An Analytical Model of Configurable Systolic Arrays to find the Best-Fitting Accelerator for a given DNN Workload

Tim Hotfilter, Patrick Schmidt, Julian Hoefer, Fabian Kreß, Tanja Harbaum, Juergen Becker
{hotfilter,patrick.schmidt2,julian.hoefer,fabian.kress,harbaum,becker}@kit.edu
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany

ABSTRACT

Since their breakthrough, complexity of Deep Neural Networks (DNNs) is rising steadily. As a result, accelerators for DNNs are now used in many domains. However, designing and configuring an accelerator that meets the requirements of a given application perfectly is a challenging task. In this paper, we therefore present our approach to support the accelerator design process. With an analytical model of a systolic array we can estimate performance, energy consumption and area for each design option. To determine these metrics, usually a cycle accurate simulation is performed, which is a time-consuming task. Hence, the design space has to be restricted heavily. Analytical modelling, however, allows for fast evaluation of a design using a mathematical abstraction of the accelerator. For DNNs, this works especially well since the dataflow and memory accesses have high regularity. To show the correctness of our model, we perform an exemplary realization with the state-of-the-art systolic array generator Gemmini and compare it with a cycle accurate simulation and state-of-the-art modelling tools, showing less than 1% deviation. We also conducted a design space exploration, showing the analytical model's capabilities to support an accelerator design. In a case study on ResNet-34, we can demonstrate that our model and DSE tool reduces the time to find the best-fitting solution by four or two orders of magnitude compared to a cycle-accurate simulation or state-of-the-art modelling tools, respectively.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; *Modeling and simulation*; • **Computer systems organization** → *Systolic arrays*; *Embedded systems.*

KEYWORDS

Analytical Modelling, Neural Networks, Design Space Exploration

ACM Reference Format:
Tim Hotfilter, Patrick Schmidt, Julian Hoefer, Fabian Kreß, Tanja Harbaum, Juergen Becker. 2023. An Analytical Model of Configurable Systolic Arrays to find the Best-Fitting Accelerator for a given DNN Workload. In *Proceedings of the 2023 Workshop on System Engineering for constrained embedded systems (RAPIDO 2023), January 17–18, 2023, Toulouse, France.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3579170.3579258

RAPIDO 2023, January 17–18, 2023, Toulouse, France

ACM ISBN 979-8-4007-0045-3/23/01.
https://doi.org/10.1145/3579170.3579258

1 INTRODUCTION

Deep Neural Networks (DNNs) entered more and more domains and areas over the last decade due to their higher prediction performance compared to traditional algorithms. In image recognition, for example, face recognition is used in assistive robotics to support the elderly [12] or in particle physics they support the compression of large datastreams [2]. While DNNs already show great performance in many tasks, over time their computational complexity and memory requirements grew rapidly to fulfill even more sophisticated tasks. Especially considering yet unsolved problems such as autonomous driving, the complexity is foreseen to grow even further. This trend poses a challenge to the underlying hardware architecture, which executes the DNN. Since the computation of DNNs is a highly dataflow driven and memory bound task, traditional computing devices like CPUs or GPUs cannot keep pace with the fast rising demands. To address this challenge, dedicated DNN accelerator architectures, like systolic arrays, are currently state of the art. Those DNN accelerators can compute operations in parallel and reuse data to achieve a high performance and efficiency. In addition, accelerators can incorporate optimization techniques like pruning or quantization [6].

While DNN accelerators can support fast and efficient inference, the design parameters of such an accelerator have to be evaluated carefully. The level of exploited parallelism or the number of processing elements and the size of memories and their interfaces, have a strong impact on various design metrics like throughput, latency, power consumption or area. Besides the architecture parameters, the DNN workload itself has a strong dependency with the performance, since proper mapping of the workload is also important. All these parameters open a large design space from which one solution has to carefully picked to reach high performance and efficiency. However, due to the complexity of DNN accelerators, determining the hyperparameters and specifications for a given accelerator configuration is costly, since each configuration has to be elaborated individually. Highly accurate results can be achieved though cycle-accurate simulation of the whole workload [8, 10], which takes a long time for each iteration. Considering the large design space, cycle-accurate simulation is not feasible for an extensive design space evaluation.

In this paper, we therefore present our analytical model of systolic arrays, which are a very common type of DNN accelerator. The analytical model is the centerpiece of our evaluation tool, shown in Figure 1. Our analytical model estimates performance, area, and energy consumption for a given design configuration and DNN workload in a fast and accurate way. Therefore, our approach uses the well-established roofline model for performance estimation and bottleneck identification. During the design process, constraints,

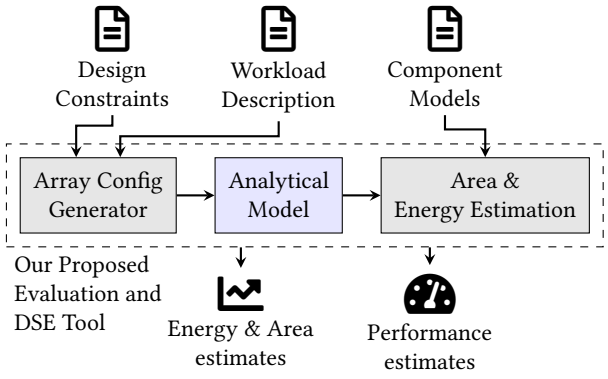


Figure 1: Overview of our DNN accelerator analysis tool with our analytical model of a systolic array at its center

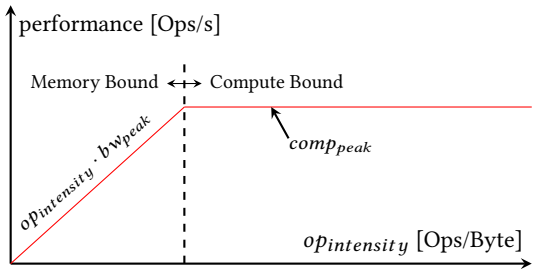


Figure 2: Roofline model showing peak computational and memory performance [13]

for example, an upper area or power limit, can be defined. This allows us to find a solution, which meets all design requirements. With our analytical model, we are able to evaluate a design in up to 12000x less time compared to a cycle-accurate simulation. We verify found solutions by comparing them against the cycle-accurate simulation, showing less than 1% deviation using a 16×16 systolic array. In a case study on ResNet-34, we use our evaluation tool for a design space exploration (DSE), to show its capabilities in finding an optimal DNN accelerator for this workload.

2 RELATED WORK

Over the last decade, various DNN accelerators were presented and have established themselves. One very prominent accelerator is Eyeriss by Chen et al. [4]. It implements a 12×14 array of compute elements, each equipped with a small memory to buffer inputs and weights. Their row-stationary dataflow allows for an efficient inference by minimizing the data movement to the main memory. However, Eyeriss has a fixed architecture and cannot be scaled for different performance requirements. SIMBA [11] is a chiplet accelerator made from multiple processing elements (PEs). In contrast to Eyriss, the architecture can be scaled and configured towards the different performance requirements of the DNN. However, both Eyriss and SIMBA are standalone chips and cannot be integrated into a System-on-Chip (SoC) for full flexibility. The systolic array generator Gemmini by Genc et al. [5] allows generating a fully flexible DNN accelerator design, which can be integrated into a SoC design. Within the Chipyard [1] project, Gemmini can be coupled with a RISC-V processor. Besides the flexible hardware architecture, Gemmini also offers a rich software stack and is compatible with common DNN frameworks.

As stated before, the choice of design parameters for complex DNN accelerators is a yet unsolved challenge. Therefore, some research on modelling these accelerators has been carried out. Timeloop [9] is a flexible tool, capable of performing analytical simulation for a wide range of architectures, which can be modelled through a set of primitives. Additionally, it allows defining the mapspace, i.e., how workloads can be mapped to the accelerator, and supports finding optimal mappings. To estimate performance and energy, Timeloop exploits the regularity of DNN workloads to analytically calculate action counts of the various system components. However, Timeloop limits itself to convolutional and fully-connected layers while neglecting other operations, like activations and pooling layers. Further, describing the mapspace requires prior knowledge of the dataflow the targeted accelerator uses, and hence it is difficult to generalize it for all systolic arrays. In contrast, ScaleSim [10] reduces the complexity by limiting itself to the simulation of only systolic arrays. The memory hierarchy inside ScaleSim consists of two input memories for input and weight data, and a separate memory to store results. All common dataflows for systolic arrays, weight stationary, output stationary and input stationary are supported. Mapping of complex problems onto the compute array is determined automatically. However, ScaleSim performs a cycle accurate simulation, which leads to a very high simulation time. In addition, ScaleSim lacks support for pooling operations and batched data. To get a very accurate simulation, Chen et al. [3] propose a custom simulation model that reflects the underlying hardware directly. Their model includes various aspects of the accelerator, such as the number of PEs, their arrangement in the array, mapping and available bandwidth. When all influences of these parameters are understood, it is possible to analytically determine the performance of the system. The downside of this approach, is that such models will only work for the specific accelerator they are designed for. The benefit of these models, however, is their close relation to the used hardware. Unlike the previously mentioned simulators, no additional modelling of the accelerator is necessary, as all relevant information can be extracted from the software layer.

3 CONCEPT OF OUR ANALYTICAL MODEL

Our analytical model enables fast and systematic exploration of a systolic array to find the best-fitting solution in the vast design space. As stated before, crucial design parameters like buffer sizes, the number of PEs or the interface bandwidths can have a very strong impact on the performance, energy efficiency and chip area consumption of the DNN accelerator. Hence, our model has to deliver fast and accurate estimates of the accelerator characteristics. Therefore, we base our model on the well-established roofline model introduced by Williams et al. [13] and use Accelergy [14] for area and energy estimation. Those tools allow us to design a highly abstracted model of the underlying hardware architecture. In general, the roofline model as shown in Figure 2 can be applied to all computational tasks. With the roofline model, we can calculate

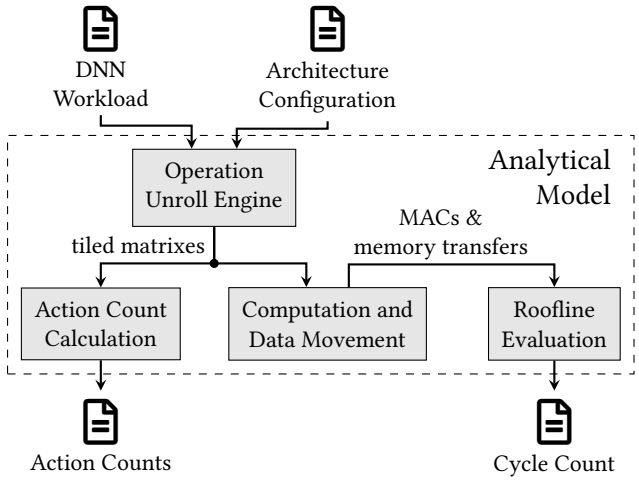


Figure 3: Overview of our analytical model

peak operational performance ($comp_{peak}$) and peak memory performance ($op_{intensity} \cdot bw_{peak}$), as well as the operational intensity. The horizontal roof gives the maximum computational performance, while the diagonal line gives the peak memory performance.

3.1 Analytical Modelling of a Systolic Array

The components and computation steps of our proposed analytical model are shown in Figure 3. To compute cycle and actions counts of a DNN inference, our model takes two inputs: A DNN workload description, that holds the layer shapes, and an architecture description that features, e.g., the array and memory sizes. Those inputs are evaluated in the *Operation Unroll Engine*, which splits large matrix operations into smaller tiles that match the underlying systolic array size. Based on the tiles, the *Action Count Calculation* module generates action counts for Accelergy's energy estimation. They represent how often an action is performed by a component, e.g., the number of memory accesses.

The *Computation and Data Movement* module takes the same outputs from the Operation Unroll Engine and estimates the number of MAC operations performed and the amount of data moved over the bus. MAC operations account for all compute cycles. Their number can be derived from a matrix-matrix multiplication between an $l \times m$ input matrix A and the $m \times n$ weight matrix B in the systolic array. In total, this accounts for $l \cdot m \cdot n$ MAC operations. However, we cannot map an arbitrary matrix onto a systolic array directly, but we have to account for mapping fragmentation effects as already defined by Chen et al. [3]. For example, *spatial mapping fragmentation* occurs, when the dimensions of the matrices A or B are smaller than the size of the systolic array. In this case, we have to pad the matrix such that it fits the array size. This is done through the scaling factor η. For example, a 10×10 matrix multiplication on a 16×16 array gives $\eta = 16/10 = 1.6$. *Temporal mapping fragmentation* can occur, when one input matrix is larger than the array. For example, a 24×16 matrix computed the same 16×16 array, leaves after one full iteration eight columns. Thus, in the second pass the array is only 50% utilized, resulting in an overall utilization of 75%. To consider this effect, which we use a scaling factor δ. In the example $\delta = \frac{\lceil 24/16 \rceil \cdot 16}{24} = \frac{4}{3}$. We can then describe the number of scaled MAC operations in each computation with Equation 1.

$$macs_{scale} = l \cdot m \cdot n \cdot \eta \cdot \delta \tag{1}$$

For an accurate modelling of the performance, it is also important to consider the data movement, as DNN inference is a very memory intense task. It is strongly dependent on the bus bandwidth. In general, data movement occurs in the form of block transfers between external and on-chip memories. We can view these data blocks as matrices of size $row \times col$ which are transferred row-wise. Similar to the systolic array matrices, we have to pad bus transfers. For example, even if only 1 B is transferred over a bus, it still effectively blocks the full bus-width. As such, we scale every single transfer to match the maximum bus-width bw_{peak}. Individual rows of the data blocks are split up into multiple transfers if they are larger than the bus-width. Additionally, we have to account for idle periods on the bus when data movement and computations do not overlap. This leads to additional overhead, which we account for through a scaling factor α. The calculation of the number of bus transfers N_{bw} and the scaled data moved are given in Equation 2.

$$\begin{aligned} N_{bw} &= \left\lceil \frac{col}{bw_{peak}} \right\rceil \cdot row \\ data_{scale} &= N_{bw} \cdot bw \cdot \alpha \end{aligned} \tag{2}$$

Based on the adjusted data movements and MAC operations, we can apply the roofline model to determine the cycle count for a given DNN workload on the systolic array. This evaluation happens in the *Roofline Evaluation* model by applying Equation 3. Besides the data movements and MAC operations, we also need to take architectural constraints into account. This is done through $comp_{peak}$ and bw_{peak}. They represent the number of MAC units in the systolic array and the available bandwidth. With all these four variables, we can compute the performance, from which we can calculate the cycle count.

$$\begin{aligned} op_{intensity} &= \frac{macs_{scale}}{data_{scale}} \\ performance &= \min\left(op_{intensity} \cdot bw_{peak}, comp_{peak}\right) \\ cycles &= \frac{macs_{scale}}{performance} \end{aligned} \tag{3}$$

3.2 Estimation of Energy and Area

To estimate the energy and area of the systolic array, we need action counts derived from our model. Action counts include information about which module has performed which operation and how often. The collected action counts are evaluated by Accelergy [14] to estimate the energy consumption and area. Accelergy provides a set of primitives, e.g., MAC units and memories, from which more complex architectures can be modelled. In general, a systolic array can be modelled as a $DIM \times DIM$ array of MAC units and registers. The on-chip memory can be modelled as banked SRAM.

To get the number of performed MAC operations, we look at Equation 1. For the number of memory accesses, we have to take the memory organization into account. As such, this can differ

between different architectures. We will discuss the calculation of their number for an example architecture in section 4.

3.3 Design Space Exploration

The main objective of our model is to speed up the design process. To enable automatic exploration of valid designs, we first have to put all possible design parameters options in the architecture description. Some parameters might be fixed like the layout of the on-chip memories, the number of memory banks and rows per bank. The user can limit the valid design space through a set of *Design Space Constraints*, for example, a maximum size of the systolic array and maximum sizes of the on-chip memories can be specified. Based on these inputs, our array configuration generator will automatically construct the design space and generate valid architecture descriptions that adhere to all constraints of the accelerator. Our analytical model then evaluates each architecture description for the given DNN workload layer-by-layer and emits action and cycle counts. Based on the action counts, we can estimate energy consumption and area using Accelergy. After the DNN workload has been evaluated on all generated architectures, we can post-process and analyze the results. The first step checks found solutions for constraint violations, like a too large area, and removes them. Next, the Pareto front and the global optimum are determined. The Pareto front is evaluated with regard to area, power, and performance, while the global optimum depends on a user-given target function, that for example minimizes energy. Finally, all results are visualized and stored. Results can also be imported for later evaluation with a different set of constraints.

4 USING GEMMINI AS AN EXEMPLARY SYSTOLIC ARRAY

To demonstrate and evaluate our proposed analytical model, we selected Gemmini [5], which is an open-source systolic array generator. Gemmini offers a high degree of freedom in its design parameters and supports a wide range of DNN workloads. It is integrated into the Chipyard framework [1]. The architecture consists of a scratchpad to store operands, an accumulator memory in which results are stored and the systolic array performing the computations. To integrate Gemmini into our systolic array analytical model, we have to adjust the operation unroll engine to match Gemmini's tiling and account for Max-Pooling, which is performed during write-backs to the main memory. While Gemmini is very flexible, there are, however, some architecture constraints we have to consider. Most important, due to the address generation, the size of the systolic array and the number of rows in each memory has to be a power of two. In the following, we denote Gemmini's systolic array size as $DIM \times DIM$. Additionally, the available bandwidth is fixed to 128 bit per transfer. Hence, we define $peak_{comp} = DIM \cdot DIM$ and $bw_{peak} = 128\,bit$. To model energy, we added models of the scratchpad, accumulator and the systolic array to Accelergy.

To calculate the number of MAC operations and data movements, we analyze all individual instructions that Gemmini can execute. This enables us to model the performance of the DNN inference. Taking each instruction into account, we can also derive action counts of each component, which allows us to estimate the energy consumption.

For the number of MAC operations, we evaluate the two compute instructions: `compute_preload` and `compute_accumulate`. The number of MAC operations follows the considerations from Equation 1. To account for spatial fragmentation, we define the scaling factor η for m and n so that they match the according array dimension DIM. This way, we can model each of these instructions blocking the full systolic array. For temporal fragmentation, in the `compute_accumulate` instruction, the scaling factor is set to one as this effect does not occur. For `compute_preload`, we have to scale l to match DIM as no more calculations are performed after l cycles, but the next computation cannot begin. To analyze the data movement, we can utilize Equation 2 to get Equation 4.

$$\begin{aligned} \eta &= \frac{DIM}{m} \cdot \frac{DIM}{n} \\ \delta_{preload} &= \frac{DIM}{l} \\ \delta_{accumulate} &= 1 \\ macs_{scale,preload} &= l \cdot m \cdot n \cdot \eta \cdot \delta_{preload} \\ macs_{scale,accumulate} &= l \cdot m \cdot n \cdot \eta \cdot \delta_{accumulate} \end{aligned} \tag{4}$$

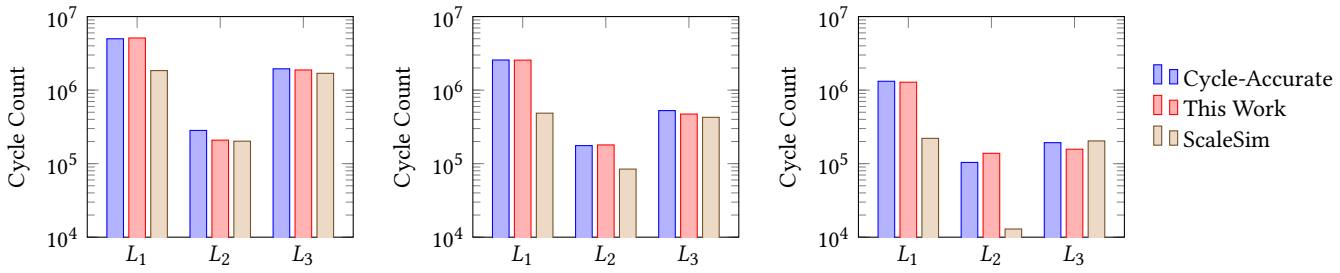
Besides the performance metrics, we also want to estimate the energy consumption and thus need action counts. For the number of MAC operations, we can apply Equation 1 as it was discussed previously. To determine the number of memory accesses requires knowledge of the memory organization. For the transfer of a $row \times col$ block of data, a total of $\lceil \frac{col}{DIM} \rceil \cdot row$ memory accesses are performed. To match the behavior of Gemmini, we integrate these formulas into the roofline evaluation model of our analytical model.

5 EVALUATION

Our analytical model is implemented in Python to allow for straightforward integration into common DNN frameworks like PyTorch. As workload, for all experiments that we evaluate, we choose ResNet [7], since it is a well-established CNN and features a wide range of different kernel sizes. The network's input size is 3×224×224, and we set the batch size to one. The results of our model are compared with a cycle accurate simulation, which is provided by the Chipyard framework [1], and with the state-of-the-art CNN accelerator simulator ScaleSim [10]. Since we target energy and area constrained applications like embedded systems, we picked 8×8, 16×16 and 32×32 as systolic array sizes. For area estimation, all components for Accelergy are assumed to be implemented in a 40 nm technology node. All experiments with our model are performed on a single AMD EPYC 7702P core running Rocky Linux, multiple cores can be used to run individual experiments in parallel.

5.1 Estimation Accuracy and Simulation Time

The results of the accuracy and runtime evaluation can be found in Figure 4 and Table 1, respectively. The plot depicts the estimated cycle count for the three different array sizes. For each size, the first three ResNet layers, representing all kernel sizes occuring in a ResNet, are shown. Each experiment is performed using our model and ScaleSim. In addition, a cycle-accurate simulation of Gemmini serves as a cycle count reference. The first layer with 7×7 convolutions (L_1) has, in contrast to the others, a Max-Pooling operation.



(a) Array size 8x8 **(b) Array size 16x16** **(c) Array size 32x32**

Figure 4: Evaluation of different layer configurations across different array sizes on a logarithmic scale. L_1, L_2 and L_3 represent 7×7, 1×1 and 3×3 convolution operations, respectively

Table 1: Simulation time comparison of our work with cycle-accurate simulation and ScaleSim on a 16×16 systolic array

Workload	This Work	ScaleSim [10]	Cycle-Accurate
L_1	1.8 s	165 s (55x)	9179 s (5099x)
L_2	0.17 s	16 s (138x)	2201 s (12947x)
L_3	0.76 s	147 s (18x)	2667 s (3509x)
ResNet-34	28 s	1 h	> 48 h

Especially here, the differences between the cycle-accurate simulation and ScaleSim are significant. This can be explained by two factors: First, ScaleSim does not model Max-Pooling operations at all. Especially, in Gemmini, pooling and convolutions are fused into one layer. In general, pooling has an impact on the performance estimates. In our experiments, we have observed that 23% more cycles are required for layers with pooling. For this reason, modelling these effects is crucial to accurately model performance. Besides pooling, the underlying mapping plays a role. ScaleSim assumes a different mapping compared to Gemmini allowing for a higher utilization and therefore a deviating cycle count. We are able to take both of these effects in our model into account. Hence, our model reflects the cycle count observed during the simulation more accurately than ScaleSim. Similar trends can be observed over the different array sizes. Looking at the 1×1 convolution operation (L_2), ScaleSim is also unable to accurately reflect the correct cycle counts. The calculated values are too low, since ScaleSim assumes a too high bandwidth, leading to fewer stalls than are acutally present. In case of a 3×3 convolution (L_3), all tools are able to give close estimates.

Besides accuracy, the simulation time for one design evaluation is another very important metric for design space exploration, since faster evaluation aids faster design space exploration. Table 1 shows the simulation time of our approach compared to ScaleSim and the cycle-accurate simulation using the same array sizes and ResNet layers. It has to be noted, that a cycle-accurate simulation of an entire ResNet-34 takes multiple days, making it infeasible for design space exploration. Depending on the workload, the table shows that our approach provides a speed-up of up to 12947x and 138x compared to cycle-accurate simulation and ScaleSim, respectively.

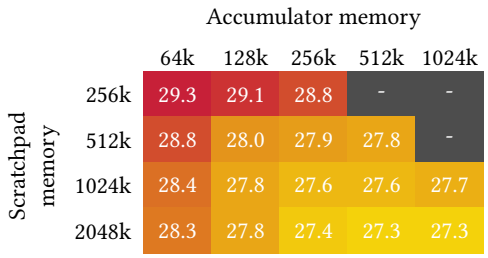


Figure 5: ResNet-34 inference cycle count (in millions) for different memory configurations on a 16×16 array

5.2 Impact of Memory and Array Size

Providing sufficient on-chip memory is a major challenge while designing a DNN accelerator. On-chip memory is very expensive, hence, it is advisable to carefully choose the memory sizes to achieve a high efficiency. To show how our tool can help to choose the memory size, we perform an exploration of a wide range of memory sizes for a ResNet-34 workload, while keeping the array size fixed to 16×16. For our evaluation we assume an off-chip memory with a fixed latency, since Gemmini has an L2-cache in between the off-chip DRAM and the local memories, making this memory hierarchy difficult to model. However, looking at the local memories is still very important, since they have a significant impact on the area. The impact on the cycle-count of different memory sizes using our analytical model is shown in Figure 5. In general, memory sizes affect the tiling of data across the scratchpad and accumulator. Larger memories tend to have a greater impact on area and energy than on performance. A 16×16 array in which the memories are set to the largest configuration (2048k and 1024k) results in 7x more area (in total $11.1mm^2$) and only 7% more performance, in comparison to the smallest configuration which only requires $1.5mm^2$. Due to the significant increase in area, making the memory larger might not always be the correct optimization choice. In comparison, an increase of array size from 16×16 to 32×32 with fixed 256k scratchpad and 64k accumulator memories, adds 73% area and increases performance by 217%. Hence, it should be considered that a larger array can be a better choice than larger memories. Especially in area constrained embedded designs, increasing the array size is the preferable choice.

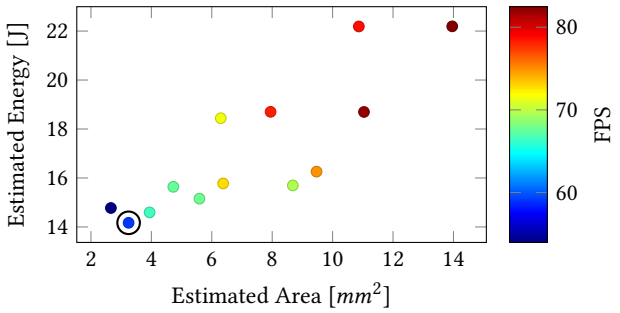


Figure 6: Achieved FPS for ResNet-34 of Pareto optimal array configurations and the associated area and energy

5.3 Exemplary Design Space Evaluation of ResNet-34

To showcase the insights our analytical model can generate, we use our evaluation tool for design space exploration. We explore a ResNet-34 as workload, which demonstrates good prediction accuracy on image processing tasks. For the case study, we assume an inference use case on an embedded compute platform. Our objective is to minimize the energy consumption, while maintaining a high performance. The clock frequency is assumed to be 700 MHz. The design space is limited by a set of architecture constraints. We use the same array sizes as before, but apply less restrictive constrains to the memories. Scratchpad memory size can be set between 128 kB and 4 MB and accumulator memory between 64 kB and 2 MB. Finally, we add a performance constraint that all architectures have to yield at least 30 FPS on ResNet-34 to be considered valid.

With the given constraints, the full design space consists of 57 points and 13 Pareto optimal points. Figure 6 shows the Pareto points with the associated area, energy consumption and cycle count. The performance for the different design points ranges between 34 and 117 FPS. From the plot, we can see gaps in the performance domain instead of a continuous trend. This is caused by Gemmini's architecture constraints. Since array sizes cannot be defined arbitrarily, we have to move for example from a 16×16 array directly to a 32×32, resulting in a large performance gap. It has to be noted, that none of the 8×8 array configurations satisfies the performance requirements. With the results, we determine that the array size is the main indicator for performance. Considering our envisaged use case, we found an energy efficiency to performance sweet-spot with an array size of 32×32, scratchpad memory of 256 kB and accumulator memory of 64 kB. This design configuration is estimated by our tool to have 3.25 mm^2 area and consumes 14.17 J total energy per inference. The total performance settles at 59 FPS.

6 CONCLUSION

In this paper, we have introduced our analytical model for systolic arrays. For an efficient and fast inference of a DNN, it is crucial to design the right DNN accelerator for a given application. Since the design space of DNN accelerators is very large and the complexity of the workload is high, a cycle-accurate evaluation of all design points is infeasible. Our model speeds up the evaluation of a design configuration accurately. We verified our model with a cycle-accurate evaluation of the same architecture, showing less than 1% deviation on a 16×16 array, while the average deviation over all array configurations amounts to 7%. Compared to state-of-the-art systolic array simulators, we demonstrated an improvement in cycle count estimation accuracy and were able to include more instructions like Max-Pooling into our simulation. Moreover, we coupled our analytical model with Accelergy to get estimates of energy consumption and area, besides the raw cycle count, making a design space exploration feasible. Exemplary, we performed this on a case study with ResNet-34, revealing valuable insights on how different design parameters influence energy consumption, area and overall performance.

ACKNOWLEDGMENTS

This work was funded by the German Federal Ministry of Education and Research (BMBF) under grant number 16ME0454 (EMDRIVE). The responsibility for the content of this publication lies with the authors.

REFERENCES

[1] Alon Amid et al. 2020. Chipyard: Integrated Design, Simulation, and Implementation Framework for Custom SoCs. *IEEE Micro* 40, 4 (2020), 10–21. https://doi.org/10.1109/MM.2020.2996616

[2] Steffen Baehr et al. 2019. Low Latency Neural Networks using Heterogenous Resources on FPGA for the Belle II Trigger. *arXiv:1910.13679 [hep-ex, physics:physics]* (Oct 2019). http://arxiv.org/abs/1910.13679

[3] Yu-Hsin Chen, Joel S. Emer, and Vivienne Sze. 2018. Eyeriss v2: A flexible and high-performance accelerator for emerging deep neural networks. *CoRR* abs/1807.07928 (2018). http://arxiv.org/abs/1807.07928

[4] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. 2017. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE Journal of Solid-State Circuits* 52, 1 (Jan. 2017), 127–138. https://doi.org/10.1109/JSSC.2016.2616357

[5] Hasan Genc et al. 2021. Gemmini: Enabling Systematic Deep-Learning Architecture Evaluation via Full-Stack Integration. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. 769–774. https://doi.org/10.1109/DAC18074.2021.9586216

[6] Song Han, Huizi Mao, and William J. Dally. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *arXiv:1510.00149 [cs]* (Feb 2016). http://arxiv.org/abs/1510.00149

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. https://doi.org/10.48550/ARXIV.1512.03385

[8] Tim Hotfilter, Julian Hoefer, Fabian Kreß, Fabian Kempf, and Juergen Becker. 2021. FLECSim-SoC: A Flexible End-to-End Co-Design Simulation Framework for System on Chips. In *2021 IEEE 34th International System-on-Chip Conference (SOCC)*. 83–88. https://doi.org/10.1109/SOCC52499.2021.9739212

[9] Angshuman Parashar et al. 2019. Timeloop: A Systematic Approach to DNN Accelerator Evaluation. In *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 304–315. https://doi.org/10.1109/ISPASS.2019.00042

[10] Ananda Samajdar, Yuhao Zhu, Paul N. Whatmough, Matthew Mattina, and Tushar Krishna. 2018. SCALE-Sim: Systolic CNN accelerator. *CoRR* abs/1811.02883 (2018). http://arxiv.org/abs/1811.02883

[11] Yakun Sophia Shao et al. 2019. Simba: Scaling deep-learning inference with multi-chip-module-based architecture. In *Proceedings of the 52nd annual IEEE/ACM international symposium on microarchitecture (MICRO '52)*. Association for Computing Machinery, New York, NY, USA, 14–27. https://doi.org/10.1145/3352460.3358302

[12] Iris Walter et al. 2021. Embedded Face Recognition for Personalized Services in the Assistive Robotics. In *Machine Learning and Principles and Practice of Knowledge Discovery in Databases*. Springer International Publishing, Cham, 339–350.

[13] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: An insightful visual performance model for multicore architectures. *Communications of The Acm* 52, 4 (April 2009), 65–76. https://doi.org/10.1145/1498765.1498785

[14] Yannan Nellie Wu, Joel S. Emer, and Vivienne Sze. 2019. Accelergy: An Architecture-Level Energy Estimation Methodology for Accelerator Designs. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–8. https://doi.org/10.1109/ICCAD45719.2019.8942149