Optimizing Accelerator Configurability for Mobile Transformer Networks

Steven Colleman (*), Peter Zhu (**), Wei Sun (**), Marian Verhelst (*)
(*) Department of Electrical Engineering, MICAS-ESAT, KULeuven, Belgium. (**) OPPO Electronics

***Abstract*—Transformers are increasingly used to process time series data. Their deployment in mobile devices is however challenging due to their large computational requirements. Hardware acceleration through custom neural network processors can reduce the resulting latency and energy footprint of the network. Yet, the large variety of different layer types in mobile transformers troubles the selection of the best accelerator architecture and hardware mapping. Specifically, the layer performance strongly depends on the spatial unrolling dimensions, i.e. the layer loop dimensions along which hardware parallelization is enabled. This paper will therefore research the best datapath organization, and required datapath flexibility to efficiently support mobile transformers. The MobileViT-S network is selected as the reference network for its wide variety in layer types. Results are explored across a wide range of accelerator area (datapath dimensions, memory size) and bandwidth constraints.**

***Index Terms*—CNN, transformer, cross-layer, configurability, hardware modelling and optimization**

I. Introduction

The importance of neural networks is rising strongly across a broad range of applications. However, modern neural networks consist of tens or hundreds of layers with a big variability in layer type. A transformer network like MobileViT-S [1] contains classical layers with a 3x3 kernel, pointwise layers, depthwise layers and attention layers that perform a matrix-matrix-multiplication. There are even major differences between layers of the same type: layers deeper in the network contain more channels and a smaller spatial dimension in comparison with layers in the beginning of the network. As illustrated in Figure 1, parallelization in the computation of a convolutional layers happens both spatially (SU, spatial unrolling) as temporally (TU, temporal unrolling). SUs deal with the parallel execution of computations in the PE array in a given clock cycle. In the example, computations for 8 input channels and 8 output channels happen in parallel. These SU factors will be indicated with a suffix 'u'. TUs deal with the order in which computations happen, the order of the for loops.

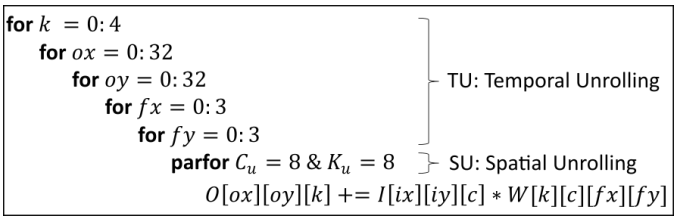


Fig. 1: Meaning of SU and TU.

Different layer types lead to other optimal SUs, as illustrated in [2] [3]. They show that if a hardware accelerator supports a few SUs, latency and energy benefits are obtained. These SUs impact the dataflow and resources for overhead on-chip and must therefore be carefully selected. According TUs can be implemented with the controller.

[3] has presented COAC, an automated design flow tool to derive the optimal set of SUs for a given network and the according resource overhead. However, transformer networks (especially the matrix-matrix multiplication layers) are not handled in this study. Above, it is not discussed what ideal according TUs are for the found SUs and the study is only limited to 1 hardware architecture. Therefore, the main contribution of this paper is to analyze the optimal execution (both SU and TU) for transformer networks. These are very difficult networks as they contain all layer types. We have selected MobileViT-S as a case study network to draw general conclusions. In addition, we also discuss the impact of the architecture dimensions (number of PEs and memory hierarchy).

II. Background and Methodology

Traditional CNN layers contain 6 nested loops: Fx and Fy for filter kernel, Ox and Oy for output feature dimensions and C and K for input and output channels, respectively. For depthwise layers, the latter two are replaced by G for groups. The scheduling of these loops on custom neural network hardware accelerators can happen temporally, spatially or a combination of them. The challenge in neural accelerator design is to find the most optimal subset of SUs to support in hardware. This subset should be diverse enough, to enable efficient mapping of the wide variety of neural layer topologies. Yet, it should be sufficiently similar/simple to avoid large datapath reconfiguration overhead. For this reason, [3] introduced COAC (Cross-Layer Optimization of Accelerator Configurability).

A. COAC

COAC starts by generating all SUs which fully cover the complete PE array consisting of a predefined number of nb_{PEs} processing elements (PE), such that for a classical and pointwise layer, $nb_{PEs} = Ox_u \cdot Oy_u \cdot Fx_u \cdot Fy_u \cdot C_u \cdot K_u$ and $nb_{PEs} = Ox_u \cdot Oy_u \cdot Fx_u \cdot Fy_u \cdot G_u$ for a depthwise layer. Subsequently, COAC will for each network layer under study and each derived SU, use ZigZag [4] to find the two most optimal TUs to minimize energy and latency, respectively. Based on these results, optimal subsets of supported SUs are found to cross-layer optimize the latency/energy for (a suite of)

complete networks. COAC is however characterized by a few shortcomings, making it unable to assess emerging transformer networks. These will be overcome in this paper.

B. COAC extensions/optimizations

1) SU pruning: In large PE arrays, the number of possible SUs can be very large, negatively impacting optimization time. Therefore, a general SU pruning technique has been added to COAC that can drastically reduce the number of SUs if all layers have $Ox = Oy$ and $Fx = Fy$. To minimize input memory bandwidth without increasing the weight or output memory bandwidth, we always select Ox_u and Oy_u such that the number of required input pixels per channel in parallel, $(Ox_u + Fx_u - 1).(Oy_u + Fy_u - 1)$, is minimal. For $Fx_u = Fy_u$, this comes down to minimizing $Ox_u + Oy_u$. Due to the parallelism in $Ox.Oy$, we can select $Ox_u \geq Oy_u$ without loss of generality.

2) Matrix-matrix multiplication: COAC [3] does not support matrix-matrix multiplications. Yet, these are of crucial importance for transformer networks (attention units). To overcome this, and still be able to exploit the strength of a tool like COAC, we will cast every matrix-matrix multiplication into an equivalent convolution. A matrix-matrix multiplication $(P * Q).(Q * R) = (P * R)$ can be implemented as a convolutional layer with $C = Q, K = R, Fx = Fy = 1$ and $Ox.Oy = P$. To be able to exploit the SU pruning trick in Section II-B1, we select $Ox = Oy = \sqrt{P}$, which is possible if all values for P are a square, which is the case in MobileViT-S. In transformer networks, many independent matrix-matrix multiplications have to happen in parallel (heads). H multiplications of $(P * Q).(Q * R) = (P * R)$ can be represented using the concept of grouped convolutions, with $G = H, C = H.Q, K = H.R, Fx = Fy = 1$ and $Ox = Oy = \sqrt{P}$.

III. Discussion: impact of SU and TU

In this section, we will study the impact of the SU and TU on the latency and energy to evaluate a given convolutional layer on a given architecture. First, we discuss the latency. SU will impact the PE utilization (how much of the PEs is used), TU will impact the temporal utilization (ratio of number of clock cycles the computation would take without stalling cycles for the given SU versus the actual number of clock cycles the computation is taking), the total utilization (which is inversely proportional with the latency) is then the product of the PE utilization and the temporal utilization. As mentioned by [3], the PE utilization for a depthwise layer is given by

$$PE_{ut} = \frac{Ox.Oy.Fx.Fy.G}{Ox_u\left\lceil\frac{Ox}{Ox_u}\right\rceil Oy_u\left\lceil\frac{Oy}{Oy_u}\right\rceil Fx_u\left\lceil\frac{Fx}{Fx_u}\right\rceil Fy_u\left\lceil\frac{Fy}{Fy_u}\right\rceil G_u\left\lceil\frac{G}{G_u}\right\rceil}$$

For classical layers, a similar formula is derived. This formula illustrates why supporting 2 SUs in the same hardware architecture might be beneficial: two layers with completely different layer dimensions can have completely different PE utilizations. When supporting 2 different enough SUs, each layer can be executed by its best fitting SU.

This paper will now also analytically derive the impact of the TU on the temporal utilization. As the TU contains the loop parameters that are not in the SU, we look for an optimal TU for a given SU. Consider a 2-level memory architecture and assume that the bandwidth of the upper level memory is big enough to make sure that (eventual) stalling cycles are determined by the bandwidth of the lower level.

We start with defining the number of bits are needed every clock cycle for each operand in case this operand is not stationary ('stationary' means that the data from this operand does not change every clock cycle). Here, we take p to be the resolution of each data word. For intermediate outputs, we take double resolution $2p$.

$$W_{needed,SU} = p \cdot C_u \cdot K_u \cdot Fx_u \cdot Fy_u$$

$$I_{needed,SU} = p \cdot C_u \cdot (Ox_u + Fx_u - 1) \cdot (Oy_u + Fy_u - 1)$$

$$O_{needed,SU} = 2p \cdot K_u \cdot Ox_u \cdot Oy_u$$

For the 'for C', 'for K', 'for Ox/Oy' and 'for G' loops as most inner loop, we can now write down the temporal utilization for a given SU, in function of the bandwidths of the three operand memories, taking into account which operands are not stationary.

$$temp_{ut,C} \approx min\left(1, \frac{BW_W}{W_{needed,SU}}, \frac{BW_I}{I_{needed,SU}}\right)$$

$$temp_{ut,K} \approx min\left(1, \frac{BW_W}{W_{needed,SU}}, \frac{BW_O}{O_{needed,SU}}\right)$$

$$temp_{ut,Ox/Oy} \approx min\left(1, \frac{BW_I}{I_{needed,SU}}, \frac{BW_O}{O_{needed,SU}}\right)$$

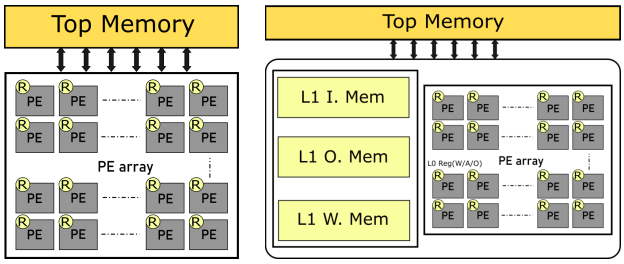
$$temp_{ut,G} \approx min\left(1, \frac{BW_W}{W_{needed,SU}}, \frac{BW_I}{I_{needed,SU}}, \frac{BW_O}{O_{needed,SU}}\right)$$

The equations for the 'for Fx/Fy' loops are not given for complexity reasons, as they are what is named in ZigZag partially relevant loops [4]. However, in practice they might be interesting in many cases as the weights are stationary and inputs/outputs can be shared in between two subsequent loops, decreasing the bandwidths requirements here. We remark that the memory bandwidths highly impact the optimal TU for a given SU and also the optimal SU as an SU with a slightly lower PE utilization can be better if the best according TU has a much better temporal utilization.

For energy, similar equations can be derived. However, energy will only depend on the TU as this determines the number of times each data has to be accessed.

IV. Experiments and discussions

The experiments make use of the extended COAC tool, using memory energy numbers obtained by Cacti [5] using a 32nm technology. The energy-delay product (EDP) is used as optimization objective in all performed optimizations.



(a) Architecture with 1 memory.

(b) Architecture template with 2 memory levels.

Fig. 2: Architectures for experiments

A. Impact of number of PEs

First, we want to investigate whether the EDP gain of supporting 2 SUs versus 1 SU depends on the number of PE arrays and try to understand why. We use an architecture with a single memory level, which supports all operands (weights, inputs and outputs), as illustrated in Fig. 2a. Every PE in the PE array has 3 local registers: 8-bit registers for the weight and the input and a 16-bit register for the (intermediate) output. We vary the number of PEs in the PE array and the bandwidth (BW) of the top memory to demonstrate the impact of the number of PEs on the EDP gain. The reported EDP is the performance across the complete MobileViT-S network. The results are given in Table I.

TABLE I: Impact of number of PEs on EDP gain, with selected individual SU

PEs	BW [bits]	$\frac{EDP_{1SU}}{EDP_{2SUs}}$	Optimal individual SU
256	128	1.557	$[K_u = 8, Oxy_u = 32]$
256	512	1.409	$[K_u = 8, Oxy_u = 32]$
256	2048	1.415	$[K_u = 4, Oxy_u = 64]$
8192	2048	4.124	$[C_u = 2, K_u = 16, Oxy_u = 256]$
8192	8192	3.839	$[C_u = 4, K_u = 8, Oxy_u = 256]$

From this, it is clear that the relative benefit in terms of EDP from combining 2 SUs is larger for an architecture with more PEs (8192 vs 256). This will be the case for every network containing a broad range of layer types. If only 1 SU can be used, most parallelization will be applied on loops that are shared between all the layers. The table also shows which unrollings are optimal in each of the researched cases. Ox_u and Oy_u show to be dominant unrollings in this experiment (product of these two noted as Oxy_u), as these are the only loop unrollings that occur in every network layer type. Bigger PE arrays, however, can not use these unrolling factors as only SU anymore, as this would result in PE under-utilization for some layers with smaller dimensions. This results in additional spatial unrolling across other loop dimensions, in this case C_u and K_u. But not all layers types (depthwise) have a C or K loop, which leads to PE under-utilization there. Thus, there is a maximal unrolling that can be shared between all layers. For big PE arrays, this number becomes smaller in comparison with the total number of PEs. Therefore, it becomes a bigger problem for big PE arrays to support only 1 SU, leading to a higher EDP gain there when they support 2 SUs. The exact gains we obtain are dependent on the relative ratio of all layer types in the network suite under study. A network with as much depthwise layers as classical layers will have a higher benefit as a network with 90% classical layers and only a few depthwise layers.

B. Impact of memory hierarchy on SUs

Of course, one big top memory directly serving all PEs with all 3 operands (weights, inputs and outputs) is not realistic, due to routing constraints and memory port requirements. In a second experiment series, we therefore study the impact of different memory hierarchies on the optimal SU configurability. The architecture template for this experiment is given in Fig. 2b: a top level memory with bandwidth of 2048 bits per cycle serves an Level-1 (:1) operand memory of size 128kB for each of the three operands and local registers for each of 8192 PEs. In Table II, 4 different customizations of this architecture template are given to analyze the impact of the L1 memory bandwidths on the optimal SU and TU combinations. We selected one architecture where the BWs for all operands were equal and 3 architectures where 1 BW was bigger than the other 2 to see how that impacts unrollings for that operands loop(s). We also compare with 'Architecture 0', which is the 1-level memory architecture from previous experiment with 8192 PEs and BW = 2048 bits per clock cycle.

TABLE II: Parameters of memory hierarchies under study.

	BW_W [bits]	BW_I [bits]	BW_O [bits]
Architecture 1	2048	2048	2048
Architecture 2	4096	1024	1024
Architecture 3	1024	4096	1024
Architecture 4	1024	1024	4096

The found SUs are given in Fig. 3. The according latency and energy are shown in Fig. 4, together with the according EDP gains for combining 2 SUs. We first remark that the architectures with BW_I = 1024 bits have Fx_u and Fy_u unrolling in their individual SU. For small BW_I, we need SUs with few input features to have good latency. Unrolling Fx_u and Fy_u increases input reuse and is therefore optimal. However, if bandwidth permits, Fx_u and Fy_u are never selected in individual SUs as there are many layers with kernel 1x1 that would have bad utilization otherwise. For every architecture, in the combination of 2 SUs, there is 1 SU unrolling Fx_u and Fy_u, the other not. The unrolling with Fx_u and Fy_u is always combined with unrolling over G_u as depthwise layers never have a 1x1 kernel. This value for G_u however should not be taken very large to make sure that classical layers can efficiently be executed with them as well. Similarly, the values for G_u are higher for architectures with higher bandwidth which can afford more data reads/writes. In summary: architectures with more severe bandwidth constraints, benefit from unrollings with bandwidth-saving SU loops (more Fx_u/Fy_u, less G_u), at the expense of a lower utilization for several layers. For bigger bandwidths, we can

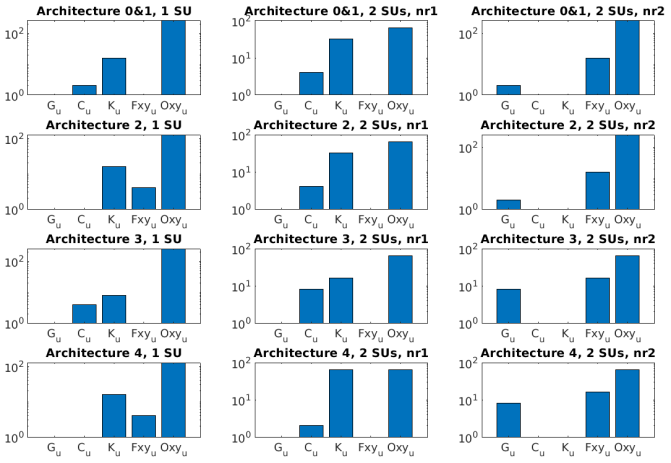


Fig. 3: Selected SUs for all architectures.

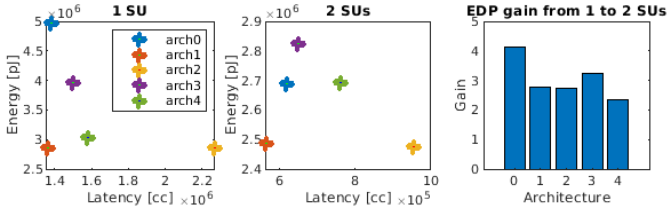


Fig. 4: Latency and energy results for 5 architectures.

afford to select more specific loop unrollings that are ideal for a given (sub)range of layers, leading to good optimization.

We also notice that C_u and K_u increase together with BW_I and BW_O, respectively.

C. Impact of memory hierarchy on TUs

Table III contains the inner-loop of the TUs for the SUs that are selected for the architecture template from Fig. 2b with $BW_W = BW_I = BW_O = 1024$ bits per clock cycle, for each different layer type.

TABLE III: Most selected inner TU loop for each SU for non-depthwise and depthwise layers.

SU	G_u	C_u	K_u	Fxy_u	Oxy_u	non-DW	DW
1 SU	1	1	16	4	128	for C	for Fx
2 SUs, nr 1	1	64	2	1	64	for K	for Fx
2 SUs, nr 2	1	2	4	16	64	for C	for Ox

The first SU has $K_u >> C_u$, leading to $O_{needed} >> I_{needed}$ and therefore the 'for C' loop is chosen over the 'for K' loop to reduce the required output bandwidth through temporal stationarity. For the second SU, it's exactly the other way around. We saw before that for small BW_I, $K_u >> C_u$, leading to a 'for C' loop and therefore output stationarity. Every clock cycle, a small number of input channels is read to compute intermediate results for many output channels. This is even true for architectures where $BW_I = BW_O$. In our hypothesis, this is due to the double resolution with which intermediate results are stored. For depthwise layers, a 'for Fx' loop is selected if Fx_u and Fy_u are not equal to 4 (because that has the lowest BW requirements) and a 'for Ox' loop otherwise (as that is still better than a 'for G' loop).

In general, we see that loops that are heavily parallelized in the SU are unlikely to be the most inner loop of the TU (except for depthwise layers).

V. Conclusion

Transformer networks recently showed great performance in machine learning domain but are hard to execute efficiently in hardware due to their wide variety in layer topologies, for which an architecture using a single hardware parallelization configuration with 1 spatial unrolling and temporal unrolling is not sufficient. This paper, therefore, analyzes optimal combinations of SUs and according TUs for transformer networks. Experiments show that larger PE arrays benefit more from the use of more than 1 SU, as the shared loop parameters across all layers become relatively small in comparison with the total number of PEs. Additionally, the impact of the memory hierarchy on the selected SUs and TUs is assessed. Results clearly show that architectures with more severe bandwidth constraints, benefit from unrollings with bandwidth-savings SU loops (more Fx_u/Fy_u, less G_u), at the expense of a lower utilization for several layers. For bigger bandwidths, we can afford to select more specific loop unrollings that are ideal for a given (sub)range of layers, leading to good optimalization. We also showed that for a given SU, the optimal according TU depends on the bandwidths of the memories of the three operands. In practice, loops that are heavily parallelized in the SU are unlikely to be the most inner loop of the TU (except for depthwise layers). Due to the raising complexity of new neural networks and increasing dimensions of hardware architectures, the methodology to select a good set of unrollings to support will become increasingly important.

The code of the extended COAC version and for all experiments in this paper will be made available on www.github.com/StevenColleman1234/AICAS2022.

Acknowledgment

This work has been realized with support of KU Leuven, the Flemish Government (AI Research Program), the EU ERC project Re-SENSE under grant agreement ERC-2016-STG-715037, and Oppo.

References

[1] S. Mehta and M. Rastegari, "Mobilevit: Light-weight, general-purpose, and mobile-friendly vision transformer," *arXiv preprint arXiv:2110.02178*, 2021.

[2] H. Kwon, L. Lai, M. Pellauer, T. Krishna, Y.-H. Chen, and V. Chandra, "Heterogeneous dataflow accelerators for multi-dnn workloads," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 71–83.

[3] S. Colleman, M. Shi, and M. Verhelst, "Hyper-flexible single core cnn execution," in *arXiv*.

[4] L. Mei, P. Houshmand, V. Jain, S. Giraldo, and M. Verhelst, "Zigzag: Enlarging joint architecture-mapping design space exploration for dnn accelerators," *IEEE Transactions on Computers*, 2021.

[5] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 14, no. 2, pp. 1–25, 2017.