NLCMAP: A FRAMEWORK FOR THE EFFICIENT MAPPING OF NON-LINEAR CONVOLUTIONAL NEURAL NETWORKS ON FPGA ACCELERATORS

Giuseppe Aiello[1], *Beatrice Bussolino*[1], *Emanuele Valpreda*[1], *Massimo Ruo Roch*[1], *Guido Masera*[1], *Maurizio Martina*[1], *Stefano Marsi*[2]

[1]Politecnico di Torino, Department of Electronics and Telecommunications
[2]Università di Trieste, Department of Engineering and Architecture

ABSTRACT

This paper introduces NLCMap, a framework for the mapping space exploration targeting Non-Linear Convolutional Networks (NLCNs). NLCNs [1] are a novel neural network model that improves performances in certain computer vision applications by introducing a non-linearity in the weights computation. NLCNs are more challenging to efficiently map onto hardware accelerators if compared to traditional Convolutional Neural Networks (CNNs), due to data dependencies and additional computations. To this aim, we propose NLCMap, a framework that, given an NLC layer and a generic hardware accelerator with a certain on-chip memory budget, finds the optimal mapping that minimizes the accesses to the off-chip memory, which are often the critical aspect in CNNs acceleration.

Index Terms— Non-linear signal processing, Convolutional Neural Networks, Dataflow, HW Mapping

1. INTRODUCTION

Convolutional Neural Networks (CNNs) are nowadays dominating in a vast range of fields in which AI is adopted, such as robotics, healthcare, business and finance, or computer vision. CNNs have achieved outstanding results particularly in the field of image processing, exceeding the accuracy of humans in tasks such as object recognition and tracking, image reconstruction, or noise removal. However, these cutting-edge performances are obtained at the cost of heavy memory and computational requirements.

For this reason, the hardware acceleration of neural networks has become relevant in the last years, as shown by the explosion of architectures dedicated to these workloads. Among the various available hardware platforms, e.g. GPUs or ASICs, FPGAs are often the first choice for edge computing. Their structure allows in fact to obtain high parallelism and throughput, maintaining however the power consumptions low.

In the context of image processing, in particular, in the case of high resolution or high FPS requirements, it is very often not possible to store all the data and intermediate results in the on-chip memory. For this reason, data need to be continuously moved between the off-chip and the on-chip memory, increasing the consumed power. Moreover, modern architectures are facing the von Neumann bottleneck, i.e., the maximum performance is limited by the memory bandwidth rather than by the availability of computational resources. To overcome this issue, several optimizations can be applied at various levels, e.g., algorithmic, software, or hardware.

Recently, a novel neural network architecture, called Non Linear Convolutional Network (NLCN) [1], has been proposed. In traditional CNNs the kernels are learned at training time and are then used as constant coefficients during the inference. On the other hand, NLCNs use space-variant kernels, i.e., the weights, adopted in the convolution, are input dependent and continuously updated during inference time. Therefore, the more complex kernels in NLCNs make the system more capable of adating to local spatial variations in the input images.

Moreover, as demonstrated in [1], on equal performance, NLCNs require a lower number of layers compared to CNNs. This makes them a good candidate for embedded FPGA accelerators, since they have an overall lower computational and memory footprint. However, the advantages of NLCNs come at the cost of more complex NLC layers. As it will be shown in the following sections, a NLC layer requires a double set of weights and two convolutions, which make the mapping of the operations on hardware accelerators much harder, as in the case of FPGAs with limited on-chip memory. To tackle these limitations, we propose **NLCMap**, the first framework for the automatic mapping of NLC layers on generic hardware accelerators with limited on-chip memory and bandwidth.

The paper is organized as follows: in Sec. 2 we give the necessary background on NLC layers and present the related works on automatic mappings of neural networks on hardware accelerators, in Sec. 3 we present the NLCMap framework, and in Sec. 4 we provide a use case.

2. BACKGROUND AND RELATED WORKS

2.1. Non-Linear Convolutional Networks (NLCNs)

The main novelty of the NLC layer proposed in [1] is the substitution of the fixed weights with a set of space-variant

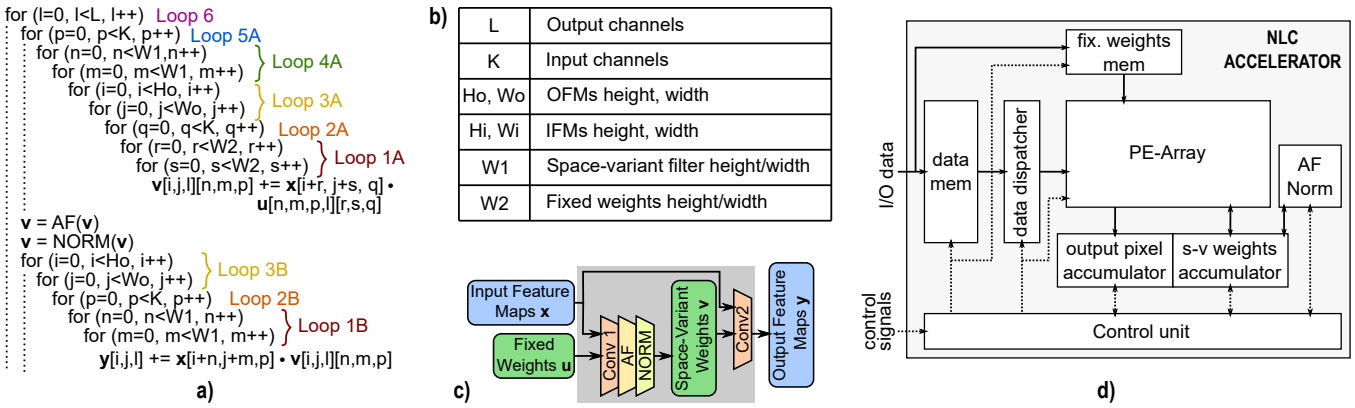


Fig. 1. **a)** Pseudocode of a NLC layer, **b)** dimensions of the input feature maps (IFMs), output feature maps (OFMs), and weights, **c)** block diagram of a NLC layer, **d)** NLC accelerator architecture.

weights $\mathbf{v}$, which change the response of the layer according to the characteristics of its input. To obtain this behavior, the space-variant weights $\mathbf{v}$ are computed convolving a set of constant weights $\mathbf{u}$ with the input feature maps $\mathbf{x}$, and then applying a non-linear activation function (AF) and normalization (Norm) as in (1). In (1), $o1$ and $o2$ are the padding values needed to have the output feature maps of the same size of the input feature maps. Fig. 1 (a) and (b) show the pseudocode of the computations and the parameters of a NLC layer. Fig. 1 (c) and (d) highlight a block scheme of a NLC layer and the corresponding accelerator architecture.

$$\mathbf{v}_{i,j,l}(n,m,p) = \mathrm{Norm}\Big\{\mathrm{AF}\Big\{\sum_{q=1}^{K}\sum_{r=1}^{W2}\sum_{s=1}^{W2} \mathbf{x}(i+r-o2, j+s-o2, q)\cdot \mathbf{u}_{n,m,p}(r,s,q)_l\Big\}\Big\} \quad (1)$$

$$\mathbf{y}_{i,j,l} = \sum_{p=1}^{K}\sum_{n=1}^{W1}\sum_{m=1}^{W1} \mathbf{x}(i+n-o1, j+m-o1, p)\cdot \cdot \mathbf{v}_{i,j,l}(n,m,p)$$

The proper activation and normalization functions must be chosen depending on the application, as they affect the filter response of the layer. A practical use of NLCNs for image denoising is shown in [2].

2.2. Hardware Mapping Space Exploration

Many prior works have proposed solutions to efficiently map convolutional layers onto hardware accelerators, differing mainly in how the design space is explored. [3] and [4] propose mapping algorithms dedicated to specific architectures. In [5, 6, 7, 8] the design space is reduced by considering only a subset of the possible computation orders, while [9, 10] focus on the tiling factors. [11] considers a limited set of mappings and chooses among them. [12] considers all the components of hardware mapping (loop order, tiling, unrolling), but pruning the search space, while [13] and [14] explore the full space with exhaustive and random search.

3. NLCMAP WORKFLOW AND DETAILS

As shown in Section 2, a NLC layer requires a larger memory to store the weights and the intermediate results of the operations. Moreover, in NLC layers it is also necessary to take into account the data dependencies when tiling the computations. This leads to a more complex management of data movement between off-chip and on-chip memories, in particular in systems in which the latter is limited, such as FPGAs. For this reasons it is crucial to carefully orchestrate and optimize the dataflow, to maximize the data reuse and limit data movements between different levels of memory.

To this end, we propose **NLCMap**, a framework that given a NLC layer configuration and a generic hardware accelerator (Sec. 3.1), optimizes the scheduling of data movements and operations. The space of the possible dataflows (Sec. 3.2) is pruned with heuristic knowledge and by swiping through the loop factors in a discrete way. The on-chip memory requirements (Sec. 3.3) and off-chip memory accesses (Sec. 3.4) are estimated analytically, and these estimations are used to find the optimal dataflow given an on-chip memory budget.

3.1. Accelerator structure

The accelerator used for the estimations of memory accesses and data movements has a generic structure, including: an array of processing engines (PEs) to perform multiply-and-accumulate (MACs) operations in parallel, on-chip data memories dedicated to the inputs (input feature maps and fixed weights) and the intermediate/output results (space-variant weights and output feature maps). Data are loaded from an off-chip memory, and a computational unit is dedicated to the activation functions and normalization.

3.2. Dataflow and Tiling

Mapping a layer to an accelerator requires the definition of the dataflow, i.e., the order and parallelism of the operations, and of the tiling strategy [15, 16]. In the following, these

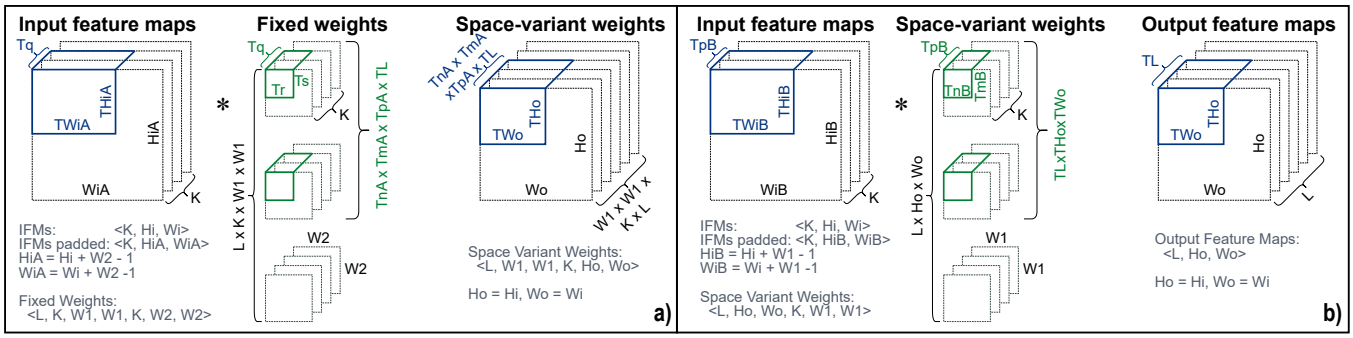


Fig. 2. Volumes and tiling factors for the computation of: **a)** the space-variant weights, **b)** the output pixels.

three components will be briefly explained, together with the considerations that need to be made for NLC layers mapping.

- **Loop reordering**: The order of the loops in Fig.1 will determine the data dependencies and the opportunities of data reuse, both spatial and temporal. Optimizing the loop order will therefore reduce the data movements between off-chip and on-chip memory. When reordering the loops of a NLC layer, particular attention must be paid to data dependencies. When performing the normalization, all the weights $\mathbf{v}_{i,j,l}(:,:,:)$ along directions W1, W1, and K are needed to compute one element $\mathbf{v}_{i,j,l}(n,m,p)$. Therefore, we impose to complete the loops 5A and 4A *before* scheduling the normalization. In this way the un-normalized weights can be kept in the on-chip memory, an the normalization is then performed without needing to fetch data from the off-chip memory. This heuristic constraint prunes the space of possible loop orders.

- **Loop tiling**: loop tiling consists in splitting large loops in groups of smaller loops, reducing the number of operands (IFMs, weights, OFMs) that need to be loaded in the on-chip memory of the accelerator. In this scope, the tile size along a dimension d will be denoted as $\mathrm{T}d$. Refer to Fig. 2 for the graphical representation of the tiling factors along each dimension.

- **Loop unrolling - Parallelization**: the execution of a loop can be speeded up by scheduling part of its operations in parallel, taking advantage of hardware parallelism. As a consequence, the required hardware resources, i.e.. the number of PEs, are dependent from the parallelization factors, and viceversa. For a NLC layer, it is necessary to account for the multiply-and-accumulate operations (N_M) of *two* convolutions. Knowing the parallelization factors, denoted in this paper as $P(\mathrm{T}d)$ for tile $\mathrm{T}d$, it is possible to analytically compute the number of MACs required by each of the convolutions (2) as:

$$\begin{aligned} N_{M_{c1}} &= P(\mathrm{Tr}) \cdot P(\mathrm{Ts}) \cdot P(\mathrm{Tq}) \cdot \Phi_A \cdot P(\mathrm{TL}), \\ N_{M_{c2}} &= \Phi_B \cdot P(\mathrm{TL}), \end{aligned} \tag{2}$$

where $\Phi_d = P(\mathrm{THi}d) \cdot P(\mathrm{TWi}d) \cdot P(\mathrm{Tn}d) \cdot P(\mathrm{Tm}d) \cdot P(\mathrm{Tp}d)$ with $d \in \{\mathrm{A}, \mathrm{B}\}$ and all the parameters are defined in Figs. 1 (b) and 2.

3.3. On-chip memory size

The on-chip memory must be large enough to store all the data used during the execution of a computational block, i.e., the input pixels I_PXL, the fixed weights F_W, the space-variant weights SV_W and the output pixels O_PXL (3) and can be evaluated as:

$$\sigma\_mem = \mathrm{I\_PXL} + \mathrm{F\_W} + \mathrm{SV\_W} + \mathrm{O\_PXL}, \tag{3}$$

where the different contributions depend on the tiling factors (see Fig. 2) as follows:

- **I_PXL**: the input volume to be placed in on-chip memory depends on whether the first or second convolution is being performed (see Fig. 2). Assuming one convolution is computed at a time, the memory size should be such to accommodate the larger of the two volumes, as shown in (4), where Din_DW accounts for the input data width.

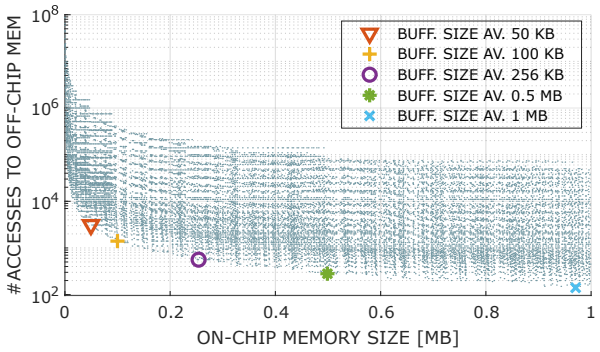
$$\begin{aligned} \mathrm{I\_PXL} &= Din\_DW \cdot \max\left\{V_A, V_B\right\} \\ V_A &= \mathrm{TWiA} \cdot \mathrm{THiA} \cdot \mathrm{Tq} \\ V_B &= \mathrm{THiB} \cdot \mathrm{TWiB} \cdot \mathrm{TpB} \end{aligned} \tag{4}$$

- **F_W**: the memory required to store the fixed weights $\mathbf{u}$ depends on the tiling factors and data width fw_DW:

$$\begin{aligned} \mathrm{F\_W} = \mathrm{Tr} \cdot \mathrm{Ts} \cdot \mathrm{Tq} \cdot \mathrm{TnA} \cdot \mathrm{TmA} \cdot \\ \cdot \mathrm{TpA} \cdot \mathrm{TL} \cdot fw\_DW \end{aligned} \tag{5}$$

- **SV_W**: as discussed in Sec. 3.2, due to the data dependencies introduced by the normalization, all the space-variant weights along dimensions W1 and K are computed and stored in the on-chip memory. The memory occupied by the weights $\mathbf{v}$ depends also on the data width svw_DW:

$$\mathrm{SV\_W} = \mathrm{TWo} \cdot \mathrm{THo} \cdot \mathrm{W1}^2 \cdot \mathrm{K} \cdot \mathrm{TL} \cdot svw\_DW \tag{6}$$



BUFF SIZE AV	π_acc	$\langle$THo, TWo$\rangle$	Tr,Ts,Tq, TmA,TnA, TpA,TpB	TL
50 KB	3.0E+03	$\langle 27, 20\rangle$	3	3
100 KB	1.4E+03	$\langle 43, 26\rangle$	3	3
256 KB	5.6E+02	$\langle 104, 14\rangle$	3	6
0.5 MB	2.8E+02	$\langle 262, 11\rangle$	3	6
1 MB	1.4E+02	$\langle 512, 11\rangle$	3	6

$\langle \text{Ho}, \text{Wo}, \text{K}, \text{W1}, \text{W2}, \text{L}\rangle = \langle 512, 512, 3, 3, 3, 6\rangle$
$(Din\_DW, fw\_DW, svw\_DW, Dout\_DW) = 8\ bit$

Fig. 3. **(left)** On-chip memory vs off-chip memory access for all the mappings explored by the framework and **(right)** details of the optimal mappings.

- **O_PXL**: the memory required by the output pixel is show in (7), where also the data width $Dout\_DW$ is taken into account and can be evaluated as:

$$\text{O\_PXL} = \text{TWo} \cdot \text{THo} \cdot \text{TL} \cdot Dout\_DW \tag{7}$$

3.4. Off-chip memory accesses

If the on-chip memory constraints discussed in Sec. 3.3 are satisfied, then the off-chip memory accesses can be estimated as follows. Since the space-variant weights only need to be stored on-chip, they do not contribute to off-chip memory accesses. Therefore, the total off-chip memory accesses depend only on: i) the input pixels for Conv1 (π_px_c1), ii) the fixed weights (π_fw), and iii) the input pixels for Conv2 (π_px_c2). In particular, π_px_c1 is minimized by scheduling loops 3A and 2A in Fig. 1 (a) as the outermost loops, maximizing the input pixels reuse. On the other hand, π_fw is minimized by scheduling loops 6, 5A, 4A, 2A, and 1A as the outermost.

Considering that the space-variant weights are stored on-chip only, π_px_c2 can be computed as

$$\begin{aligned}\pi\_px\_c2 = &\lceil \text{Ho}/\text{THo} \rceil \cdot \lceil \text{Wo}/\text{TWo} \rceil \cdot \\ &\cdot \lceil \text{K}/\text{TpB} \rceil \cdot \lceil \text{L}/\text{TL} \rceil\end{aligned} \tag{8}$$

4. EVALUATION AND RESULTS

In memory-bounded algorithms such as neural networks, the accesses to the off-chip memory risk to be the higher source of power consumption. Hence, in this case study off-chip memory access, estimated as

$$\pi\_acc = \pi\_px\_c1 + \pi\_fw + \pi\_px\_c2, \tag{9}$$

is used as a metric for the quality of the NLC mappings proposed by the framework.

The off-chip memory accesses depend not only on the mapping but also on the available on-chip memory, as it allows to store data close to the PEs, avoiding off-chip data movements. However, the available on-chip memory $BUFF\_SIZE\_AV$ is a design constraint of the FPGA that cannot be modified. Therefore, a mapping is optimal if:

$$\begin{aligned}&\textbf{min} \quad \pi\_acc \\ &\textbf{s.t.} \quad \sigma\_mem \leq BUFF\_SIZE\_AV.\end{aligned} \tag{10}$$

Fig. 3 shows the possible mappings explored by NLCMap for a given NLC layer configuration and for different values of available on-chip memory. The configuration of the layer ($\langle \text{Ho}, \text{Wo}, \text{K}, \text{W1}, \text{W2}, \text{L}\rangle = \langle 512, 512, 3, 3, 3, 6\rangle$) has been chosen considering the typical sizes of an image with three RGB channels K in input.

As expected and previously discussed, the off-chip memory accesses are inversely proportional to σ_mem. As it can be noted from right part of Fig. 3, only THo, TWo, and TL are considered as tunable parameters, whereas the other tiling factors are kept fixed. This choice is due to the consideration that the input pixels need to be used by two convolutions. It is therefore convenient to maximize their reuse at the expense of reducing the reuse of the fixed weights. As a consequence, if enough on-chip memory is available, THo, TWo, and TL can be increased, reducing the off-chip memory accesses. The points highlighted in different colors in the left part of Fig. 3 are the optimal mappings selected by NLCMap for different values of on-chip memory, they lie on the Pareto front of the σ_mem-π_acc curve, indeed.

5. CONCLUSION

In this paper we propose the first framework for the optimal mapping of NLC layers on generic hardware accelerators. The mapping space is first pruned by heuristic considerations, and then fully explored. Presently, the only metric considered for the choice is the number of accesses to the off-chip memory, which are often the bottleneck in data-centric systems. Other relevant metrics, such as latency and required DSP slices, can also be considered when choosing a mapping, and they will be included in the framework as a future work.

6. REFERENCES

[1] S. Marsi, J. Bhattacharya, R. Molina, and G. Ramponi, "A non-linear convolution network for image processing," *Electronics*, vol. 10, no. 2, 2021.

[2] C. Xie, X. Tian, R. Jiang, and Y. Chen, "Dilated kernel prediction network for single-image denoising," *Journal of Electronic Imaging*, vol. 30, no. 2, pp. 1 – 15, 2021.

[3] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "TETRIS: scalable and efficient neural network acceleration with 3D memory," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2017, Xi'an, China, April 8-12, 2017*, Y. Chen, O. Temam, and J. Carter, Eds. 2017, pp. 751–764, ACM.

[4] S. Venkataramani, J. Choi, V. Srinivasan, W. Wang, J. Zhang, M. Schaal, M. J. Serrano, K. Ishizaki, H. Inoue, E. Ogawa, M. Ohara, L. Chang, and K. Gopalakrishnan, "DeepTools: Compiler and execution runtime extensions for rapid AI accelerator," *IEEE Micro*, vol. 39, no. 5, pp. 102–111, 2019.

[5] Y. Shen, M. Ferdman, and P. A. Milder, "Maximizing CNN accelerator efficiency through resource partitioning," in *Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA 2017, Toronto, ON, Canada, June 24-28, 2017*. 2017, pp. 535–547, ACM.

[6] L. Song, J. Mao, Y. Zhuo, X. Qian, H. Li, and Y. Chen, "HyPar: Towards hybrid parallelism for deep learning accelerator array," in *25th IEEE International Symposium on High Performance Computer Architecture, HPCA 2019, Washington, DC, USA, February 16-20, 2019*. 2019, pp. 56–68, IEEE.

[7] S. Venkataramani, A. Ranjan, S. Banerjee, D. Das, S. Avancha, A. Jagannathan, A. Durg, D. Nagaraj, B. Kaul, P. Dubey, and A. Raghunathan, "ScaleDeep: A scalable compute architecture for learning and evaluating deep networks," in *Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA 2017, Toronto, ON, Canada, June 24-28, 2017*. 2017, pp. 13–26, ACM.

[8] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, February 22-24, 2015*, G. A. Constantinides and D. Chen, Eds. 2015, pp. 161–170, ACM.

[9] M. Song, J. Zhang, H. Chen, and T. Li, "Towards efficient microarchitectural design for accelerating unsupervised gan-based deep learning," in *IEEE International Symposium on High Performance Computer Architecture, HPCA 2018, Vienna, Austria, February 24-28, 2018*. 2018, pp. 66–77, IEEE Computer Society.

[10] A. Stoutchinin, F. Conti, and L. Benini, "Optimally scheduling CNN convolutions for efficient memory access," *CoRR*, vol. abs/1902.01492, 2019.

[11] W. Lu, G. Yan, J. Li, S. Gong, Y. Han, and X. Li, "FlexFlow: A flexible dataflow accelerator architecture for convolutional neural networks," in *2017 IEEE International Symposium on High Performance Computer Architecture, HPCA 2017, Austin, TX, USA, February 4-8, 2017*. 2017, pp. 553–564, IEEE Computer Society.

[12] X. Yang, M. Gao, Q. Liu, J. Setter, J. Pu, A. Nayak, S. Bell, K. Cao, H. Ha, P. Raina, C. Kozyrakis, and M. Horowitz, "Interstellar: Using Halide's scheduling language to analyze DNN accelerators," in *ASPLOS '20: Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, March 16-20, 2020*, J. R. Larus, L. Ceze, and K. Strauss, Eds. 2020, pp. 369–383, ACM.

[13] A. Parashar, P. Raina, Y. S. Shao, Y. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. S. Emer, "Timeloop: A systematic approach to DNN accelerator evaluation," in *IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2019, Madison, WI, USA, March 24-26, 2019*. 2019, pp. 304–315, IEEE.

[14] Y. S. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. R. Pinckney, P. Raina, S. G. Tell, Y. Zhang, W. J. Dally, J. S. Emer, C. T. Gray, B. Khailany, and S. W. Keckler, "Simba: Scaling deep-learning inference with multi-chip-module-based architecture," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2019, Columbus, OH, USA, October 12-16, 2019*. 2019, pp. 14–27, ACM.

[15] V. Sze, Y. Chen, T. Yang, and J. S. Emer, *Efficient Processing of Deep Neural Networks*, Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2020.

[16] M. Capra, B. Bussolino, A. Marchisio, G. Masera, M. Martina, and M. Shafique, "Hardware and software optimizations for accelerating deep neural networks: Survey of current trends, challenges, and the road ahead," *IEEE Access*, vol. 8, pp. 225134–225180, 2020.