VersaTile Convolutional Neural Network Mapping on FPGAs

A. Muñío-Gracia, J. Fernández-Berni, R. Carmona-Galán and Á. Rodríguez-Vázquez
Instituto de Microelectrónica de Sevilla, IMSE, CNM (CSIC, Universidad de Sevilla)

***Abstract*—Convolutional Neural Networks (ConvNets) are directed acyclic graphs with node transitions determined by a set of configuration parameters. In this paper, we describe a dynamically configurable hardware architecture that enables data allocation strategy adjustment according to ConvNets layer characteristics. The proposed flexible scheduling solution allows the accelerator design to be portable across various scenarios of computation and memory resources availability. For instance, FPGA block-RAM resources can be properly balanced for optimization of data distribution and minimization of off-chip memory accesses. We explore the selection of tailored scheduling policies that translate into efficient on-chip data reuse and hence lower energy consumption. The system can autonomously adapt its behavior with no need of platform reconfiguration nor user supervision. Experimental results are presented and compared with state-of-the-art accelerators.**

***Index Terms*—Hardware acceleration, FPGA implementation, Convolutional Neural Networks.**

I. Introduction

Society experiences the unceasing arrival of *intelligent* embedded systems directed at facilitating daily life by the utilization of Deep Learning (DL) algorithms. Integration of both sensor and processor components in those devices creates standalone autonomous platforms whose energy consumption must remain low, while offering maximum performance at a reduced cost. FPGAs compete against ASIC, GPU and CPU alternatives serving as a balanced solution in this scenario. ConvNets have become mainstream in image and video processing due to their accuracy, including a series of compact models specially suitable for embedded platforms [1]. By means of a reduced number of MAC operations and lessened memory footprint, these networks enable lower hardware utilization when compared with bigger models as VGG [2].

The constant evolution of ConvNet models invites for accelerators to be adaptable in order to cope with novel network structures and not to get obsolete. An unsupervised readjustment in their operation facilitates system independent response, critical in applications as IoT that require of no direct human guidance. Acceleration platform adaptation to each layer of a ConvNet is confined by hardware available resources. One way to address this obstacle is to exploit full and partial reconfiguration capabilities of FPGA devices [3], [4]. Nonetheless, those solutions introduce a bitstream reload time overhead that lasts in the order of hundreds of $\mu secs$ to *msecs* based on bitstream size and operating frequency. The alternative is to design the accelerator hardware so its structure and data movement policy can be revised *on the fly* by the modification of a set of configuration registers.

```
1  for(n=0; n<N; n++){      // L0 - Number of filters
2  for(y=0; y<H; y+=S){     // L1 - In fmaps (Height)
3  for(x=0; x<W; x+=S){     // L2 - In fmaps (Width)
4  for(c=0; c<C; c++){      // L3 - In fmaps (Channel)
5  for(ky=0; ky<K; ky++){   // L4 - Kernel (Height)
6  for(kx=0; kx<K; kx++){   // L5 - Kernel (Width)
7   O[n][x][y] = I[c][x+kx][y+ky]*W[n][c][kx][ky]
8  }}}
9   O[n][x][y] = O[n][x][y] + bias[n]
10 }}}
```

Fig. 1: ConvNet inference computation loop decomposition

ConvNet accelerator performance relies on data movement optimization, governed by *data compression* and *data allocation* techniques. A reduction in model data size can be further developed by the variable precision enabled by FPGAs as fine-grained computation devices. An aggressive quantization (even binarization) in both feature maps (*fmaps*) and filters (*weights*) translates into lower memory requirements [5]. Ideally, this removes the need of energy expensive intermediate off-chip data transfers. The diversity in inter-layer dimensions invites for data mover schedulers to be versatile so that on-chip memory accesses are balanced and parallelizable. Memory bandwidth needs to guarantee computation nodes with enough data for the system not to remain in an idle state that reduces overall efficiency. Generally, ConvNet accelerators computation blocks are organized in Network-on-Chip (NoC) topologies composed of Processing Elements (PEs) linked by router blocks [6]–[11]. The flexibility in this structure facilitates per-layer specialization of memory-to-computation-nodes fmaps and weights transfer for higher data reuse. The translation of network topologies to specific hardware constrained devices prompts for the study of ConvNet model partitioning strategies from their loop representation presented in Fig. 1. Loop optimization techniques as tiling, unrolling, and folding maximize fmaps concurrent computation [9]–[11].

This work proposes a scheduler architecture at register transfer level (RTL) that adapts its behavior to dynamically configurable tiling scenarios. The scheduling blocks arbitrate not only mutual data transfer between global buffer (GLB) and NoC, but also extend blocking configurability to the storage pattern on FPGA memory elements. The necessary parameters to define ConvNet structure (P_{CNN}), hardware resources (P_{RTL}), and user selectable tiling strategy (P_{USR}) are enumerated and linked with each of the scheduler modules based on their dependency relationship. Acceleration performance is benchmarked by SqueezeNet v1.1 ConvNet execution.

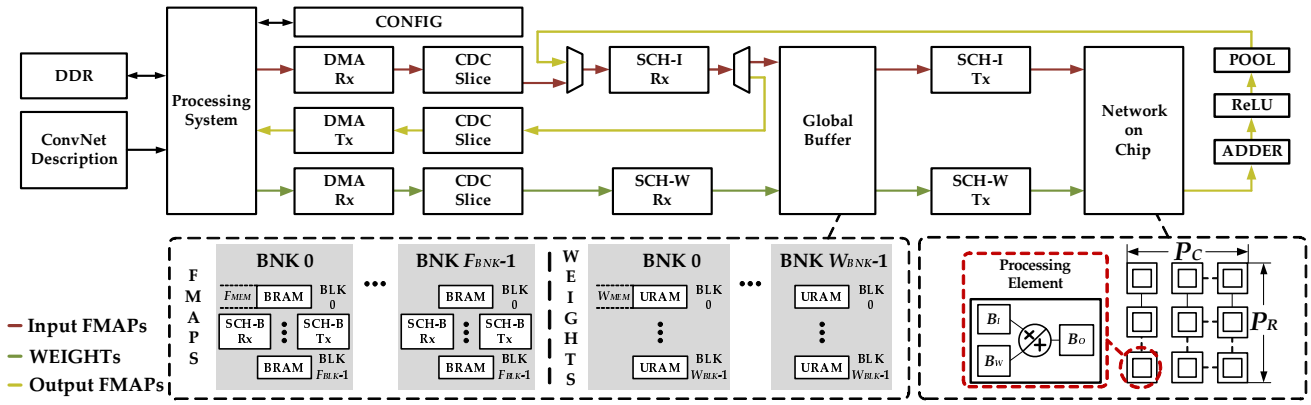

Fig. 2: Block diagram of the proposed ConvNet accelerator

II. ConvNet model mapping in FPGAs

The predominant hardware limiting factors in ConvNet FPGA accelerator designs are on-chip memory and DSPs. On-chip memory comprises both GLB, and PE local buffers employed to increase data reuse through data locality. DSPs availability limits the number of PEs, arranged in a $P_R \times P_C$ mesh as illustrated in Fig. 2 lower-right corner. P_{RTL} consist of NoC dimensions, data precision, and memory structure.

The convolution product calculation requires to decide of a data distribution scheme, namely *dataflow*, so that ConvNet operands are mapped onto hardware. Independence in fmap and weight datapaths endows the design with multiple degrees of freedom regarding data movement. The impact of dataflow choice on energy efficiency is thoroughly explored, concluding that in case of carefully selected tiling strategies are applied, power consumption remains constrained [11]. To balance data reuse and throughput is the final objective of the design [6].

At a structural level, *convolutional* layers are often followed by *pooling* layers, used to down-sample output fmaps by the application of average or maximum filters. The parallel calculation of *H* and *W* fmaps by unrolling loops {L1, L2, L4} over P_R and P_C allows for direct pooling with no need of complete layer computation and later GLB data refetch. In case multiple *H*×*W* tiles fit on the NoC, *replication* can be used to compute them concurrently. The unrolling of loop L0 over P_C increases filter data reuse while replication in P_R speeds up their transfer from global buffer. Loops {L3, L5} are unrolled in PE buffers *BI* and *BW*. Output fmaps stored at *BO* buffers can be parallelly flushed out in case accumulation over P_C is required, what happens for a kernel size greater than one. This dataflow is commonly referred as output stationary [8].

The subset of viable dataflow patterns is restricted by P_{RTL} and P_{CNN} parameters. On top of those, P_{USR} parameters modify how computed data are tiled into GLB memory blocks to better cope with variability in *H*, *W*, and (*C*, *N*) while moving through ConvNet layers. Table I gathers both P_{CNN} and P_{USR} that accelerator has to load on a per layer basis.

Independent datapaths exist to transport input/output fmaps and weights. The separation of these datapaths facilitates to employ different communication patterns to transfer each between GLB and NoC nodes. The decision of an specific workload transfer pattern (unicast, multi-cast, broadcast, ...) entails the acceptance of a compromise between data reuse and bandwidth [8]. In particular, our design uses a broadcast network for weights, and 1D-Multicast, 1D-Systolic networks for input and output fmaps respectively. Scheduler modules (SCH) are responsible for the connection among NoC nodes and GLB storage. For the sake of simplicity, no multiple in-parallel schedulers are used in this design, hence connections are either many-to-one or one-to-many with previous and subsequent stages. When input fmaps or weights exceed those that can be spatially mapped in NoC PEs, temporary loop partitioning (*folding*) needs to be applied. In this paper, parameter decomposition into partitioned components adopts the following nomenclature: P_B unrolling, P_A folding, and P_M replication. P_X is used to refer to any of them indistinctly.

The dependence relationship between each SCH and the ConvNet parameters that dictate their behavior is conditioned by both datapath and position between blocks (Tx, Rx) together with selected dataflow as in Table II. Because weights are trained off-chip, SCH-W Rx operation can be simplified by the assumption of an ordered input filter datastream.

TABLE I: ConvNet accelerator dynamic parameters

P_{CNN}		P_{USR}
H/W : Height/Width	K : Kernel size	$T_{H(Tx,Rx)}$: GLB Tile H
C : Input channels	S : Stride	$T_{W(Tx,Rx)}$: GLB Tile W
N : Output channels	PA : Padding	$T_{C(Tx,Rx)}$: GLB Tile C
CT : Concatenate layer	PO : Pooling	

TABLE II: Schedulers parameter dependency

Scheduler	Unrolling order based on dataflow (no replication)
SCH-I Tx	$C_B \rightarrow W_B \rightarrow H_B \rightarrow C_A \rightarrow N_A \rightarrow W_A \rightarrow H_A$
SCH-W Tx	$C_B \rightarrow K \rightarrow N_B \rightarrow C_A \rightarrow N_A$
SCH-I Rx	$W_B \rightarrow H_B \rightarrow C_B \rightarrow C_A \rightarrow W_A \rightarrow H_A$
SCH-B Tx	$T_{CB} \rightarrow T_{WB} \rightarrow T_{HB} \rightarrow T_{CA} \rightarrow T_{WA} \rightarrow T_{HA}$
SCH-B Rx	$T_{WB} \rightarrow T_{HB} \rightarrow T_{CB} \rightarrow T_{CA} \rightarrow T_{WA} \rightarrow T_{HA}$

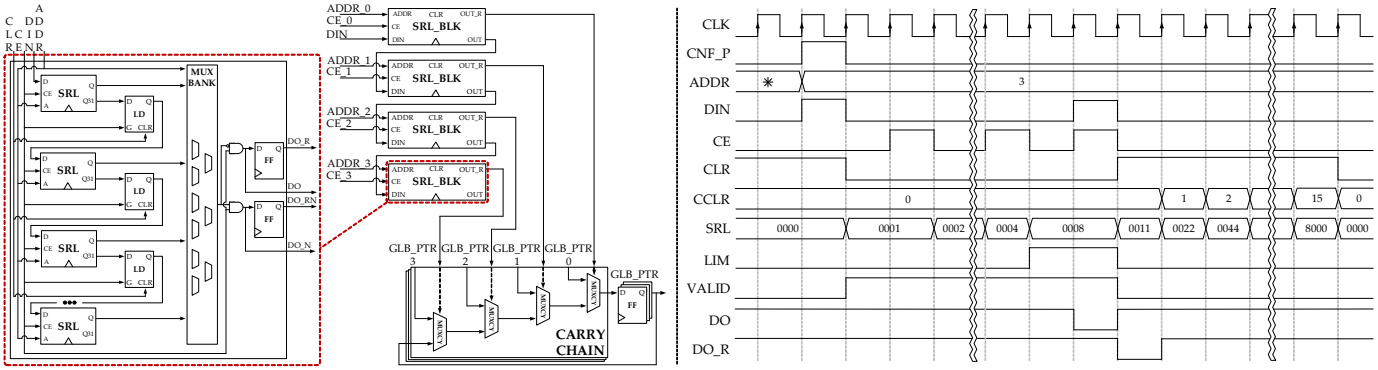

Fig. 3: Scheduler RTL implementation and SCH_BLK operation chronogram

III. Dynamic scheduler RTL implementation

The scheduler block is composed of a chain of cascaded shift-registers (SRLs) each of them S_D bits deep conditioned by FPGA platform. The total number of chained SRLs is calculated from the P_X parameter associated with each SCH_BLK as $\lceil max(P_X)/S_D \rceil$. Transparent latches can be placed between SRLs to reduce the number of cycles required to fully flush out data from $\#SRL\_BLK \times S_D$ to S_D in exchange of higher area. SRLs allow for dynamic depth adjustment by the modification of their ADDR value, which is updated after the calculation of an unrolled tile. SRLs clear out period is overlapped with inference computation and triggered by the last block in the chain in P_B cases, or individually for each P_A element after their DO pulse. The internal operation of SRL_BLK is epitomized at the chronogram in Fig. 3

Hierarchical organization of GLB is in *BNK→BLK→MEM* levels. Shape selection of RAM matrices for both fmaps and weights has to distribute blocks (*BLK*) that belong to a bank (*BNK*) in the same clock region to avoid increased access latency due to pipelining. RAM depth (*MEM*) is purely platform dependent. Data precision (D_F,D_W) establishes the number of RAMs (D_B wide) that have to be accessed simultaneously to retrieve a data word. Table III summarizes memory configuration details where fmaps are stored in block-RAMs and weights in ultra-RAMs, as this work has been developed on a Zynq 104 MPSoC. SRL_BLK concatenation builds the core of the scheduler shown in Fig. 3, they are used to select among GLB address pointers handled by priority multiplexers implemented with $\log2[(F,W)_{MEM}]$ CARRY chains.

Provided with a ConvNet description, the Processing System (PS) calculates P_X parameters which demand of a total of 6 AXI4-Lite words (192b) to describe a layer. Thus full system configuration for a 26-layer ConvNet as SqueezeNet v1.1 can be transferred in 780 ns at a 200 MHz clock frequency.

TABLE III: FPGA memory related parameters

PE (16b/8b/16b)			FMAPs - block-RAM			WEIGHTs - ultra-RAM		
BI	*BW*	*BO*	*BNK*	*BLK*	*MEM*	*BNK*	*BLK*	*MEM*
32	32	64	64	1	8192	12	4	4092

IV. Flexible On-Chip Memory Allocation

Memory allocation flexibility must be guaranteed to cope with ConvNets inter-layer workload variation that otherwise could ruin accelerator performance. Dataflow strategy secures data reuse at lower hierarchy memory blocks (PE buffers) so the number of accesses to next memory level (GLB) is optimized [10]. There exists the requirement for data at GLB to be allocated in a pattern that leverages available block-RAM resources while offering enough parallelism not to become a bandwidth bottleneck. By updating P_{USR}, SCH-B (Tx, Rx) manipulate tile shape for a given $\{H, W, C, N, P_{RTL}\}$ configuration. On-the-fly rewrite of the fmaps GLB with calculated inference products involves higher complexity in terms of data routing than for the weights GLB, thus this work focuses on their allotment.

Calculation of fmaps during inference results in a dynamic memory utilization that needs to be characterized before computation so that write/read pointers collision is eluded. The number of output fmaps calculated after N_A epochs is determined by dataflow strategy as detailed in Section II:

$$D_{OF} = (W_B/S) \cdot (H_B/S) \cdot N \tag{1}$$

which enables to overwrite the following number of input fmaps provided that there are no more concatenated layers:

$$\begin{aligned} D_{IF} = (W_B - \overline{W_{A,LIM}} \cdot (K-1)) \\ \cdot (H_B - \overline{H_{A,LIM}} \cdot \lfloor K/2 \rfloor) \cdot C \end{aligned} \tag{2}$$

A pessimistic approach is to consider that all input and output fmaps need to coexist in memory at the same time ignoring intermediate overwrites. However, this adds a memory usage overhead that translates into misleading off-chip data transfer expectations. A better approach is to incrementally model in-memory data, so at iteration ($H_t \in [0, H_A]$, $W_t \in [0, W_A]$):

$$\begin{aligned} D_T(H_t, W_t) = H \cdot W \cdot C + \sum_{lc}^{\#CT} \left(\frac{W_{lc}}{S_{lc}} \cdot \frac{H_{lc}}{S_{lc}} \cdot N_{lc} \right) \\ + \sum_{i=0}^{H_t} \sum_{j=0}^{W_t} \left(D_{OF}(i,j) - \overline{CT} \cdot D_{IF}(i,j) \right) \end{aligned} \tag{3}$$

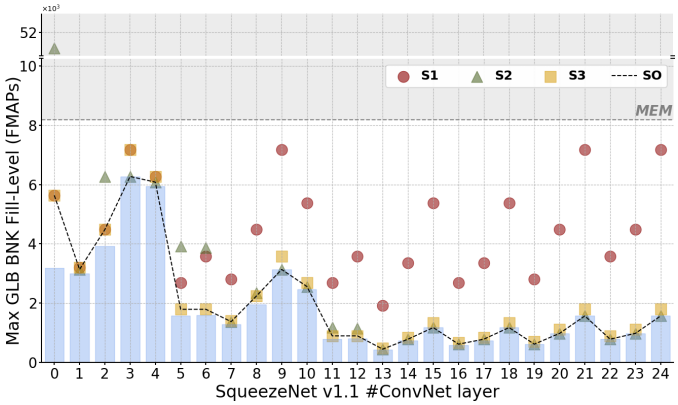

Fig. 4: GLB memory bank peak utilization depending on tiling

V. Experimental Results

The storage of output fmaps in consecutive GLB positions achieves best mapping efficiency as writes span the full address space. However, to reconstruct model information during the computation of the next layer entails severe difficulties because of NoC mapping uneveness. In this work we explore model division into tiles sized according to configuration parameters as follows:

- **S1**: Unrolls H over BNK not partitioning W nor C. Shows better mapping efficiency for those layers where $H \geq BNK$, hence worsens while going deeper into the network.
- **S2**: Unrolls C over BNK not partitioning H nor W. Shows better mapping efficiency for those layers where $C \geq BNK$, hence worsens at input and initial squeeze layers.
- **S3**: Combines **S1** and **S2** by partially unrolling both H and C over BNK. The increased granularity of this approach leads to a closer to optimal mapping efficiency.

Because SqueezeNet receives input data in a square format, to interchange H and W in former strategies has no impact in performance. Fig. 4 depicts the deviation of the examined strategies (**S1**, **S2**, **S3**) with respect to the maximum bank memory required by a completely balanced strategy, in blue bars. Off-chip fmaps data transfer is required in those cases where peak memory utilization exceeds *MEM* which happens in **S2**. By the combination of best performing strategies at each layer (**SO**) it is possible to fully avoid external memory accesses, thereby reducing inference time.

Table IV summarizes the performance and hardware utilization of our solution, which are compared with other state-of-the-art designs that use same FPGA technology. Accelerator reported specifications are for a P_R=32, P_C=24 Network-on-Chip whose PE utilization efficiency only drops in deeper ConvNet layers as replication is not yet fully supported. A total of 56 LUTRAMs are required to implement the scratchpad memories at each PE. Because of routing congestion, NoC size limits achievable operating frequency. System scalability allows to adjust the number of PEs so the design is portable to other FPGAs, in addition to reach a compromise between maximum frequency and number of computation nodes.

TABLE IV: Comparison with other ConvNet FPGA designs

	[7] VGG-16	[11] Various CNNs	This work SqueezeNet
CNN Size (GOP)	15.36	0.57-15.36	0.78
FPGA	Kintex XCKU060	Zynq 102 XCZU9EG	Zynq 104 XCZU7EV
Freq. (MHz)	200	200	200
DSPs	1058 (38%)	1030 (40%)	768 (44%)
LUTs	100K (31%)	39K (7%)	152K (66%)
Block-RAM	782 (36%)	1670 (91%)	281.5 (90%)
Ultra-RAM	N/A	0	48 (50%)
Through. (GOPs)	266	185	167.5

VI. Conclusions

An RTL scheduler architecture that arbitrates data movement between GLB and NoC components of a ConvNet accelerator has been presented. Its adaptability allows for tuning data allocation patterns not only for different networks but also to cope with inter-layer features diversity. Accelerator performance has been demonstrated by the computation of SqueezeNet on Zynq 104 FPGA and is competitive when compared with other designs.

Acknowledgment

This work was supported by EU H2020 MSCA through Project ACHIEVE-ITN (Grant No 765866), by the Spanish MINECO and European Region Development Fund (ERDF/FEDER) through Project RTI2018-097088-B-C31 and by the US Office of Naval Research through Grant No. N00014-19-1-2156.

References

[1] F. N. Iandola *et al.*, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size," *CoRR*, 2016.
[2] S. Liu and W. Deng, "Very deep convolutional neural network based image classification using small training sample size," in *2015 3rd IAPR Asian Conf. on Pattern Recognition (ACPR)*, Nov 2015, pp. 730–734.
[3] K. Abdelouahab *et al.*, "Tactics to directly map CNN graphs on embedded fpgas," *CoRR*, vol. abs/1712.04322, 2017.
[4] M. Putic *et al.*, "Dyhard-dnn: Even more dnn acceleration with dynamic hardware reconfiguration," in *Proc. of the 55th Annual Design Automat. Conf.*, ser. DAC '18, 2018, pp. 14:1–14:6.
[5] M. Blott *et al.*, "FINN-R: an end-to-end deep-learning framework for fast exploration of quantized neural networks," *CoRR*, vol. abs/1809.04570, 2018.
[6] M. Peemen *et al.*, "Memory-centric accelerator design for convolutional neural networks," in *ICCD*, Oct 2013, pp. 13–19.
[7] C. Zhang *et al.*, "Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks," in *2016 IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD)*, Nov 2016, pp. 1–8.
[8] Y. Chen *et al.*, "Architecture design for highly flexible and energy-efficient deep neural network accelerators," Ph.D. dissertation, Massachusetts Institute of Technology, 2018.
[9] Y. Ma *et al.*, "Optimizing loop operation and dataflow in fpga acceleration of deep convolutional neural networks," in *Proc. of the 2017 ACM/SIGDA International Symposium on FPGAs*, 2017, pp. 45–54.
[10] A. Stoutchinin *et al.*, "Optimally scheduling CNN convolutions for efficient memory access," *CoRR*, vol. abs/1902.01492, 2019.
[11] X. Yang, M. Gao, J. Pu, A. Nayak *et al.*, "DNN dataflow choice is overrated," *CoRR*, vol. abs/1809.04070, 2018.