abstract*—This paper proposes a scalable DNN processor that can be flexibly reconfigured to maximize inference efficiency on a wide range of DNN models. The processor consists of 18 computing nodes with various precision modes support. To improve the computation throughput, we propose a sub-image parallelization strategy, where the original input image is divided into multiple sub-images and computed on multiple nodes in parallel. In addition, the cross-layer pipeline is implemented to improve resource utilization. The proposed processor is implemented in 28nm CMOS technology and achieves a peak performance of 4.17 TOPS and an energy efficiency of 2.08 TOPS/W.**

***Index Terms*—DNN, precision, processor, scalable, pipelining**

I. Introduction

Image recognition is one of the essential Deep Neural Network (DNN) applications [1]–[3]. DNN models become larger and more complex to achieve higher recognition accuracy, which demands more powerful hardware to execute. However, depending on the applications, the performance requirement of DNNs varies. For example, the performance of a tiny network on energy-constrained edge devices would be very different from a large network in data centers. Similarly, the hardware (i.e., DNN accelerators) optimized for smaller networks might not be as efficient when executing larger networks. It is challenging to design a DNN accelerator that delivers good computational performance, and meanwhile can adapt to various DNN models while maintaining good efficiency. In this paper, we seek a path toward a one-size-fits-all solution and aim to design a dedicated image recognition DNN processor optimized for both performance and flexibility.

A large number of DNN processors have been proposed over the past years, most of which have been focusing on optimizing single-core performance, thereby improving the overall processor performance [4]–[6]. Some designs further push the performance limit by introducing multiple Piles [7]. However, single-core or Pile organizations lack flexibility and scalability. Different DNN models or even different layers in the same model might have diverse computation/bandwidth/accuracy requirements [8]. While some designs propose to augment each PE with additional hardware to support different precision modes, not all the layers can utilize the extra hardware all the time, therefore resulting in a waste of resources. Pipeline mechanisms have been proposed to improve the throughput of DNN processors [9]. Pipelining is beneficial when a large number of consecutive images can be fed into the processor. However, most edge devices only need to process a small number of images at a time, and therefore cannot take full advantage of pipeline computation.

To address the above challenges, this paper introduces CompoundEye, a scalable DNN processor consisting of 18 computing Nodes. The Nodes can perform DNN computation independently or combined to output greater throughput, thus providing high performance and great flexibility for DNN applications in different scenarios. To further improve performance, we propose cross-layer pipeline and sub-image parallelism strategies, where a centralized controller is introduced to support data transfer between the sub-images. The accelerator is implemented in 28nm CMOS technology and can achieve a peak performance of 4.27 TOPS.

II. CompoundEye Architecture

A. Overall Architecture of CompoundEye

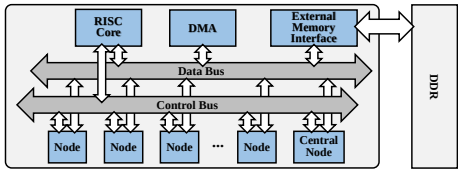

Fig. 1: Overall architecture of CompoundEye.

CompoundEye comprises 18 Nodes, a RISC core, a DMA engine, and an external memory interface, as shown in Figure 1. The Nodes are connected with the RISC core, DMA, and memory interface through the data bus. The Central Node is the data exchange center for the other 17 Nodes. The RISC core loads the DNN model parameters and generates control and configuration instructions, including the segmentation of the DNN model, the allocation of computing tasks for each Node, and the reorganization of the intermediate data. These instructions are sent to the Nodes via the control bus.

B. Node Architecture

The Node architecture is shown in Figure 2. Nodes are connected through the data bus and can handle all DNN operations independently. Each Node consists of four PE arrays. The PEs can be reconfigured to support multi-precision convolutional computation (see Sec III-C). Each Node contains weight register files (Weight RegFiles) and activation register files (AT RegFiles) to enable data reuse. The Weight RegFiles transfer weights to the four PE arrays, then these weights are broadcasted to all 16 PEs in each PE array. AT RegFiles are connected to the PEs on the right edge of each PE array, so each PE array receives the same activations. The Node has two Data Buffers to form a Ping-Pong buffer system. The

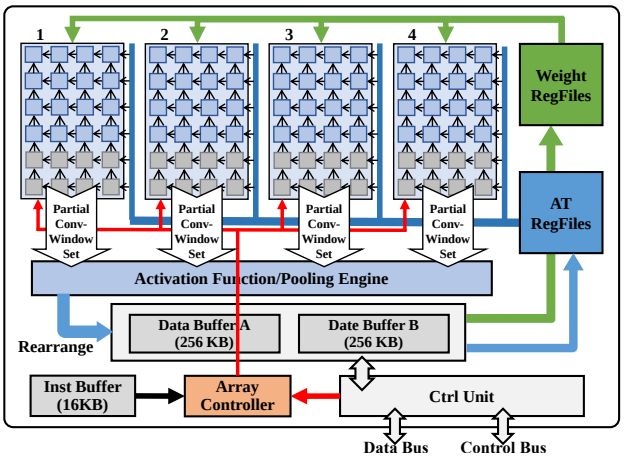

Fig. 2: Architecture of a single Node.

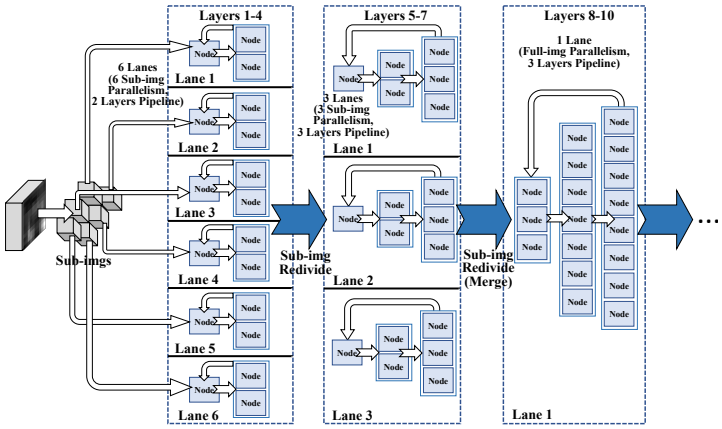

Fig. 3: Cross-Layer Pipelining for VGG-16.

Data Buffers transfers features to AT RegFiles and weights to Weight RegFiles. After performing activation function, pooling, and rearranging operations, outputs of the PE arrays become input feature maps of the next layer and are stored in the Data Buffers. The operands in the PE array can flow between adjacent PEs from right to left or from bottom to top. Four PE arrays, Weight RegFiles, and AT RegFiles form the output stationary (OS) dataflow. We chose OS dataflow to retain the operands inside the PE at the maximum, which is conducive to reusing weights and activations in temporal and spatial manners. OS dataflow reduces the bandwidth and power consumption caused by data transmission. The Array Controller loads the instructions from the Inst Buffer to configure the Node. The Control Unit manages the status of the Node and communicates with the data and control buses.

III. Computation Strategy and Dataflow

A. Cross-Layer Pipelining of Sub-images

Cross-layer pipelining allows different layers of a DNN model to be executed on a processor, which effectively improves performance and resource utilization [10]. Historically, cross-layer pipelining techniques can hardly reach their full potential due to limited on-chip storage. In this paper, we combine cross-layer pipelining with image segmentation techniques to address the storage challenges.

1) Pipelining Strategy: In our design, the input image is divided into N_{sub} sub-images to form a sub-image pipeline. As shown in Figure 3, these sub-images form a cross-layer pipeline across all Nodes. Since each Node only processes one sub-image at a time, so long as the input image is partitioned appropriately, the on-chip SRAM in each Node can easily store all the operands, eliminating the expensive off-chip communication. The value of N_{sub} can affect the pipeline performance. If N_{sub} is too small, each sub-image can be too large to fit in the on-chip storage, so off-chip data transfer becomes unavoidable. If N_{sub} is too large, each sub-image can be too small to fully utilize the Node's computation resource, which impacts the pipelining performance. In our design, N_{sub} will be determined by the RISC core for each layer according to 1) the parameters of the DNN model and 2) the on-chip storage capacity inside a Node. Note that in CompoundEye, one buffer (e.g., Data Buffer A) stores the input activation and the current layer's weights, and the other buffer (e.g., Data Buffer B) stores the output activation and the next layer's weights. So in principle, the size of a sub-image must be less than half of a Node's storage capacity, i.e., $N_{sub} \geq 2 \times \frac{S_{feature}}{S_{Node}}$ ($S_{feature}$ is the size of the input feature map, S_{Node} is the storage in each Node).

2) Pipelining Implementation: To determine the number of DNN layers K that participate in cross-layer pipelining at a given time, we need to consider both performance and storage requirements. If K is too small, we may not take full advantage of pipelining and harm performance. If K is too large, the intermediate data generated may not fit in the on-chip storage, so off-chip memory accesses are needed. In this paper, Node is the basic allocation unit for cross-layer pipelining, so the number of Nodes also limits the value of K, i.e., $K \times M \times N_{sub} \leq N_{Node}$, where M is the average number of Nodes allocated to each layer in the pipeline. In addition, the selection of K should consider the load balancing of each layer during cross-layer pipelining. Specifically, the ratio of computation demand of each layer $P_{comp} = C_1 : C_2 \cdots : C_K$ and the number of Nodes allocated to each layer $P_{Node} = M_1 : M_2 \cdots : M_K$ needs to satisfy $P_{comp} \approx P_{Node}$. The selection of N and K will be determined through experiment (Section IV-B), and the key metrics are performance and off-chip accesses.

Take VGG-16 as an example (Figure 3), in the first four layers, the size of the input feature map is large (the storage requirement can reach 1.6MB). Considering the on-chip storage limitation, we divide the input into six sub-images ($N_{sub} = 6$). Therefore, each sub-image can be assigned to 3 Nodes ($18/6 = 3$). Then, we need to determine the value of K (the number of DNN layers that participate in cross-layer pipelining at a given time). Possible K values are 1, 2, and 3. We find that layer 2 does more computation than layer 1). To balance the load of each layer in the pipeline, we divide the three Nodes into two groups: one Node processes the first layer, and the other two Nodes handle the second layer, that is, K=2. Meanwhile, the weight parameters of these two layers can all be stored in the Nodes without accessing the

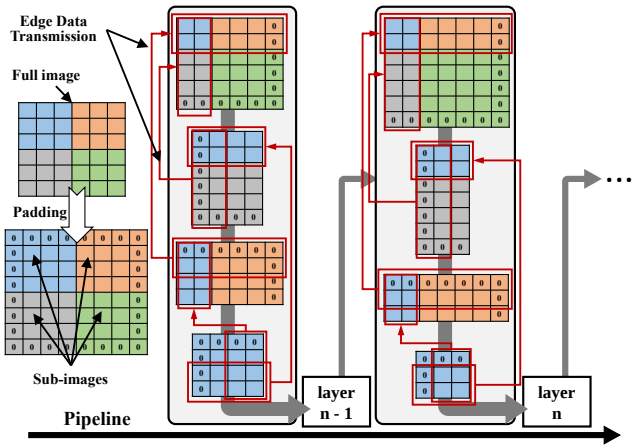

Fig. 4: Edge data exchange between sub-images.

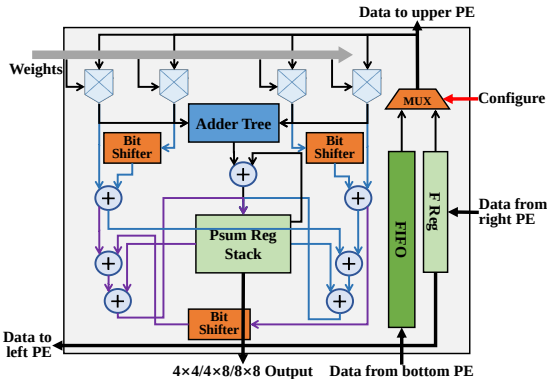

Fig. 5: PE architecture.

off-chip memory. For layers 3 and 4, it is also found that the size of features and weights are similar to that of the first two layers, and so are the computation loads. Therefore, we apply the same strategy to layers 3 and 4. As the size of the feature map decreases and the weight increases, when the weight size is close to or exceeds $\frac{S_{Node}}{2}$, the sub-image should be re-divided. We analyze the computation loads for layers 5-7, whose $P_{comp} \approx 1:2:3$. To balance the load we set $P_{Node} = 1:2:3$, that is, $K = 3$. And CompoundEye is accordingly re-partitioned into three clusters, each with six Nodes. When the size of the input feature is less than $\frac{S_{Node}}{2}$, no sub-image needs to be divided since the feature map can be stored in a single Node (i.e., $N_{sub} = 1$). For example, for layers 8 and later, the feature map sizes are less than 0.2MB, but the weight increases to more than 0.6MB. The feature size is already very small, and the weight of a single layer can no longer be stored in the Node. Therefore, instead of dividing sub-images, we keep all the features in each Node. So, we set $K = 3$ for layers 8-10. For layers 11-16, we set $K = 6$, we can flexibly allocate the 18 Nodes to the six layers. In this process, the Nodes of each layer process the convolutional kernel in parallel. Ideally, these Nodes can be combined to store all the weights of the layer, effectively reducing the access to off-chip memory when the weight size increases.

B. Dataflow of the Sub-images

For convolution computation, the sliding of the convolution kernel windows on the feature maps is continuous. However, the segmentation of the sub-images breaks the integrity of the original image. Therefore, we must transfer the data on the edges of sub-images between Nodes to ensure the correct computation. As shown in Figure 4, the image is padded and divided into four sub-images coded with different colors. Each sub-image is dispatched to a Node for computation. To ensure correctness, the two right-most columns of the light blue sub-image need to be transferred and concatenated with the dark blue sub-image. Likewise, the bottom-most two rows of the light blue image must be transferred and concatenated with the gray sub-image, and so on. The data dependency of the sub-images is regular. We have studied two methods for edge data transfer. In the first method, each sub-image exchanges edge data with all adjacent sub-images. This method ensures that the size of every sub-image in each layer stays constant, but its communication is too costly to be suitable for cross-layer pipelining. In the second method, the sub-images only transfer the edge data with the adjacent sub-images on the left and above in one direction w.r.t. the original input images or input feature map. Compared with the first method, the communication cost is reduced by nearly $\frac{1}{4}$. Therefore, CompoundEye adopts the second method.

C. Multi-Precision Support

The PE architecture is shown in Figure 5. Each PE contains four 4×4bit multipliers, one addition tree, and a one-bit shifter that allows the PE to be reconfigured to 4×8bit or 8×8bit precision. To enable data reuse and reduce off-chip accesses, additional memory modules such as FIFO, F Reg, and Psum Reg Stack are implemented in the PE for supporting OS dataflow and caching activations. Specifically, F Reg is used to receive data from the adjacent PE on the right. The data in F Reg will be transferred to the adjacent PEs or sent to the computing unit inside the PE according to the control signals, so that the operands can flow flexibly in the PE Array. FIFO is used to store the feature data transferred from the adjacent PE below, so that the feature data of the lower half of the convolution window does not need to be loaded from the SRAMs. The feature data in the FIFO will also be transmitted to the adjacent PE above or to the computing unit inside the PE the next time. The MUX selects whether to load the data from FIFO or F Reg according to the control signal.

Each Node can perform three precision modes: 4×4bit *N-l* (low-precision) mode, 4×8bit *N-m* (medium-precision) mode, and 8×8bit *N-h* (high-precision) mode. As shown in Figure 5, the black, cyan, and purple lines represent the dataflow of *N-l*, *N-m*, and *N-h* mode, respectively. In *N-l* mode, each PE can simultaneously process four pairs of activations and weights with the same relative position on different channels of a convolution window. The number of PEs corresponding to the three precision modes is determined according to the proportion of three precisions required in a particular DNN layer. Assuming that the number of multiply-accumulate (MAC) with 4×4, 4×8, and 8×8bit precision accounts for $a_1\%$, $a_2\%$, and $a_3\%$, respectively. P_a represents the proportion of different computation precision tasks for the

TABLE I: Configurations for VGG-16

Layer	N_{sub}	K	P_{Node}	PE Utilization
1-4	6	2	1 : 2	0.91
5-7	3	3	1 : 2 : 3	0.94
8-10	1	3	3 : 7 : 8	0.94
11-16	1	6	3 : 3 : 4 : 4 : 2 : 2	0.96

TABLE II: P_b and performance for VGG-16

P_b	2:5:5	3:5:4	5:3:4	7:2:3	9:1:2
TOPS	4.05	4.17	3.88	2.96	1.46

TABLE III: Performance and energy efficiency comparison

Designs	DaDianNao [7]	STICKER [5]	JSSC'20 [6]	CompoundEye
Technology	28nm	65nm	65nm	28nm
Area (mm^2)	$\geq$1.88[a] @1Node	12	10.24	3.9 @1Node
Storage	$\geq$2.25MB @1Node	170KB	364KB	552KB @1Node
Max Freq.	606MHz	200MHz	160MHz	530MHz
Precision	16/32 FP	8 FP	8 FP	4/8 FP
Power (mW)	$\geq$384 @1Node $\geq$6150 @16Node	248	120.5	106.4 @1Node 2014 @18Node
Performance (TOPS)	0.348 @1Node 5.58 @16Node	0.102	0.134 chips	0.24 @1Node 4.17 @18Node
TOPS/W	0.91	0.48	1.03	2.08

[a]Area of on-chip memory not included

DNN model and $P_a = a_1 : a_2 : a_3$. In a Node, if the number of PEs in *N-l*, *N-m*, and *N-h* modes account for $b_1\%$, $b_2\%$, and $b_3\%$, respectively, then the proportion of three precision PEs of the Nodes is $P_b = b_1 : b_2 : b_3$. We need to configure the Nodes in a way that P_a and P_b are as close as possible (i.e., $P_a \approx P_b$) to achieve the best resource utilization.

IV. Evaluation

A. Implementation

We implement CompoundEye in Verilog RTL. To obtain the processor's area and power numbers, we synthesize, placed and routed the design with Synopsys Development Kits [11] under 28nm CMOS technology [12]. We use CACTI [13] and Memory Compiler [12] to model the DRAM memory and on-chip SRAM buffers, respectively. Each Node contains 552KB of on-chip SRAM. The supply voltage can be dynamically changed between 0.6-1.0V, where the frequency varies between 50-530MHz. In total, a Node consumes $3.95mm^2$ area (1.82mm$\times$2.17mm) and 106.4mW of power.

B. Optimizations

The following key factors prevent CompoundEye from achieving its optimal performance. Firstly, spatial-temporal idle bubbles are generated and cause pipeline stalls. For example, the current sub-image cannot immediately progress to the next Node when waiting for the edge data of other sub-images. Additionally, the number of Nodes with the corresponding precision might not perfectly match the sub-images, which causes some Nodes to stay busy while others idle. Secondly, when CompoundEye changes mode, it needs to first collect the feature maps of each Node, reorganize them, then broadcast them to the PEs, during which the Nodes must be suspended. Lastly, for some DNN models, the feature maps in certain layers are too small to fully utilize the PE Array, resulting in performance loss. As a result, to improve the performance and energy efficiency of CompoundEye, we optimize the number of sub-image N_{sub}, the number of cross-layer pipeline K, the Nodes proportion for the K layers pipeline P_{Node}, and the proportion of three precision P_b. The parameters and performance numbers for VGG-16 are shown in Table I and Table II. The tables show that longer cross-layer pipelining leads to better PE utilization. The performance becomes optimal when the precision ratio provided by CompoundEye matches the precision requirements of the DNN model. In cases of unmatched precision, CompoundEye needs to supplement high-precision PEs for low-precision operations, thus significant performance degradation is observed in Table II.

Sub-image parallelization is more efficient for larger input features because it dramatically reduces off-chip memory accesses. While this work focuses on 3×3 filters (most DNN models use 3×3 filters), other filter sizes (e.g., 5×5, 7×7) can be easily supported at the expense of lower PE utilization or be executed in the RISC core.

C. Comparisons

CompoundEye allows the 18 Nodes to be flexibly configured, where the number of active Nodes called N_{ac} can be set to any number from 1 to 18. In our experiment, we can obtain a minimum computing performance of 2.23 ($N_{ac} = 1$ Node) to a maximum of 4.17 TOPS ($N_{ac} = 18$ Nodes). CompoundEye's design is flexible and scalable. On the one hand, one can increase the number of Nodes to improve the performance further. On the other hand, by reducing the number of Nodes, CompoundEye enjoys very low power consumption. Table III shows the comparison results against state-of-the-art designs. Compared with the well-known multi-node DNN processor DaDianNao [7], CompoundEye achieves 2.28$\times$ better energy efficiency with smaller on-chip storage. Compared with other designs, CompoundEye achieves over 20$\times$ higher performance and can be reconfigured to support various DNN applications.

V. Conclusion

In this paper, we introduced CompoundEye, a scalable DNN processor for image recognition applications. The processor can be reconfigured to adapt to various computation precision requirements for different DNN models and achieves great energy efficiency. To further unleash CompoundEye's potential, we have proposed an image partition strategy and optimized the cross-layer pipelining mechanism based on DNN applications' requirements. Evaluation results showed that CompoundEye outperforms state-of-the-art DNN accelerators in terms of TOPS-per-watt.

Acknowledgment

The authors sincerely thank the anonymous reviewers for their constructive feedback and suggestions. This work was supported in part by National Nature Science Foundation of China grants #62102193, State Key Laboratory of Computer Architecture (ICT CAS) grants #CARCH202106, Jiangsu Distinguished Professor Award, and two startup fundings from the Nanjing University of Posts and Telecommunications.

REFERENCES

[1] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[2] A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Neural Information Processing Systems*, vol. 25, 01 2012.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[4] C.-H. Lu, Y.-C. Wu, and C.-H. Yang, "A 2.25 TOPS/W Fully-Integrated Deep CNN Learning Processor with On-Chip Training," in *2019 IEEE Asian Solid-State Circuits Conference (A-SSCC)*, 2019.

[5] Z. Yuan, Y. Liu, J. Yue, Y. Yang, J. Wang, X. Feng, J. Zhao, X. Li, and H. Yang, "STICKER: An Energy-Efficient Multi-Sparsity Compatible Accelerator for Convolutional Neural Networks in 65-nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 2, pp. 465–477, 2020.

[6] S. Choi, J. Sim, M. Kang, Y. Choi, H. Kim, and L.-S. Kim, "An Energy-Efficient Deep Convolutional Neural Network Training Accelerator for In Situ Personalization on Smart Devices," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 10, pp. 2691–2702, 2020.

[7] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "DaDianNao: A Machine-Learning Supercomputer," in *47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2014.

[8] L. Lu, N. Guan, Y. Wang, L. Jia, Z. Luo, J. Yin, J. Cong, and Y. Liang, "Tenet: A framework for modeling tensor dataflow based on relation-centric notation," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021.

[9] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016.

[10] Y. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. R. Pinckney, and P. Raina, "Simba: Scaling deep-learning inference with multi-chip-module-based architecture," 2019.

[11] Synopsys, "EDA Tools, Semiconductor IP and Application Security Solutions," https://www.synopsys.com.

[12] SMIC, https://www.smics.com.

[13] R. Balasubramonian, A. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "CACTI 7: New Tools for Interconnect Exploration in Innovative Off-Chip Memories," *ACM Transactions on Architecture and Code Optimization*, vol. 14, pp. 1–25, 06 2017.