FLECSim-SoC: A Flexible End-to-End Co-Design Simulation Framework for System on Chips

Tim Hotfilter, Julian Hoefer, Fabian Kreß, Fabian Kempf, Juergen Becker
Karlsruhe Institute of Technology
Karlsruhe, Germany
{hotfilter, julian.hoefer, fabian.kress, fabian.kempf, becker}@kit.edu

***Abstract*—Hardware accelerators for deep neural networks (DNNs) have established themselves over the past decade. Most developments have worked towards higher efficiency with an individual application in mind. This highlights the strong relationship between co-designing the accelerator together with the requirements of the application. Currently for a structured design flow, however, it lacks a tool to evaluate a DNN accelerator embedded in a System on Chip (SoC) platform.**

To address this gap in the state of the art, we introduce FLECSim, a tool framework that enables an end-to-end simulation of an SoC with dedicated accelerators, CPUs and memories. FLECSim offers flexible configuration of the system and straightforward integration of new accelerator models in both SystemC and RTL, which allows for early design verification. During the simulation, FLECSim provides metrics of the SoC, which can be used to explore the design space. Finally, we present the capabilities of FLECSim, perform an exemplary evaluation with a systolic array-based accelerator and explore the design parameters in terms of accelerator size, power and performance.

***Index Terms*—neural network simulation, neural network accelerator, simulation framework, SoC simulation**

I. Introduction

In the past decade, development and research in the area of deep neural networks (DNNs) gained pace by the order of magnitude. This trend is notably highlighted by emerging topics such as autonomous driving as well as by research in robotics, medicine and science. For instance, DNNs already surpass humans in filtering data much faster than traditional algorithms in particle physics experiments [1]. However, DNNs, in general, come with a very high computational complexity and large memory requirements. As the algorithms gain in complexity to enter more areas of application, these factors scale even further. Hence, many current general-purpose computer architectures like CPUs or GPUs cannot keep up with the demand for computational power.

As a result, a trend towards hardware accelerators for DNNs can be observed since the 2010s. This is further supported by the advances in silicon manufacturing, which allow for more integrated architectures. Recent accelerators employ optimization strategies of DNNs such as exploiting parallel data flow [2], sparsity in the intermediate results [3] or reoccurring data. Thereby, they can quickly achieve 100x in performance increase or over 1000x better energy efficiency [4]. However, the growing DNN size and additional constraints like real-time capabilities pose yet unanswered memory, bandwidth and performance challenges on the architectures.

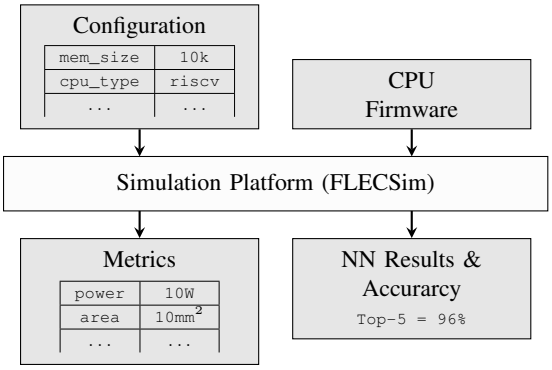


Fig. 1: Principle of FLECSim: Based on an architecture configuration the simulation is performed. The tool eventually reports metrics of the architecture and of the neural network.

Since it is impossible to achieve a high throughput accelerator with low power consumption and a small area footprint, current approaches have to take the application and the DNN topology into account from the beginning. While an embedded IoT accelerator emphasizes a low power consumption, a data-center accelerator rather focuses on high throughput. Thus a well-fitting accelerator for a given DNN can exploit optimizations to reduce power consumption or achieve higher performance. All requirements can as well be explored automatically through a proper framework.

In this paper, we present our design approach to bring together the variety of requirements that current applications pose on hardware accelerators for DNNs. Therefore, we introduce FLECSim, a simulation platform that allows us to evaluate an accelerator in a System on Chip (SoC) setup, with additional modules like memories or CPUs. As depicted in Figure 1 the SoC parameters can be configured in advance and after simulation, the tool reports valuable design insights of the overall architecture such as power consumption, chip area or execution time of each individual components as well as the results of the neural network itself.

In summary, our contributions are threefold:

- We present our simulation framework FLECSim and its versatile applications due to the full SoC simulation with interchangeable accelerator models. FLECSim allows for

an early design evaluation through the provided simulation insights and metrics such as an estimation of chip area, execution time or energy consumption.
- We show how FLECSim can guide the co-design process between an accelerator and the according application.
- We finally evaluate our simulation framework by exemplary means of a systolic array-based accelerator.

II. Motivation and Challenges

The development of an SoC requires the consideration of various mutually exclusive design decisions, which results in a crucial co-design of both hardware and software. In this process, each functionality is evaluated and afterwards either assigned to run in hardware or software. While pure software solutions offer the most flexibility and straightforward deployment, they lack energy efficiency and latency compared to dedicated hardware accelerators. Nevertheless, they have almost no flexibility and take more chip area than software implementations. The advantages of hardware accelerators come into play when parallelism and special operations might be exploited. Only the well-balanced combination of software and hardware solutions allows for an efficient SoC design.

AI-supported applications based on DNNs are becoming more prevalent and are finding their way into embedded systems, which leads to dedicated SoCs with a DNN accelerator to support the parallel data flow of the algorithms. Over the recent decade, many DNN accelerators were presented from a more general area of application such as the in-datacenter tensor processing unit (TPU) [5] to highly specific ones like Eyeriss [2], which focuses on an energy-aware inference of convolutional DNNs by leveraging sparsity in the data.

The design of the DNN accelerator strongly depends on the DNN topology to be executed. Larger or more complex computation units require more energy and chip area but promise either a faster execution or more precise results. To achieve high utilization of a DNN accelerator, it is essential to guarantee a steady supply of data from the memory. Since the data movement represents a typical bottleneck, the orchestration of the data flow has to be meticulously considered [6]. Hence, the shorter the data path is the more performance can be achieved. However, large chunks of local memory require much chip area and energy as well, which becomes a limiting factor in embedded solutions. As a consequence, the memory hierarchy has to be taken into account while dimensioning the accelerator. Moreover, the accelerator has to be configured and controlled from an application software running on a CPU. This metadata has a direct influence on the accelerator performance. For an accurate estimate of the overall system performance, the interactions between the DNN accelerator and software have to be considered as well.

As a consequence, the main challenge and motivation while designing an SoC for a given DNN application is to carefully balance the trade-offs between energy consumption, chip area and performance. To get credible metrics in these domains, it is important to consider and align the complete setup from the accelerator over the memory hierarchy to the CPU integration. This requires a simulation framework that can simulate the system for different configurations.

III. Related Work

There have been numerous research efforts done on hardware accelerators for DNNs [2]–[5], [7]. They explore different data flow and reuse strategies as well as various memory orchestrations based on a given application or network architecture. In the same period, many novel network architectures were presented. While some focus on higher prediction accuracy, there is a growing number of networks available that put effort into reducing the number of network weights and maintaining the accuracy. Systematic pruning techniques [8] e.g. yield network architectures such as SqeezeNet [9] or Tiny-Darknet [10]. Thereby, these can reduce the number of weights by 50x and the number of operations by 3x, respectively. To further reduce the memory footprint or the bandwidth requirements without accuracy degradation, quantization and compression techniques are widely employed.

These approaches highlight the strong relationship between the DNN topology and the underlying hardware accelerator. To investigate this relationship from a methodical point of view, concomitant simulation frameworks for DNN accelerators or highly adjustable hardware architectures were introduced. Approaching from a top-down view, ScaleSIM [11] offers fast estimation of an analytical model of a fixed systolic array supporting different data flows. While the tool reports utilization and allows for good insights, it lacks architectural design parameters such as chip area and energy consumption. Timeloop [12] takes the analytical DNN model further to find the best matching mapping for a DNN topology onto a systolic array. Moreover, it can be coupled with Accelergy [13], an open-source tool for generally applicable static energy estimation, to report energy consumption as well as chip area and thus to overcome the design parameter restriction. Similarly, the authors of STONNE [14] present a simulation framework that optimizes towards a high utilization by exemplary means of an accelerator of an MAERI [7] architecture.

The Co-Exploration of Hardware Design Space and Neural Architecture shows potential to optimize accuracy and hardware efficiency. In [15] and [16] the authors partition the execution of the neural architecture on multiple FPGAs. The mapping of layers on heterogeneous ASIC architectures is covered in [17]. In all three works, a feedback loop optimizes the neural architecture to maximize accuracy. Abdelfattah et al. [18] present an approach to co-design a DNN accelerator with high-level synthesis tools to obtain precise area and energy estimates. On a lower level, Reagan et al. present their fine-granular adjustable architecture Minerva [19] and explore various configurations to achieve an 8x better energy efficiency reported from EDA tools, while preserving the accuracy. However, its fine-granularity is limited to small applications.

An overall analytical SoC simulation and design flow is shown in SMAUG [20]. The authors present their approach starting from the network application through a dedicated scheduler down to their architecture, consisting of a CPU

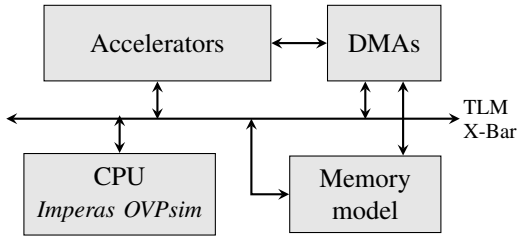


Fig. 2: Usual setup of our SystemC simulator inside FLECSim

model in gem5 [21] and a user-specific DNN accelerator model. The latter itself is interchangeable, however, it can only model the data flow taxonomy in an analytical way to fit their scheduler and thus cannot reveal control flow bottlenecks.

FLECSim addresses the current gap in the state of the art that lacks an end-to-end SoC simulation framework with a straightforward and interchangeable accelerator interface. In a similar way, we use an accurate architecture description in SystemC or RTL combined with a CPU model for data and control flow analysis. This is coupled with an analytical model for energy and area estimation to achieve a fast and highly flexible SoC simulation that offers an early system integration.

IV. FLECSim Simulation Platform

FLECSim allows us to simulate an SoC design with various components like accelerators, memories or CPUs that can communicate via a TLM crossbar. A usual setup is depicted in Figure 2. The tool allows the instantiation of accelerators for specific tasks e.g. neural networks but not limited to any application. Different data movement and orchestration schemes can be realized through dedicated configurable DMAs and memories. FLECSim features a CPU to simulate control flow aspects in an instruction set simulator (ISS). All other SoC modules are implemented in SystemC to enable cycle-accurate execution and simple extension in C++.

FLECSim and its components can be set up with a JSON-based configuration file, which can consider for example the memory size or the design parameters of the accelerator; factors that heavily impact the SoC. During the simulation, FLECSim collects valuable information, from which we obtain design metrics such as energy consumption or execution time.

A. CPU Platform and TLM Crossbar

To enable a full SoC simulation we have to consider data flow driven applications that can be accelerated in dedicated hardware structures and control flow aspects, which might be executed in a CPU. As an example, inside a systolic array, the incoming data is forwarded and processed through a mesh without using separate instructions for each step. However, setting up a DNN and the accelerator requires at least one master component, which controls the data flow from and to the accelerator and the memory, respectively. This can either be done within the accelerator itself, by adding a DMA to the SoC or by using a dedicated control flow entity such as a processor. An entire CPU offers the possibility to port various DNN implementations and frameworks such as Darknet, PyTorch or TensorFlow by cross-compiling to the target CPU architecture.

Since the main focus of this work is on the SoC architecture simulation and fast exploration of hardware accelerators the use of a cycle-accurate processor simulation can be avoided. Instead, we attach an instruction-accurate ISS to FLECSim which in turn leads to a significantly reduced simulation time. This approach also enables the user to run a DNN on a Linux kernel in the ISS which is supported by widely used simulators such as gem5 [21] or Imperas OVPsim [22].

Transmitting signals and values between the instruction-accurate ISS and the attached cycle-accurate simulated peripherals is realized by using the SystemC TLM-2.0 (Transaction Level Modeling) [23]. However, the ISS usually executes multiple instructions between the cycles according to the quantum and the set MIPS (million instructions per second) value. The quantum period thereby sets the time each processor model instance has to wait before it can continue. Using one of the core interfaces specified by SystemC TLM-2.0, i.e. the blocking transport or the direct memory interface, would therefore hold the ISS back from moving to the next instruction in case of a read access. In contrast, values sent via the TLM analysis port are immediately propagated and handled by corresponding functions. This approach leads to a considerable reduction of the overall simulation time since thousands of instructions can be simulated without an intervening SystemC scheduler.

In this work, we use the Imperas OVPsim ISS since it inherently implements the TLM analysis port as communication interface to our SystemC modules. Additionally, it offers a wide range of processor models and peripherals to simulate actual SoCs. Every module of the simulated SoC can be mapped to a configurable address space and connected to the internal CPU Platform system bus over a dedicated peripheral interface. FLECSim can therefore serve a broad range of applications and hardware platforms to guide the hardware and software co-design process.

B. Accelerators

The design of the FLECSim simulation platform allows for various accelerator architectures. These must only provide at least one port that is connected to one of the SoC modules or the CPU. The width of the port is not fixed hence it is customizable to model any kind of accelerator. Also, the number of ports to the DMA engine and thus to the memory is not limited, which allows us to simulate SoCs with an arbitrary number of DMAs. A connection to the processor can be established via the TLM crossbar to access an internal set of registers from the outside to e.g. read the current state or configure internal parameters of the accelerator during run time. Thus, FLECSim comes with a common TLM interface for the accelerator models.

Besides a pure SystemC accelerator model, FLECSim offers also a simulation of RTL models written in Verilog or SystemVerilog. Therefore, FLECSim provides a wrapper to Verilator, an open-source tool that compiles Verilog or

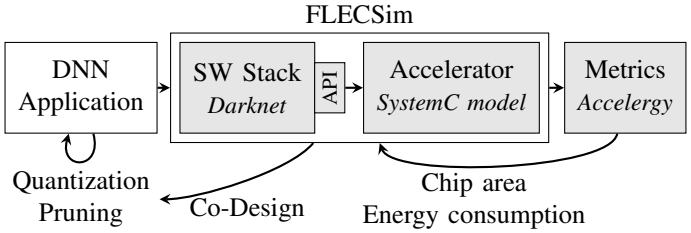


Fig. 3: Design flow of the DNN application and hardware accelerator co-design realized in FLECSim. The gray boxes are covered in this work.

SystemVerilog RTL code into SystemC [24]. Consequently, this approach eliminates the need to translate the RTL code into a SystemC model manually and since the RTL simulator model can be validated in the SoC in an early design stage, it speeds up the entire design process.

C. Memory model and stream data movement

A holistic simulation of the SoC has numerous advantages. Especially an integrated memory model allows for a more precise performance evaluation. Accordingly, we implemented a simple DRAM-based memory model which offers a configurable size only limited by the system architecture. To be able to offload algorithm parts from the CPU to the accelerator, the process can access the TLM memory via the TLM crossbar. The data flow between the memory and the accelerators can be set up by one or more DMAs. With a set of registers accessible by the CPU, these can be configured in terms of the address range or the access pattern to model various application scenarios. Additionally, the transfers can be triggered and the status retrieved from the same place.

V. Design Flow

FLECSim enables an end-to-end simulation of an SoC including e.g. a DNN accelerator. The flow of a DNN hardware accelerator design and how FLECSim can support this is depicted in Figure 3. In the first place, the DNN topology might be optimized to reduce the complexity through quantization or pruning. We can then deploy the network topology on the FLECSim CPU and control the accelerator structure via an API. FLECSim collects information about the energy consumption, execution time and chip area, which can be used to optimize the hardware architecture and DNN application.

A. CPU Software Framework

A crucial bottleneck during SoC design is the mutual dependency of hardware architecture, API and software, which leads to an increased development time and long time to market. The optimal solution is the parallel and iterative design to enable early validation and make faster design decisions. FLECSim enables this through high flexibility in the choice of the software architecture. The application can be compiled for the target CPU instruction set and subsequently executed by the CPU ISS. The application spectrum ranges from simple C programs to software frameworks such as Darknet or TensorFlow to operating system (OS) kernels e.g. Linux. In this manner to provide a wide range of state-of-the-art DNN applications, we adapt the well-established DNN framework Darknet [25] to be executed as bare-metal directly on the CPU. The Darknet framework offers easy access to the underlying BLAS functions and the general matrix multiplication (GEMM) implementation, which accounts for the vast majority of computations in DNNs. Hence, these functions can be adapted according to the corresponding custom API such that the underlying accelerator is configured and supplied with data to further perform the very same functionality. This allows us to run arbitrary DNN topologies by providing the corresponding Darknet configuration files. As a result, early estimates about the overall execution time and efficiency of software, API and hardware accelerator are enabled, addressing the aforementioned bottleneck and allowing early parallel and iterative optimization.

B. Design Metrics

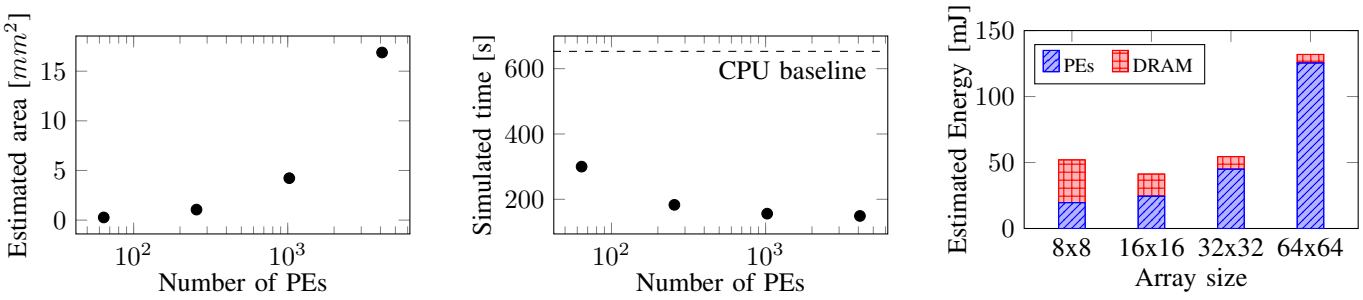
Along with the result of the DNN, FLECSim outputs various platform metrics. Gathering metrics is a crucial indicator to support designers in the hardware and software co-design. While the result of the DNN can be used to compute the accuracy, the collected metrics give insights to optimize the application in terms of its efficiency. This can be done, for example, by reducing the number of idle cycles for each accelerator and revealing bottlenecks in the SoC architecture.

OVPsim already provides various metrics such as the number of executed instructions and the overall simulated time. It also offers traces to capture each executed instruction. Beyond the ISS, the number of register accesses from the processor is counted within the TLM interface. Apart from the CPU, SystemC modules inside FLECSim can be annotated with actions such that the tool creates a JSON action counts file. Since all modules except the CPU are simulated in a cycle-accurate manner, the status of each can be logged for every single clock cycle. This reveals different run and idle states, from which the energy consumption can be derived.

To estimate the area of the SoC modules and their respective energy consumption, we integrated Accelergy [13] into our tool flow. Dynamic energy and area estimation of Accelergy relies on a primitive component library based on estimation plug-ins. We use CACTI [26] for memory components and Aladdin [27] for accelerator structures. From gathered action counts, the module attributes and architecture descriptions Accelergy generates the estimates. The results can be combined with metrics from the ISS and the TLM interface to calculate the overall power consumption of the SoC architecture.

VI. Experimental results

To prove our concept, we designed a SystemC model of a dedicated AI accelerator based on widely used architectures with a systolic array of processing elements (PE). The size of the accelerator, respectively the number of PEs, is configurable. We evaluate the effects of different accelerator sizes in



(a) Area estimation of the accelerator with different dimensions

(b) Simulated execution time on the SoC depending on accelerator dimensions

(c) Estimated dynamic energy consumption with different array sizes broken down into DRAM and accelerator consumption

Fig. 4: Comparison of Tiny-Darknet inference on an exemplary SoC for different accelerator sizes

an exemplary SoC, consisting of a 32-bit RISC-V processor model, a TLM-Memory and a SystemC DMA model. The energy and area estimates of the accelerator are based on a $40nm$ reference table, for our memory model, a $65nm$ library is used, respectively.

A. Neural Network Application

Small DNNs offer good accuracy results while only having a small memory footprint and are likely implemented in SoCs for mobile devices. To simulate a realistic application in our SoC, we selected Tiny-Darknet [10], a DNN classifier, that is hosted in the Darknet framework. Tiny-Darknet consists of 16 convolutional layers, with around 0.98 billion operations in total and a memory footprint of 4 MB. We simulate the entire execution, namely weight and image loading, feature extraction and classification. Thereby, we work with standard floating-point precision (32-bit). The vast majority of computations are caused by the convolutional layers, which will be offloaded to the accelerator. To prove correctness, we work with pre-trained weight data, allowing us to evaluate the classification accuracy with potential architectural optimizations such as dynamic input feature map pruning.

B. Accelerator Structure

Since the Darknet framework uses the im2col transformation for simpler memory access patterns, the convolutional layers can be represented as GEMM. The accelerator performs this matrix multiplication in an output stationary manner. Partial results of the inner products are kept local inside the individual PEs. Weight and input feature map data are distributed among the PEs. Weight data is handed to a whole row of PEs and input feature map data to a column accordingly. To match the accelerator dimensions, the matrices are divided into multiple chunks that are streamed from the TLM memory via the DMAs to the accelerator. The DMAs can be configured from the CPU platform by setting control registers to perform the necessary address generation depending on matrix dimensions, accelerator array size and datatype used. Per cycle, each DMA reads a line of data corresponding to the accelerator rows or columns and calculates the next address depending on the matrix dimension. Communication between DMAs and the accelerator is modeled in a similar way to the well-known AXI Stream protocol.

C. Evaluation and Design Insights

With the accelerator and SoC architecture described above, we perform an estimation of the area requirements of the modules as well as the execution time and energy consumption of Tiny-Darknet. We define the execution time as the time of the last trace file entry, meaning that data loading and classification by the CPU and feature extraction by the accelerator have finished and thus taken into consideration. The RISC-V processor model is configured to perform 40 MIPS, comparable to other low-power processors used in mobile devices. To simulate the accelerator and other modules accurately, we choose a quantum duration of $1\mu s$ according to the Imperas OVPsim guidelines [22]. For error avoidance reasons, the simulation clock cycle duration of the SystemC-modules must be higher than the quantum duration of $1\mu s$. Therefore, we set the clock cycle of the SoC processor peripherals to $10\mu s$.

Figure 4 shows the evaluation results of our test case. We explored four different accelerator sizes (8x8, 16x16, 32x32, 64x64), revealing the trade-offs between execution time, energy consumption and chip area. As expected, the area of the accelerator depicted in Figure 4a increases with the number of PEs. The high area demands for the large accelerator arrays are reinforced by the usage of 32-bit floating-point precision simulated for the PEs, surpassing the demands of e.g. 8-bit integer multiply-accumulate units.

When looking at the simulated execution time of the network (Figure 4b) of merely the CPU, a comparably high inference time of $653s$ can be observed, which is due to the simulation of the 40 MIPS low-power CPU. Even with a small 8x8 systolic array accelerator, the simulation reveals significant potential for execution time improvements. With an increasing number of processing elements and exploiting higher spatial parallelization this continues. Due to inherent data reuse inside the systolic array, fewer load operations from the memory are necessary. However, even with the four sample accelerators, attenuation can be observed. Parts of the DNN

inference that cannot be addressed by the hardware accelerator such as image and weight loading or im2col transformation are executed exclusively on the CPU.

The drawback of employing a high spatial parallelization is an increased energy demand that can be inferred from Figure 4c. As the number of PEs and likewise the data reuse increases, the demand for memory accesses shrinks leading to reduced energy consumption from DRAM reads. However, this cannot compensate for the energy demands of the PEs when looking at the 64x64 array.

In general, from the SoC perspective, a good solution for the Tiny-Darknet inference is the combination of CPU and a dedicated accelerator of size 16x16 or 32x32, having a reasonable trade-off between reduced execution time and additional area and energy demand.

VII. Conclusions and Future Work

In this paper, we introduced FLECSim an end-to-end SoC simulation framework that allows for the crucial co-design of DNN applications and underlying hardware accelerator. The advantages of FLECSim come to play when an overall SoC simulation with a straight-forward, flexible as well as early integration of new accelerator models and interchangeable software stack is favored. In the scope of this work, we demonstrate how FLECSim can support the design flow with the provided design insights by exemplary means of a systolic array accelerator in combination with a RISC-V processor running the Darknet DNN framework.

In the future, we would like to underpin our findings and metrics with an evaluation of the SoC design on an actual reconfigurable SoC platform such as the Xilinx Zynq. Besides this, we plan to extend FLECSim to allow a full and automated design space exploration aided by reinforcement learning. This might also involve network optimization through pruning or quantization and thus includes the complete design flow.

Acknowledgment

This work was funded by the German Federal Ministry of Education and Research (BMBF) under grant number 16ME0096 (ZuSE-KI-mobil). The responsibility for the content of this publication lies with the authors.

References

[1] S. Baehr, F. Kempf, and J. Becker, "Data reduction and readout triggering in particle physics experiments using neural networks on fpgas," in *2018 IEEE 18th International Conference on Nanotechnology (IEEE-NANO)*, 2018, pp. 1–4.

[2] Y. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

[3] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 1–13.

[4] S. Han *et al.*, "EIE: efficient inference engine on compressed deep neural network," *CoRR*, vol. abs/1602.01528, 2016. [Online]. Available: http://arxiv.org/abs/1602.01528

[5] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," *CoRR*, vol. abs/1704.04760, 2017. [Online]. Available: http://arxiv.org/abs/1704.04760

[6] M. Pellauer *et al.*, "Buffets: An efficient and composable storage idiom for explicit decoupled data orchestration," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 137–151. [Online]. Available: https://doi.org/10.1145/3297858.3304025

[7] H. Kwon, A. Samajdar, and T. Krishna, "Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 461–475. [Online]. Available: https://doi.org/10.1145/3173162.3173176

[8] T. Zhang, S. Ye, K. Zhang, J. Tang, W. Wen, M. Fardad, and Y. Wang, "A systematic dnn weight pruning framework using alternating direction method of multipliers," *arXiv preprint arXiv:1804.03294*, 2018.

[9] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size," *CoRR*, vol. abs/1602.07360, 2016. [Online]. Available: http://arxiv.org/abs/1602.07360

[10] "Tiny darknet," https://pjreddie.com/darknet/tiny-darknet/, accessed: 2021-04-12.

[11] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "Scale-sim: Systolic cnn accelerator simulator," *arXiv preprint arXiv:1811.02883*, 2018.

[12] A. Parashar *et al.*, "Timeloop: A systematic approach to dnn accelerator evaluation," in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2019, pp. 304–315.

[13] Y. N. Wu, J. S. Emer, and V. Sze, "Accelergy: An architecture-level energy estimation methodology for accelerator designs," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–8.

[14] F. Muñoz-Martínez, J. L. Abellán, M. Acacio, and T. Krishna, "Stonne: A detailed architectural simulator for flexible neural network accelerators," *ArXiv*, vol. abs/2006.07137, 2020.

[15] L. Yang *et al.*, "Co-exploring neural architecture and network-on-chip design for real-time artificial intelligence," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2020, pp. 85–90.

[16] W. Jiang *et al.*, "Hardware/software co-exploration of neural architectures," 2020.

[17] L. Yang *et al.*, "Co-exploration of neural architectures and heterogeneous asic accelerator designs targeting multiple tasks," in *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference*, ser. DAC '20. IEEE Press, 2020.

[18] M. S. Abdelfattah, Ł. Dudziak, T. Chau, R. Lee, H. Kim, and N. D. Lane, "Best of both worlds: Automl codesign of a cnn and its hardware accelerator," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[19] B. Reagen *et al.*, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 267–278.

[20] S. L. Xi, Y. Yao, K. Bhardwaj, P. Whatmough, G.-Y. Wei, and D. Brooks, "Smaug: End-to-end full-stack simulation infrastructure for deep learning workloads," *ACM Trans. Archit. Code Optim.*, vol. 17, no. 4, Nov. 2020. [Online]. Available: https://doi.org/10.1145/3424669

[21] N. Binkert *et al.*, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, p. 1–7, Aug. 2011. [Online]. Available: https://doi.org/10.1145/2024716.2024718

[22] "Imperas OVPsim," https://ovpworld.org, accessed: 2021-04-05.

[23] "OSCI TLM-2.0 standard," http://systemc.org, accessed: 2021-04-07.

[24] W. Snyder, "Verilator and systemperl," *North American SystemC Users' Group Design Automation Conference*, 2004.

[25] J. Redmon, "Darknet: Open source neural networks in C," http://pjreddie.com/darknet/, 2013–2016.

[26] S. Li, K. Chen, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "Cacti-p: Architecture-level modeling for sram-based structures with advanced leakage reduction techniques," in *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2011, pp. 694–701.

[27] Y. S. Shao, B. Reagen, G. Wei, and D. Brooks, "Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures," in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, 2014, pp. 97–108.