SAMO: Optimised Mapping of Convolutional Neural Networks to Streaming Architectures

Alexander Montgomerie-Corcoran*, Zhewen Yu* and Christos-Savvas Bouganis
Imperial College London, UK
{alexander.montgomerie-corcoran15, zhewen.yu18, christos-savvas.bouganis}@imperial.ac.uk

***Abstract*—Significant effort has been placed on the development of toolflows that map Convolutional Neural Network (CNN) models to Field Programmable Gate Arrays (FPGAs) with the aim of automating the production of high performance designs for a diverse set of applications. However, within these toolflows, the problem of finding an optimal mapping is often overlooked, with the expectation that the end user will tune their generated hardware for their desired platform. This is particularly prominent within Streaming Architecture toolflows, where there is a large design space to be explored. In this work, we establish the framework SAMO: a Streaming Architecture Mapping Optimiser. SAMO exploits the structure of CNN models and the common features that exist in Streaming Architectures, and casts the mapping optimisation problem under a unified methodology. Furthermore, SAMO explicitly explores the reconfigurability property of FPGAs, allowing the methodology to overcome mapping limitations imposed by certain toolflows under resource-constrained scenarios, as well as improve on the achievable throughput. Three optimisation methods - Brute-Force, Simulated Annealing and Rule-Based - have been developed in order to generate valid, high performance designs for a range of target platforms and CNN models. Results show that SAMO-optimised designs can achieve 4x-20x better performance compared to existing hand-tuned designs. The SAMO framework is open-source: https://github.com/AlexMontgomerie/samo.**

***Index Terms*—streaming architecture, neural network accelerator, optimisation**

I. Introduction

The success of deep learning models, primarily in the form of CNNs, has fuelled research into custom hardware accelerators tuned for specific models. A popular type of platform for these accelerators are FPGAs, as their versatility and range of sizes suit a variety of applications. Many toolflows have been developed that reduce the time to develop such FPGA-based systems, and the architectures generated by these toolflows generally come under two categories: Systolic Array and Streaming (dataflow) architectures [1].

Systolic Array architectures have the ability to execute nearly any CNN model by mapping all the convolution layers to a time-shared matrix multiplication engine. Due to this flexibility, Systolic Array architectures are often served as compute kernels in general-purpose neural network accelerators that are not tailored to specific CNN models [2], [3].

Streaming Architectures, on the contrary, tailor their hardware towards the computation and memory workload of a specific CNN model, with the promise of greater performance. Instead of time-sharing processing elements between layers, each layer is mapped to a custom computation kernel tailored to the characteristics of the layer, where the overall CNN computation is achieved by pipelining these kernels. As such, Streaming Architectures have higher throughput and energy efficiency compared to equivalent Systolic Array architectures [4]. However, the design process for a Streaming Architecture accelerator is often more time consuming and tedious, as the customisability brings forward a large design space to be explored. Early work on Streaming Architecture accelerators explored the design space manually, frequently resulting in many sub-optimal designs [5], [6]. Several works have proposed algorithms that allow automatic design space exploration [7]–[9], which are closely coupled to the specific accelerator framework and do not generalise to other toolflows.

This paper introduces the SAMO framework, which addresses the optimisation stage of mapping CNN models to Streaming Architectures in a unified manner. Key innovations of the tool are the introduced abstract representation for capturing the characteristics of the accelerator's building blocks (both performance and resource requirements) and the Hardware Description Graph (HD-Graph), a data structure that allows the optimisation of the CNN mapping to a target FPGA device. Furthermore, the proposed framework makes explicit use of the reconfiguration feature of FPGA devices, and introduces a partitioning methodology that allows CNN mapping toolflows to achieve high throughput designs, as well as to produce valid designs even in a resource constraint setting.

II. Background

Among existing Streaming Architecture toolflows, fpgaConvNet [10], FINN [11] and HLS4ML [6] are good representatives of that set due to their popularity within the research community, as well as their wide variation in the accelerator design space that they define [1]. These three toolflows are referred to as backends throughout the rest of this paper.

The core computation unit in FINN is the Matrix-Vector Threshold Unit (MVTU) which contains a configurable number of Processing Elements (PE) and SIMD lanes for parallel Matrix-Vector operations [4], [12]. The number of PE and SIMD lanes for each MVTU are refined by iteratively allocating extra resources to the slowest MVTU in the whole accelerator [11]. However, the authors of FINN also state this algorithm to be sub-optimal and can often be outperformed

*equal contribution

1946-1488/22/$31.00 ©2022 IEEE
DOI 10.1109/FPL57034.2022.00069

by hand-tuning[1]. FINN has also recently been extended into multi-FPGA execution where Integer Linear Programming is used to balance the workload between multiple FPGAs [13]. However the design mapped in each FPGA is still manually tuned.

HLS4ML sacrifices the configurability of the accelerator to obtain simple and low-latency designs [6], [14]. Therefore, it defines a relatively small design space compared with other toolflows and supports two hardware generation modes: *resource* and *latency*. For the *latency* mode, the design architecture is completely unrolled, and the HLS compiler is given the task of optimising the latency for a given initiation interval target. In the case of the *resource* mode, the hardware is only partially unrolled, and the performance of the design is manually tuned through the *reuse-factor* parameter, which defines how often resources are re-used.

fpgaConvNet [10] focuses on the configurability of the accelerator and explores several degrees of parallelism in the network including *coarse-grained folding* and *fine-grained folding*. The toolflow further supports bitstream reconfiguration and partial reloading of weights in order to overcome limitations imposed due to resource constraints. Therefore, fpgaConvNet defines the largest design space amongst the three considered toolflows. In terms of the optimisation algorithm, fpgaConvNet proposed a Simulated Annealing optimiser [9] to explore its design space, however this has been tailored to this specific toolflow.

To summarise, the above toolflows provide configurable hardware building blocks and define a large design space to expose a trade-off between performance and resources. However, a unified way of exploring the space across different toolflows is still missing. SAMO aims to fill this research gap by capitalising on the common characteristics of the CNN models and streaming architectures.

III. Generalised Optimisation Problem

This section outlines the generalised optimisation problem of tailoring a parameterised Streaming Architecture to a target CNN and FPGA pair. The aim is to efficiently utilise the available resources on the platform to optimise for either a throughput or latency objective, for a given network. For simplicity, this section only discusses sequential CNN models, even though SAMO can support residual networks as well.

A. Hardware Description Graph

In order to generalise the optimisation of the CNN mapping to a target FPGA device, SAMO proposes a data structure called the Hardware Description Graph (HD-Graph), which provides an abstraction of both the CNN model topology and the hardware building block implementation. The definition of the HD-Graph is as follows:

Assume the given CNN model is represented as a Directed Acyclic Graph with L layers as $M = \{l_1, \ldots, l_L\}$, where l_i is the i^{th} layer within the CNN model. This graph has edges E_M between the layers. For sequential networks, the edges are only between adjacent nodes, so $E_M = \{(l_1, l_2), \ldots, (l_{L-1}, l_L)\}$.

SAMO uses a parser (elaborated on in Section IV-A) to translate each layer of the CNN model to a computation node or set of computation nodes in the HD-Graph, where each computation node corresponds to a parameterised hardware building block implemented in the backend. After the translation, the HD-Graph is described as H, which contains N computation nodes, $H = \{n_1, \ldots, n_N\}$ where n_i is the i^{th} node within it. As the CNN models are sequential, so is the HD-Graph, and the edges of H are such that $E_H = \{(n_1, n_2), \ldots, (n_{N-1}, n_N)\}$.

B. Partitioning

SAMO introduces a partitioning methodology to generate high throughput designs by reconfiguring and time-multiplexing hardware building blocks on the single FPGA device. The partitioning methodology is described in terms of cuts of the HD-Graph H, which transform it into multiple sub-graphs $\boldsymbol{P}$ where each sub-graph constitutes an FPGA configuration containing a subset of nodes from the HD-Graph.

The positions of where the cuts take place are represented by the optimisation variable $\boldsymbol{C} = \{e_1, \ldots, e_{|\boldsymbol{P}|-1}\}$, and $|\boldsymbol{P}|$ denotes the number of sub-graphs. As such, the computation nodes from which the sub-graphs $\boldsymbol{P}$ are constructed are defined in Eq. (1).

$$\boldsymbol{P} = \begin{cases} \{n_{e_i+1}, \ldots, n_{e_{i+1}}\} \; \forall \; e_i \in \boldsymbol{C} & |\boldsymbol{C}| > 1 \\ \{n_1, \ldots, n_{e_i}\}, \; \{n_{e_{i+1}}, \ldots, n_N\} & |\boldsymbol{C}| = 1 \\ \{n_1, \ldots, n_N\} & |\boldsymbol{C}| = 0 \end{cases} \tag{1}$$

This leads to the properties that the partitions are disjoint ($\bigcap \boldsymbol{P} = \emptyset$) and complete ($\bigcup \boldsymbol{P} = H$).

C. Variables

In the HD-graph, each computation node corresponds to a parameterised hardware building block whose implementation is backend-specific. To capture the possible parametrisation of the current landscape of Streaming Architecture frameworks, SAMO defines three associated variables for each node n_i: *input channel folding* (s_i^I), *output channel folding* (s_i^O) and *kernel folding* (k_i). These variables can be vectorised across all nodes ($\boldsymbol{s}^I, \boldsymbol{s}^O, \boldsymbol{k}$).

The *input* and *output channel folding* variables describe the degree of parallelism of the channel dimension of the feature-map entering and exiting a node respectively. The *kernel folding* variable describes the parallelism within a node.

D. Objective

With all the optimisation variables defined, the optimisation problem is now outlined. SAMO supports two optimisation objectives, which are latency minimisation and throughput maximisation. Both objectives depend on the variables $V = \{\boldsymbol{C}, \boldsymbol{s}^I, \boldsymbol{s}^O, \boldsymbol{k}\}$, which denote the HD-Graph configuration.

[1]https://github.com/Xilinx/finn/blob/main/src/finn/transformation/fpgadataflow/set_folding.py

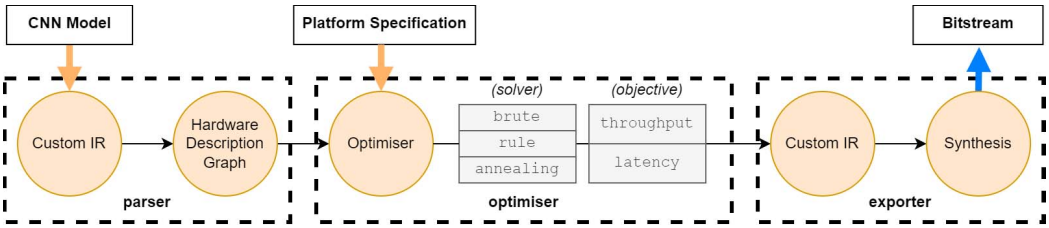


Fig. 1: Overview of the proposed SAMO framework

As the hardware building blocks are pipelined and the CNN models are assumed to be sequential, the latency of each sub-graph is dictated by its slowest node[2], defined in Eq. (2).

$$\mathcal{T}(P_i) = \max\{t(n_j|s_j^I, s_j^O, k_j) : n_j \in P_i\} \tag{2}$$

$t(n_j|s_j^I, s_j^O, k_j)$ and $\mathcal{T}(P_i)$ denotes the latency estimate for the node n_j and the partition P_i respectively. When considering the whole HD-graph, the objective for minimising latency of a design is described in Eq. (3), where t_{conf} is the reconfiguration time of the device.

$$\mathcal{O}(V) = \left(\sum_{P_i \in \boldsymbol{P}} \mathcal{T}(P_i)\right) + |\boldsymbol{C}| \cdot t_{conf} \tag{3}$$

Conversely, the objective of maximising throughput for a given FPGA and network pair is described in Eq. (4), where B is the batch size for the inputs to be run.

$$\mathcal{O}(V) = -\frac{B}{B \cdot \left(\sum_{P_i \in \boldsymbol{P}} \mathcal{T}(P_i)\right) + |\boldsymbol{C}| \cdot t_{conf}} \tag{4}$$

The above two optimisation objectives can be used to construct the following optimisation problem described in Eq. (5), where $\mathcal{O}$ represents the objective.

$$\min_{V} \mathcal{O}(V), \quad V = \{\boldsymbol{C}, \boldsymbol{s}^I, \boldsymbol{s}^O, \boldsymbol{k}\} \tag{5}$$

E. Constraints

In order to generate a valid and synthesisable accelerator design, there are certain constraints imposed on the optimisation problem. In this subsection, we define the constraints that are observed across all backends, although specific backends are not necessarily constrained by all.

Eq. (6) defines the resource constraint that each partition must fit within the FPGA on-chip resource budget.

$$\begin{aligned} \mathcal{R}(P_i) &= \sum_{n_j \in P_i} r(n_j|s_j^I, s_j^O, k_j) \\ \mathcal{R}(P_i) &\leq \mathcal{R}_{platform}, \ \forall \ P_i \in \boldsymbol{P} \end{aligned} \tag{6}$$

$r(n_j|s_j^I, s_j^O, k_j)$ and $\mathcal{R}(P_i)$ denote the resource utilisation per node and per partition respectively. The resource types considered include DSP, BRAM, LUT and FF.

Apart from on-chip resources, the memory bandwidth for a given partition is also bounded, as described in Eq. (7).

$$\mathcal{B}(P_i) = \frac{\mathcal{D}^I(P_i) + \mathcal{D}^O(P_i)}{T(P_i)} \leq \mathcal{B}_{platform}, \ \forall \ P_i \in \boldsymbol{P} \tag{7}$$

[2]Pipeline depth is ignored as it has a negligible effect on latency.

where $\mathcal{D}^I(P_i)$ and $\mathcal{D}^O(P_i)$ are the folded dimensions of the feature-map in and out of the partition respectively, and $\mathcal{B}_{platform}$ is the bandwidth upper bound of the given platform.

For backends which do not support padding, the channel folding variables must be factors of the channel dimension of the feature-map that the node is operating on, which is referred to as the *channel factor* constraint. This is described in Eq. (8), where $c^I(n_i)$ and $c^O(n_i)$ are the input and output channel dimensions of the feature-map of the i^{th} node respectively.

$$\begin{aligned} c^I(n_i) \bmod s_i^I &= 0 \ \forall \ n_i \in H \\ c^O(n_i) \bmod s_i^O &= 0 \ \forall \ n_i \in H \end{aligned} \tag{8}$$

Furthermore, there are certain types of layers, such as Max Pooling or ReLU layers, where the output channel dimension depends on the input. Therefore, for the computation nodes corresponding to these type of layers, $H' \subset H$, they must have matching *input channel folding* (s_i^I) and *output channel folding* (s_i^O), which is referred to as *intra folding matching*, as described in Eq. (9),.

$$s_i^I = s_i^O \ \forall \ n_i \in H' \tag{9}$$

The constraint of matching folding factors may also exist between nodes in order to ensure that all the data lines are connected, which is referred to as *inter folding matching*, as described in Eq. (10).

$$s_i^O = s_{i+1}^I \ \forall \ n_i \in H \tag{10}$$

IV. Backend Integration & Optimisation Schemes

So far, SAMO has defined the constrained optimisation problem in a unified way. In this section, we elaborate on how SAMO interacts with backends, and subsequently solve the optimisation problem. An overview of our framework is given in Figure 1, which highlights the key components of SAMO including the parser, optimiser and exporter.

A. Parser: customised IR to HD-graph

The existing backends take the CNN model as an input and transform it into their own customised Intermediate Representation (customised IR), a graph that contains the backend-specific information to describe the characteristics of the corresponding hardware building blocks. Therefore, the parser of SAMO is responsible for further abstracting and unifying the customised IR into the generalised HD-graph.

At the node level, the tunable design parameters in the customised IR are mapped to their respective optimisation

variables in the HD-Graph (Table I). This mapping is not necessarily one-to-one. For example, the *reuse-factor* in HLS4ML is mapped to the product of *input channel folding*, *output channel folding* and *kernel folding*.

Optimsation Variable	Backend Design Parameter		
	fpgaConvNet	*FINN*	*HLS4ML*
Input Channel Folding	Coarse-In	SIMD	Reuse-Factor
Kernel Folding	Fine		
Output Channel Folding	Coarse-Out	PE	

TABLE I: Relationship between backend-specific design parameters and HD-graph optimisation variables.

The parser exposes the resource and latency models to the HD-Graph, if these models have been provided by the backend. For backends which do not contain these models, such as HLS4ML, analytical latency and DSP predictions are provided.

B. Optimiser: Brute-Force

SAMO provides three optimisation solvers to exploit the trade-off between the optimisation evaluation time and the performance of the generated design. Among them, the Brute-Force optimiser enumerates all possible values of optimisation variables. Any design point that violates the constraints of the HD-Graph is discarded. The rest of design points are evaluated on the objective function and the optimal one is then identified. The advantage of the Brute-Force optimiser is that it guarantees the identification of the optimal design point, at the cost of a lengthy optimisation time.

C. Optimiser: Simulated Annealing

Simulated Annealing [15] is a well-known stochastic optimisation algorithm. Algorithm 1 outlines its implementation. At first, all the optimisation variables, $V = \{\boldsymbol{C}, \boldsymbol{s}^I, \boldsymbol{s}^O, \boldsymbol{k}\}$ are initialised to a resource-minimal state (V_{init}), where the computation inside each node is in sequential order and the HD-graph is split into as many partitions as possible.

Algorithm 1 Simulated Annealing Optimisation Algorithm

$K = K_{start}$, $V = V_{init}$ ▷ initialisation
while $K > K_{min}$ **do**
  $V_{prev} = V$ ▷ store previous design
  V = random transformation on V
  if constraints satisfied **then**
    if $\psi(V, V_{prev}, K) < x \sim U(0,1)$ **then**
      $V = V_{prev}$ ▷ reject new design
  $K = \lambda \cdot K$ ▷ reduce temperature

It then enters the main optimisation loop where it performs a random change to the optimisation variables in each iteration and evaluates the decision function (Eq. (11)). If the output is below the stochastic decision threshold x, which is a random variable sampled from a uniform distribution, or any constraint has been violated, then the current design is discarded and the previous design is kept.

$$\psi(V, V_{prev}, K) = exp\left(\min\left(0, \frac{\mathcal{O}(V_{prev}) - \mathcal{O}(V)}{K}\right)\right) \quad (11)$$

The algorithm requires two hyper-parameters: K and λ. K is *temperature*, which starts from K_{start} and decays by the *cooling rate* λ, for each iteration until reaching K_{min}.

D. Optimiser: Rule-Based

The Rule-Based optimiser starts at a resource-minimal state and solves the optimisation problem using a deterministic method.

Algorithm 2 Ruled-based Optimisation Algorithm

procedure OPTIMISE PARTITION(P)
  repeat
    $j = \text{argmax}\ t(n_j | s_j^I, s_j^O, k_j), n_j \in P$ ▷ slowest j
    $\Delta = \{\delta_s^I, \delta_s^O, \delta_k\}\ ,\ \Delta > 0$ ▷ folding increment
    $r_{new} = r(n_j | s_j^I + \delta_s^I, s_j^O + \delta_s^O, k_j + \delta_k)$
    $r_{prev} = r(n_j | s_j^I, s_j^O, k_j)$ ▷ predict resource
    $\min r_{new} - r_{prev}$ ▷ smallest resource change
    $s_j^I \overset{\Delta}{=} s_j^I + \delta_s^I,\ s_j^O = s_j^O + \delta_s^O,\ k_j = k_j + \delta_k$
  until no more resources **or** fully parallel

As outlined in Algorithm 2, the Rule-based optimiser deals with each partition independently. For each partition, the optimiser identifies the slowest node and reduces its latency by updating the value of the optimisation variable with the increment Δ, which causes the smallest change in resources. The above step repeats until the latency of the slowest node cannot be reduced further, either due to reaching a resource constraint, or that the slowest node is fully unrolled. The optimiser then incrementally merges pairs of partitions. This is done by applying heuristics which identify partition merges which are more likely to create an optimal design point.

These heuristics are based on the following characteristics of the partition:

- is memory-bound
- the slowest node is already fully unrolled
- latency is smaller than reconfiguration time

Once all the identified partitions cannot be merged further, the optimiser terminates.

E. Exporter: customised IR to HD-graph

The optimised HD-graph is transformed back to the custom IR belonging to the respective backend, and the design parameters of the hardware building blocks are configured with the values of the corresponding optimisation variables. This optimised IR can then be used to synthesise and generate bitstreams for the target platform.

Backend	Network/ Precision	Partitions			Latency (ms/batch)			Throughput (img/s)			Resource (%)		
		init.	*lat.*	*thr.*	*init.*	*lat.*	*thr.*	*init.*	*lat.*	*thr.*	*init.*	*lat.*	*thr.*
fpgaConvNet	LeNet/w16a16	1	1	3	16.0	2.0	283.9	62.5	500.0	901.6	50.8	50.5	83.5
	CNV/w16a16	1	6	8	289.0	421.7	1304.7	3.5	2.4	196.2	120.5	64.5	82.0
FINN	MPCNN/w4a4	1	2	3	10.0	84.5	432.1	100	11.8	592.4	48.1	44.4	47.4
	CNV/w1a1	1	1	1	289.0	0.3	73.7	3.5	3472.2	3472.2	18.7	33.0	33.0
	MobileNetV1/w4a4	1	5	13	513.8	475.3	11347.0	1.9	2.1	22.6	187.1	63.8	82.8

TABLE II: Comparison of optimisation results for the Rule-Based optimiser against unoptimised designs (*init.*) targeting both throughput (*thr.*) and latency (*lat.*) objectives for a ZedBoard device. The batch size for the throughput objective is 256. Resource is the average utilisation across DSP, BRAM and LUT. Designs with resources in red violate constraints.

V. Evaluation

The proposed SAMO framework is evaluated in terms of its ability to identify high-performance designs. Outlined in Table III, the CNN models used in the evaluation are sourced from the design examples provided by the backends. In terms of target platforms, ZedBoard, ZC706 and U250 are selected to evaluate the framework over resource profiles that span from embedded systems to high-end server-graded FPGA device.

Task	Network	No. Conv	No. Dense	Params
Jet Tagging	*3-layer*	0	4	4K
Hand Gestures	*MPCNN*	3	2	70K
MNIST	*TFC*	0	4	59K
	LeNet	2	2	430K
CIFAR-10	*CNV*	6	3	1.543M
ImageNet	*VGG11*	8	3	132.854M
	MobileNetV1	27	1	4.209M
	ResNet50	52	1	25.371M

TABLE III: Model zoo for evaluation.

A. Choice of Optimisers

In subsection IV-B to IV-D, we demonstrated the implementation of three optimiser solvers. Among them, the Brute-Force optimiser exhaustively explores the design space. Apart from the *3-layer* network, such an exhaustive exploration is intractable, with some tests estimated to take centuries to complete.

Beyond the Brute-Force optimiser, SAMO also provides the Simulated-Annealing and Rule-Based optimisers to traverse the design space and deal with its large cardinality. Fig. 2 demonstrates the performance of these two optimisers by reporting the optimisation time and achieved performance.

For Simulated Annealing, due to its stochasticity, 50 independent runs are launched with different random seeds. The annealing temperature K is initialised as 1000 and reduced by 2% every iteration until reaching the minimum temperature K_{min}=1. It is then left to run at minimum temperature and for the same time budget as the Rule-Based optimiser. The distribution across 50 runs of Simulated Annealing is calculated and compared with the Rule-Based result, which is deterministic.

For CNV, all 50 runs of the Simulated Annealing optimiser give exactly the same latency as the Rule-Based optimiser and converge quicker, suggesting that Simulated Annealing is able to find optimal designs much faster. However this is not the case when it comes to the optimisation of a wider and deeper[3] network, *MobileNetV1*, where the Simulated Annealing runs did not converge. Considering each run of the optimiser takes about 33 minutes to complete on an Intel i7-9700 CPU, the stochasticity of the annealing algorithm becomes a major drawback for reproducible high performance designs.

To summarise, the results suggest that the Brute-Force, Simulated Annealing and Rule-Based Optimiser should be used for small, medium, and large networks respectively. The Brute-Force optimiser guarantees identification of an optimal design point, but its lengthy search time makes it only feasible for small networks. Both the Simulated Annealing optimiser and the Rule-Based optimiser efficiently explore the design space at the risk of being stuck at a local minimum. However, the Rule-Based optimiser performs better whilst handling larger networks, as the randomness of Simulated Annealing makes it sub-optimal when the same time budget is considered.

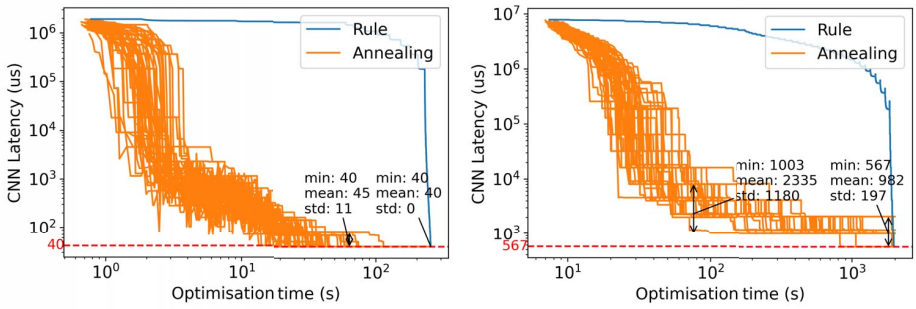
B. Discussion on Partitioning

As described in Section III-D, SAMO supports both latency and throughput objectives for optimisation. By introducing partitioning into the design space, these two objectives can lead to very different hardware outcomes. This section evaluates the difference in throughput and latency driven designs for both high and low-end devices.

Table II demonstrates the capability of SAMO to identify the design points for a resource-constrained device. Both throughput and latency objectives are compared to an unoptimised design whose optimisation variables are all set to 1.

The first benefit of introducing partitioning is the ability to overcome resource constraints. For *CNV* for fpgaConvNet, and *MPCNN* and *MobileNetV1* for FINN, the unoptimised designs exceed the available resources for the ZedBoard. By splitting the HD-Graph into multiple sub-graphs, SAMO is able to overcome this constraint for both latency and throughput optimised designs. This is at the cost of increased latency needed for reconfiguration. And in all cases where the unoptimised design fits, SAMO is able to find an improved design.

[3]"wide" and "deep" are used here to describe the number of channels and the number of layers.



(a) CNV (w1a1) (b) MobileNetV1 (w4a4)

Fig. 2: Comparison between the Simulated Annealing and Rule-Based optimisers. Networks are mapped to a U250 using the FINN backend with a latency objective.

Partitioning also enables much greater throughput to be achieved compared to being constrained to a single partition. In Table II, it can be seen that for *LeNet* for fpgaConvNet, the throughput for the throughput-driven design is nearly double that of the latency-driven one. This is in part due to the ability to amortise the cost of reconfiguration through large batch sizes. Here the throughput-driven design spends significantly more time executing hardware than reconfiguring. It is also observed that throughput-driven designs lead to more efficient use of the hardware per partition, with much higher resource utilisation compared to latency-driven designs.

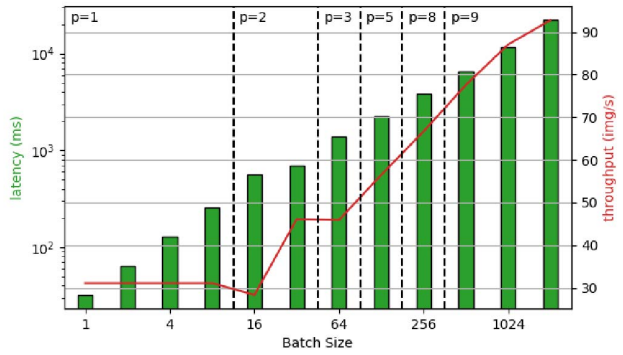


Fig. 3: Comparison of Throughput and Latency using different batch sizes for a VGG11 network targeting a U250 device using the fpgaConvNet backend. p indicates the number of partitions.

Exploring throughput-driven designs further, different batch sizes are used in Fig. 3 to show achievable throughput for a U250 device deploying *VGG11* using the fpgaConvNet backend. This figure highlights that partitioning is not only a mechanism for satisfying resource constraints, but also a way of further improving throughput, since as the batch size is increased, more partitions are used to increase throughput.

C. Comparison with Existing Designs

Table IV compares designs generated by SAMO with example designs provided by the authors of each backend, where the design parameters of the hardware building blocks are manually tuned. The results highlight the power of SAMO's automated design space exploration, as it can achieve the same or higher performance compared to a hand-crafted method, demonstrating improvements of 4x-20x in performance across different backends.

Backend	Platform	Network/Precision	Latency (us) *baseline*	Latency (us) *SAMO*
HLS4ML[4]	U250	3-layer/w16a16	0.001	0.001
fpgaConvNet [9]	Zedboard	LeNet/w16a16	7917.0	**2000.0**
		MPCNN/w16a16	3919.0	**180.0**
FINN[5]	U250	CNV/w1a1	163.8	**41.0**
		MobileNetV1/w4a4	567.9	567.9
		ResNet-50/w1a2	4515.8	**3081.3**

TABLE IV: Comparison with baseline designs. Latency is predicted by the backend performance models.

VI. Conclusion & Future Work

This paper presents the SAMO framework, an open-source Streaming Architecture Mapping Optimiser, which serves as a powerful tool for CNN Accelerator designers. The framework has been integrated with popular open-source Streaming Architectures in order to prove its ability in achieving high performance designs across a range of CNN networks and FPGA platforms. The potential of different optimisers are demonstrated, with considerable gains in performance observed. This framework can be seen as a launchpad for further research into CNN-FPGA co-design, with potential for use in the exploration of Neural Architecture Search (NAS), as well as exploring improved optimisation methods.

Acknowledgement

For the purpose of open access, the author(s) has applied a Creative Commons Attribution (CC BY) license to any Accepted Manuscript version arising.

[4]https://github.com/fastmachinelearning/hls4ml
[5]https://github.com/Xilinx/finn-examples

REFERENCES

[1] S. I. Venieris, A. Kouris, and C.-S. Bouganis, "Toolflows for mapping convolutional neural networks on fpgas: A survey and future directions," *ACM Comput. Surv.*, vol. 51, no. 3, 2018.

[2] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017.

[3] D. A. Vink, A. Rajagopal, S. I. Venieris, and C.-S. Bouganis, "Caffe barista: Brewing caffe with fpgas in the training loop," in *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 2020.

[4] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017.

[5] R. DiCecco, G. Lacey, J. Vasiljevic, P. Chow, G. Taylor, and S. Areibi, "Caffeinated fpgas: Fpga framework for convolutional neural networks," in *2016 International Conference on Field-Programmable Technology (FPT)*. IEEE, 2016.

[6] J. Duarte, S. Han, P. Harris, S. Jindariani, E. Kreinar, B. Kreis, J. Ngadiuba, M. Pierini, R. Rivera, N. Tran, and Z. Wu, "Fast inference of deep neural networks in FPGAs for particle physics," *Journal of Instrumentation*, vol. 13, no. 07, Jul. 2018.

[7] H. Li, X. Fan, L. Jiao, W. Cao, X. Zhou, and L. Wang, "A high performance fpga-based accelerator for large-scale convolutional neural networks," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2016.

[8] X. Zhang, J. Wang, C. Zhu, Y. Lin, J. Xiong, W.-m. Hwu, and D. Chen, "Dnnbuilder: an automated tool for building high-performance dnn hardware accelerators for fpgas," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018.

[9] S. I. Venieris and C.-S. Bouganis, "fpgaconvnet: A framework for mapping convolutional neural networks on fpgas," in *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2016.

[10] ——, "fpgaconvnet: Mapping regular and irregular convolutional neural networks on fpgas," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 2, 2019.

[11] M. Blott, T. B. Preußer, N. J. Fraser, G. Gambardella, K. O'brien, Y. Umuroglu, M. Leeser, and K. Vissers, "Finn-r: An end-to-end deep-learning framework for fast exploration of quantized neural networks," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 11, no. 3, 2018.

[12] J. Faraone, G. Gambardella, D. Boland, N. Fraser, M. Blott, and P. H. Leong, "Customizing low-precision deep neural networks for fpgas," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2018.

[13] T. Alonso, L. Petrica, M. Ruiz, J. Petri-Koenig, Y. Umuroglu, I. Stamelos, E. Koromilas, M. Blott, and K. Vissers, "Elastic-df: Scaling performance of dnn inference in fpga clouds through automatic partitioning," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 15, no. 2, dec 2021.

[14] T. Aarrestad, V. Loncar, N. Ghielmetti, M. Pierini, S. Summers, J. Ngadiuba, C. Petersson, H. Linander, Y. Iiyama, G. Di Guglielmo *et al.*, "Fast convolutional neural networks on fpgas with hls4ml," *arXiv preprint*, 2021.

[15] C. R. Reeves, Ed., *Modern Heuristic Techniques for Combinatorial Problems*. USA: John Wiley & Sons, Inc., 1993.