2022 21st International Symposium on Parallel and Distributed Computing (ISPDC) | 978-1-6654-8802-0/22/$31.00 ©2022 IEEE | DOI: 10.1109/ISPDC55340.2022.00014

Coarse-Grained Floorplanning for streaming CNN applications on Multi-Die FPGAs

Danielle Tchuinkou Kwadjo
ECE Department
University of Florida
Gainesville FL, USA
dtchuinkoukwadjo@ufl.edu

Erman Nghonda Tchinda
ECE Department
University of Florida
Gainesville FL, USA
enghonda@ufl.edu

Christophe Bobda
ECE Department
University of Florida
Gainesville FL, USA
cbobda@ece.ufl.edu

***Abstract*—With the vast adoption of FPGAs in the cloud, it becomes necessary to investigate architectures and mechanisms for the efficient deployment of CNN into multi-FPGAs cloud Infrastructure. However, neural networks' growing size and complexity, coupled with communication and off-chip memory bottlenecks, make it increasingly difficult for multi-FPGA designs to achieve high resource utilization. In this work, we introduce a scalable framework that supports the efficient integration of CNN applications into a cloud infrastructure that exposes multi-Die FPGAs to cloud developers. Our framework is equipped is with two mechanisms to facilitate the deployment of CNN inference on FPGA. First, we propose a model to find the parameters that maximize the parallelism within the resource budget while maintaining a balanced rate between the layers. Then, we propose an efficient Coarse-Grained graph partitioning algorithm for high-quality and scalable routability-drive placement of CNN's components on the FPGAs. Prototyping results achieve an overall 37% higher frequency, with lower resource usage compared to a baseline implementation on the same number of FPGAs.**

***Index Terms*—CNN acceleration, FPGAs, Distributed inference, FINN**

I. Introduction

In recent years, the implementation of Convolutional Neural Networks (CNN) on Field Programmable Gate Arrays (FPGAs) has drawn considerable attention as the need for more efficiency and accuracy is mitigated by the rapid increase in computational cost. CNNs achieve a higher quality of result (QoR) at the cost of significant computing and memory requirements due to their deep topological structures, complicated neural connections, and massive data to process [1], [2].

As the performance need of applications and systems increases and the power budget steadily decreases, the architecture-level response trends toward hardware specialization. Specialized hardware accelerators such as graphics processing units (GPU) have long been the default solution to accelerate workloads. However, the performance/watt advantage, flexibility, and reprogrammability of Field-Programmable Gate Arrays (FPGA) raise the interest of both academia and industry. While the industry previously focused on enabling developer-friendly CAD tools that can generate high-performance accelerators, using FPGAs within the stack of applications running in the cloud is emerging as a rising trend. As a response, FPGA devices have been recently introduced in cloud infrastructure to provide acceleration for critical workloads in machine learning. With the continuous growth of learning models, it becomes critical to distribute the inference's workload among multiple devices.

Multiple CNN architectures on FPGA have been proposed in the literature. They can be reviewed in two categories: *Single Instruction, Multiple Data (SIMD) accelerators* and *streaming-based accelerators*. This approach is flexible and is defined as a general-purpose accelerator, as it likely supports several CNN topologies. However, it is inefficient since it requires frequent memory transfer between FPGA on-chip scratchpad memories and external memory (DDR/HBM) to fetch the weights and activations. Accelerators with the streaming architecture have a layer-by-layer execution flow [3]–[5]. The main advantage of this type of architecture is to minimize the latency caused by communication with off-chip memory and thereby maximize on-chip memory communication, ensuring high throughput and avoiding any latency [6]. On the downside, this accelerator architecture cannot scale to arbitrarily large CNNs.

This work focuses on the design automation of pipelined CNNs inference on multi-die FPGA cloud platforms. We propose a framework that integrates a performance exploration tool on FINN-based accelerators [7] to find the parameters that maximize the parallelism within the resource budget while maintaining a balanced rate between the layers. Our framework enables automatic graph partition and design generation of sub-graphs, with a higher area utilization and improved productivity, and ensures minimal latency for CNNs. Our proposed changes can improve the performance of FINN and provide an efficient streaming implementation for FPGAs in data centers. Specifically, the contribution of this paper include:

- We propose an accurate model to find the optimal parameters for the configuration to assess the resource consumption and timing for streaming accelerators.
- An end-to-end framework that maps CNN models to FPGA implementations without tedious HDL programming and verification while improving the Quality of Result (QoR) compared to the traditional design flow with Vivado.
- An effective and efficient Coarse-Grained floor planning

978-1-6654-8802-0/22/$31.00 ©2022 IEEE
DOI 10.1109/ISPDC55340.2022.00014

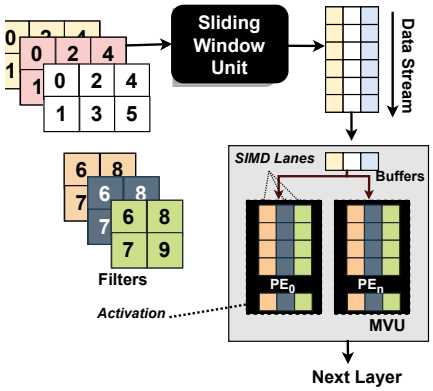


Fig. 1: FINN architecture. SWU interleaves the input by applying the image-to-column algorithm and feeds MVTU.

algorithm for high-quality and scalable routability-drive placement of components on FPGA.

II. Background

CNN inference consists of the sequential execution of M input images through L layers. Accelerators with the streaming architecture always tailor the hardware with respect to the target network [3], [4] to generate a custom architecture. The topology of such CNN accelerators is transformed into a layer-by-layer execution schedule, following the structure of the inference graph in the form of a DAG[1] [8]. The main drawback of this paradigm is resource limitations due to its spatial-processing nature. Regarding Could-based platforms, streaming dataflow architectures are preferred as the target platform theoretically provides unlimited resources.

A. FINN Architecture

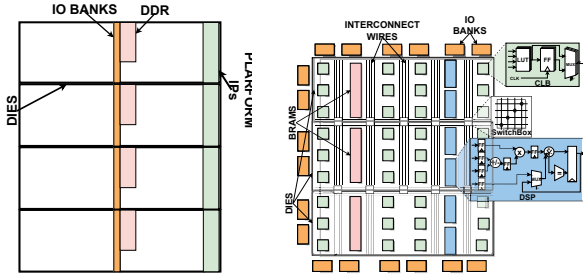
FINN enables the design of heterogeneous custom streaming architecture for a given topology rather than scheduling operations on a fixed architecture. Separate compute engines are dedicated to each layer, communicating via on-chip data streams. Each engine starts to compute as soon as the previous engine produces output. It currently supports fully connected, convolutional, and pooling layers. An overview of the FINN architecture is depicted in Figure 1. It has two main units: the Sliding Window Unit (SWU) and the matrix-vector unit (MVU).

The SWU supplies the convolution engine with the image matrix from the incoming feature map by applying interleaving and implementing the im2col algorithm. The computational core of the compute engines is the MVU, as the vast majority of computing operations in neural networks can be expressed as matrix-vector operations. An MVU computes the matrix-matrix product using a different column vector from the image matrix stream. The MVU consists of an input and output buffer and an array of Processing Elements (PEs), each with a number of SIMD lanes. The number of PEs (P) and SIMD lanes (S) is configurable to regulate the throughput and controls the folding of matrix-vector products to achieve a given FPS requirement set by the user. A PE performs a number of parallel multiplications equal to the SIMD value. It then reduces them in an adder tree for their subsequent accumulation towards the computed dot product. Finally, threshold comparisons derive the output values from the accumulation results.

[1]Data Acyclic Graph

B. Multi-Die FPGA Architectures

The multi-die FPGA is only present on devices that use the stacked Silicon Interconnect Technology (SSIT), also known as 2.5D packaging, using a silicon interposer. Each die becomes a super logic region or SLR as multiple dies are packaged together. SLRs contain a 2D array of FSRs and are typically identified as each die is fabricated from the same mask set. For logic to communicate between SLRs, the UltraScale architecture employs special tiles in the FSRs neighboring the abutment of two SLRs, which incurs additional signal delay. A column of CLBs is removed and replaced with special tiles called Laguna tiles with dedicated flip flop sites to aid in crossing the SLR divide. Additionally, the IPs on-chip consume significant programmable resources near their fixed locations that may also cause local routing congestion.



(a) XCU250 (b) XCVU9p

Fig. 2: Block diagrams of two representative FPGA architectures: the Xilinx Alveo U250,(based on the Xilinx UltraScale+), and a Virtex Ultrascale+ architecture

RapidWright [9]: is an open-source Java framework from Xilinx Research Labs that provides a bridge to Vivado backend at different compilation stages (synthesis, optimization, placement, routing, etc.) using design checkpoint (DCP) file. Once a DCP is loaded within RapidWright, the logical/physical netlist data structures and functions provided in the RapidWright APIs enable custom netlist manipulations such as cell and net instantiation, edition, and deletion. By making available logical/physical netlist data structures and functions, it enables custom netlist manipulation and direct access to logic and routing resources such as look-up tables (LUT), flip-flops (FF), and programmable interconnect points from a Java API.

C. problem formulation

Several works [6], [10]–[12] in the literature employ FINN to generate NN accelerators on FPGAs. Nevertheless, FINN-based accelerators' area consumption and parallelism parameters cannot be arbitrarily deduced. Since the performance of an accelerator is bounded by the slowest component within the design, finding the parameters to generate a balanced design can be a bottleneck. In this work, we propose an accurate model to find the optimal parameters for the configuration to assess the resource consumption and timing for FINN accelerators. We also study the granularity of composing the final accelerator and the partitioning of the computational graph in a multi-FPGA platform.

III. Proposed Framework

This section presents our coarse-grained floorplanning scheme. We assume that HLS maintains the source code's hierarchy, and each function in the HLS source code will be compiled into an RTL module. Functions communicate with each other through FIFO channels. Our focus is not on improving floorplanning algorithms; rather, we intend to use coarse-grained floorplan information to guide the placement properly. The proposed framework is depicted in Figure 3.

TABLE I: Notations

Name	Description
$G = (V, E, \omega, \phi)$	Graph G with a set of Vertices V, edge set E, vertices weights ω, edges weight ϕ. Edges are FIFO channels between vertices.
i, N	Index of a vertice, $\|V\|$
LUT_i	LUT capacity of the SLR_i.
FF_i	Flip-flop requirement of the SLR_i.
$BRAM_i$	BRAM capacity of the SLR_i.
DSP_i	DSP capacity of the SLR_i.
IFM_{DIM_i}	Dimension of the Input feature maps.
K_i	kernel size
IFM_{CH_i}	Number of channels of the input layer.
OFM_{CH_i}	Number of channels of the output layer.

(1) **Computational Graph**: First, it takes as input the computational graph at the module level of an input program. The vertices weight represents the computational workload of each module, and the edge weight is the local memory ratio, which is the amount of data (in Kb) that moves between adjacent nodes.

(2) **Platform Description**: The FPGA resources are represented as a directed dataflow graph (DFG) in which each node represents the resources of each SLR, and the edges denote the communication latency between the SLRs regardless of the physical FPGA from which they are provisioned. When FPGAs are added or removed from the platform, only the DFG needs to be updated. Users can also define how many levels of pipelining to add based on the number of boundary crossings. By default, we add two levels of pipelining to the connection for each boundary-crossing. In this work, we only use a Peripheral Component Interconnect Express (PCIe) connection between the FPGAs, but the architecture can also accommodate network interfaces.

(3) **Performance Exploration**: Given the platform description resources and the inference graph, the framework explore the parameters that will maximize the throughput given the resources budget of the FPGAs. Additionally, developing high-performance hardware accelerators on FPGA often demands skills in hardware design and long development cycles. By pre-implementing the modules of a design, higher performance can be achieved locally and maintained to a certain extent when assembling the final circuit. Furthermore, in the case of module replication, the pre-implemented designs can be reused, improving the engineering time.

(4) With the implementations and performance details (timing, floorplanning, workload), we define several constraints to guide the coarse-grained floorplanning at the module level.

A. Performance Exploration

To efficiently implement CNN inference on FPGAs, we seek by performing a design space exploration to find the optimal parameters that maximize the performances achievable by the CNN sub-functions such as Convolution, pooling, and fully connected layers (FC). It takes into consideration some design constraints such as the platform description, timing, and floor planning. If the design space exploration results in satisfactory performance, the produced netlists are saved as Design Checkpoint (DCPs).

To highlight the effect of the P and S on latency, let us consider the results presented in Figure 4. A higher level of parallelism implies a higher number of resources used. Each layer has a set of parameters (S, P) that control the degree of parallelism, which must be chosen such as: a balanced streaming pipeline, the desired performance, and the total resource footprint available within the given budget. Finding the right configuration can greatly impact the final results. Previous work has demonstrated that extensive automated search in the design space can identify accelerator configurations better than human designers. Regarding heterogeneous streaming architecture, the slowest layer will determine the overall throughput. The guiding principle is to implement rate-balancing [7] between the layers. So, each layer should use a roughly equal latency (expressed as clock cycles) to process one image.

*a) **Latency Constraints**:* For an inference model with N vertices and a platform with M SLRs, we seek maximize $\{S_i, P_i\}$ such that:

$$\begin{cases} Latency_i \simeq (1+\epsilon) \times Latency_{i+1} \quad \forall i = 1, ..., N \\ with\ Latency_i = F_i^n \times F_i^s \\ with\ F_i^n = \frac{OFM_{H_i} \times OFM_{W_i}}{P_i}, \\ and\ F_i^s = \frac{K_i^2 \times IFM_{Ch_i}}{S_i} \end{cases} \tag{1}$$

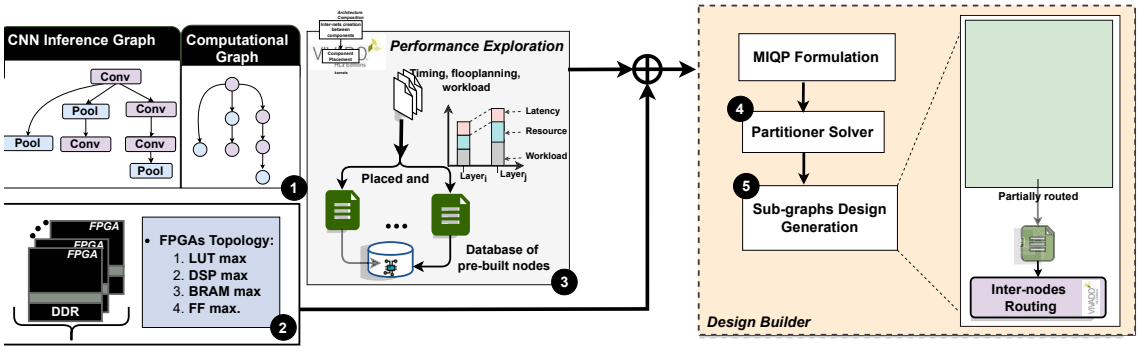


Fig. 3: Proposed Framework

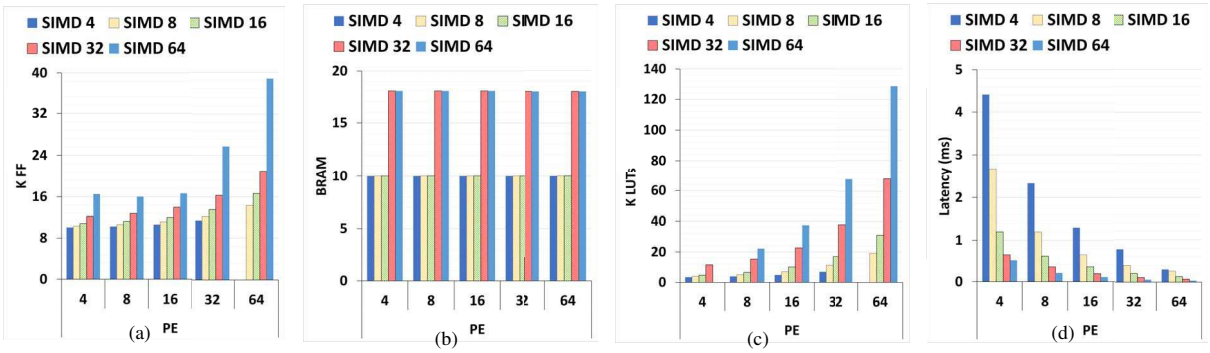


Fig. 4: Effect of the folding factor (PE/SIMD) over the performance and the resource utilization for a convolution. For a $SIMD > 32$ the BRAM utilization increases from 10 to 18, regardless of the number of PEs. A higher number of SIMD lanes leads to smaller latency, but with a higher resource utilization

As S and P porting the data packed to be processed by the MVU,

$$(K_i^2 \times IFM_{Ch_i}) \, mod \, S_i == 0$$
$$(OFM_{H_i} \times OFM_{W_i}) \, mod \, P_i == 0$$

ϵ is an imbalance factor, allowing a margin between different layers.

*b) **Variables Constraints**:* For a layer i, we denote by PS_x, the maximum value of P_i and S_i, and $\sigma_{i,p}$, a binary decision variable such that $\sigma_{i,p} = 1$ iff $P_i = x_{i,p}$, with $\forall x_p = 1, ..., 64$.

$$P_i = \sum_{p=1}^{PS_x} \sigma_{i,p} \times x_{i,p} \quad and \quad \sum_{p=1}^{PS_x} \sigma_{i,p} = 1$$
$$S_i = \sum_{p=1}^{PS_x} \gamma_{i,p} \times y_{i,p} \quad and \quad \sum_{p=1}^{PS_x} \gamma_{i,p} = 1$$
$$\epsilon_i = \sum_{p=1}^{2\times\epsilon*100} \delta_{i,p} \times z_{i,p} \quad and \quad \sum_{p=1}^{2\times\epsilon*100} \delta i, p = 1$$

With $z_{i,p}$ being the set of relaxing values. For example, if $\epsilon = 0.25 \; ns$, then a maximum difference latency of $+/- 0.25 \; ns$ is permitted between the latency of the layers. Hence $z_{i,p} \in [-\epsilon, \epsilon]$. With a maximum of two decimal numbers per relaxing factor, the search range is equal to $2 \times \epsilon * 100$.

*1) **Resources Constraints**:* The framework has to quickly estimate an accelerator's LUT, DSP, and OCM requirements from a given set of values of the parallelism variables. Design congestion can negatively impact the achievable frequency for any FPGA design. Hence, it is recommended to balance resource utilization between layers. A balance resource utilization should not exceed the maximum utilization of 70 % LUTs, 50 % FF, and 90% DSPs, Block, and Ultra RAM of total available resources. We express as $F_{t_i}(P_i, S_i)$, a linear function that estimates the the amount of resources of type t demanded by the ith layer for a given (P_i, S_i) configuration.

$$\begin{cases} \sum_{i=1}^{N} F_{lut_i}(P_i, S_i) \leq LUT_{VRs}, & \forall i = 1, ..., M \\ \sum_{i=1}^{N} F_{dsp_i}(P_i, S_i) \leq DSP_{VRs}, & \forall i = 1, ..., M \\ \sum_{i=1}^{N} F_{bram_i}(P_i, S_i) \leq BRAM_{VRs}, & \forall i = 1, ..., M \end{cases}$$

The values of the $F_{t_i}(P_i, S_i)$ are computed using the layer cost model as in [13].

B. Graph Partitioning

When deploying a single accelerator to a multi-FPGA Cloud Infrastructure, the role of the partitioner is to segment the computational graph into sets of modules and assign those

to the different FPGAs. As the sub-functions have been configured to fit within the resource budget, we only focus on having the minimum number of partitions to fit within the FPGAs' die. We proceed with a multi-way graph partitioning problem which consists of finding a k-balanced multi-partition of a graph $G = (V, E, \omega, \phi)$ that minimizes objective function over the cut nets for some value of ϵ.

*a) **Multi-level partitioning**:* we implement a recursive balanced bi-partitioning to generate the different partitions of the computational graph. More precisely, whenever a partition does not violate the constraints: **(1)** the partitions do not satisfy the FPGAs requirement in terms of resources, **(2)** The number of partitions is smaller than the number of FPGAs. We recursively bi-partition each sub-graph until one condition mentioned above does not hold anymore. In that case, we proceed to the refinement step. The weight of the heaviest partition is restricted by a fixed upper bound $U = \epsilon \times \frac{\omega(V)}{k}$, with ϵ representing the unbalanced factor since all partitions cannot have exactly the same weight, and $k \leq \#FPGAs$. Multi-level partitioning is a well-known problem in the literature, with several solutions available. Hence, we do not aim to elaborate further.

*b) **Refinement step**:* : For n iteration, a bi-partitioning will produce 2^n partitions, resulting in unbalanced partitions or too many partitions. The refinement step allows us to merge smaller partitions or further split heavier partitions (with $k \leq \#FPGAs$) to accommodate FPGAs resources.

C. Sub-graphs Design Generation

The sub-graphs generation function is to generate accelerators for the different partitions. We start by synthesizing the CNN sub-functions Out of Context (OCC). The OOC flow ensures that I/O buffers and global clock resources are not inserted into the netlists as those pre-built "modules" are still to be inserted within the top-level module of the design. The sub-function granularity are discussed in section IV-C. The sub-graphs designs are generated by stitching sub-functions netlists with RapidWright. This is achieved by creating interconnected nets between the ports of an adjacent module. In the next section, we discuss the placement of sub-graphs netlists.

D. Coarse-grained Floor planning

Given an Utrascale FPGA with logic elements, its architecture, and a graph G of modules, we need to map the module's netlist to the logic elements of the FPGA and determine their positions to minimize routed wirelength and congestion. In summary, (1) each module must be assigned to a valid position on the FPGA, and (2) the placement legalization rules of each tile are satisfied. To achieve high QoR in the implementation of modules, follow the following design considerations:

- **Strategic floorplanning:** utilizing pblock constraints allows to carefully select the FPGA resources that each design module will use. It helps improve the module-level performance and area. Given that Xilinx architectures generally replicate the resource structures over an entire column of clock regions, the smaller the area of a pblock is, the more our custom API will be capable of relocating the design modules across the chip, which increases the reusability. The automated definition of the pblock range is out of the scope of this work.
- **Strategic port planning:** the placement of the ports when pre-implementing modules are one of the most important steps to ensure high performance and productivity improvement. Failure to plan the location of the ports of the pre-implemented modules may result in long compilation time, poor performance, and high congestion in the design in which they are inserted.
- **Clock routing:** to accurately run the timing analysis on the OOC modules, source clock buffers must be specified using the constraint HD.CLK_SRC. Though the buffers are not inserted in the OOC modules, clock signals are partially routed to the interconnect tiles, and the timing analysis tool can then run timing estimations.
- **Logic locking:** Once a module attains a desirable performance (F_{max}, area, power, etc.), we lock the placement and routing to prevent Vivado from altering the design later and preserve design performance. The other advantage of locking the design is that the final inter-module routing with Vivado will only consider non-routed nets.
- **Checkpoint file generation:** pre-implemented modules are stored in the form of DCPs and can be reused.

*a) **Problem Formulation**:* : We define the problem of a multi-FPGA coarse-grained floorplanning as follows. Given a set of M with m rectangular modules, each module has a width and height denoted by $x_i, y_i,\ 1 \leq i \leq m$, respectively. The aspect ratio of a module AS_i is defined by $\frac{x_i}{y_i}$. Given a set of D FPGAs with k dies $D = \{d_1, d_2, ..., d_k\}$, where each die has the same width and height denoted by (D_{W_i}, D_{H_i}).

*b) **Solution**:* : We use a geometry-based floorplanning as each module can have a position in the 2D-dimensional space of the FPGA. The benefit of utilizing geometry is that we can compute distances between modules and use the geometric notion of distance to perform fast placement. Our problem is small as the number of vertices within a CNN inference graph is limited. Hence, we formulate the partitioning process of each iteration using integer linear programming (ILP). Each module has a w_i, h_i: width and height, and (x_i, y_i) is the lower-left corner of the module.

The algorithm works as follows: we recursively parse the temporally ordered sub-graph and place the first module. Since modules are pre-implemented within pblocks, the number of resources is reported. We assign a location on the FPGA grid for each adjacent module with minimal to interconnect wire length, i.e., the estimated half-perimeter wire length (HPWL) from the placed cell locations. To fulfill that requirement, we define timing and congestion cost functions to evaluate the cost of the assigned location. At each iteration, we want to assign each $v \in V$ in D following the constraints below:

(1) C_1: Modules must respect linear localization constraints;
(2) C_2: A geometry-based placement in respect to timing and congestion cost.

C_1: Two adjacent M_i and M_j are non overlapping. They can only be on below or above one another. We introduce two binary variables $p_{i,j}$ and $q_{i,j}$ to denote whenever M_i is below or above to a module.

$$y_i + h_i \leq y_j + H(1 + p_{i,j} - q_{i,j})$$
$$y_j - H(2 - p_{i,j} - q_{i,j}) \leq y_i - h_j$$

C_2: For optimal routing, a placement algorithm must consider the number of resources used by each inter-component net and their interaction. For instance, if all nets are limited to a relatively small portion of the chip area, the routing path request will probably be very high. Furthermore, the number of switch boxes to traverse factor into the total delay [14]. Furthermore, spreading the design across the device helps reduce local congestion. The algorithm tries to build a solution incrementally, one component at a time, removing those solutions that fail to satisfy the problem's constraints at any point in time. A placement is validated if the costs are lower than a defined threshold (Equation 3).

$$\sum_{i,j=1,i\neq j}^{k} h_i + HPWL(W_{i,j}) + cgt\_cost \leq Threshold$$

c) The timing cost: is defined by the wire length between two components.

$$timing\_cost = \sum_{i=1,i<j}^{n-1} HPWL(W_{i,j}) \tag{2}$$

Where $W_{i,j}$ is the wire between component i and j (distance from physical net's source pin to sink pin). A fan-out greater than one will, in most cases, have some branching farther (reusing a path). In this case, the unit of length is the dimension size of a tile.

d) The congestion estimation is defined as the following:

$$cgt_{coef} = \sum_{i} C_i$$
$$cgt\_cost = \sum_{W_{i,j}} \frac{\omega_{i,j} \times cgt_{coef_k}}{\#SwBox} \tag{3}$$

Where C_i is the number of components overlapping within a $tile_k$, $\omega_{i,j}$ is a weight proportional to the number of pins of wire $W_{i,j}$, and $\#SwBox$ is the number of switch boxes traversed by the nets.

IV. Experimental Results

A. Evaluation Platform and Setup

For evaluation purposes, designs are implemented on a Xilinx Kintex UltraScale+ FPGA (xcku5p-ffvd900-2-i). The hardware is generated using Vivado v2020.2 and RapidWright v2020.1, and the components are implemented with vivado HLS. The hardware generation is conducted on a computer equipped with an Intel Corei7-9700K CPU@3.60GHz×4 processor and 32GB of RAM. The performance exploration

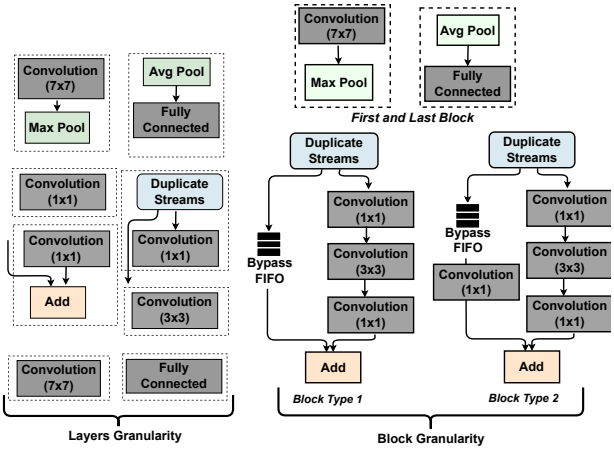


Fig. 5: Different modules granularity

stage is solved with LocalSolver [19] as it has demonstrated obtaining efficient results (optimality gap $< 10\%$) within seconds regardless of the size of the problem when compared to other Mixed-Integer Programming (MIP) solvers on NP-hard problems such as the quadratic assignment problem [20].

B. Granularity Exploration

The pre-implemented flow aims to generate high-performance implementations by reusing high-quality and customized pre-built circuits in multiple contexts and chip locations. In this section, we present the performance of the generated circuits with two levels of granularity: single layer and block structure, as presented in Figure 5. The ResNet structure propagates inputs through the residual connections across layers. The Block-based is implemented so that the residual connection is contained within the block for area optimization.

Figure 6 Compare the performance of the different granularity when scaling the number of FPGAs. CNNs have large requirements for both computation and memory. As a result, the designs generated by the Block-based algorithms may not satisfy the constraints in both resource and timing. As observed in Figure 6, from 4 FPGAs, some blocks cannot fit anymore within an SLR, as the demand in terms of resources is the highest. Nevertheless, The block-based implementation achieves a slightly lower latency with 3 FPGAs but higher resource usage. We can conclude that fine-grained explicit dataflow is preferable to embedding the dataflow into a single block. The design generation time for the Layer-based implementation is the smallest. Since the design size is relatively small, and the proportion of replication of layers within the network is also higher, it drastically reduces the overall implementation time.

C. Performance

Although several singles and multi-FPGA DNN inference implementations exist, fair performance comparisons are challenging because of several factors such as: the DNN topology,

	Ma et al. [8]	Cloud-DNN [15]	Biookaghazadeh et al. [16]	Elastic-DF [10]	Zhang et al. [17]	CNN-on-AWS [18]	Our Approach
FPGA/Platform	Intel Arria 10	AWS	Intel Arria 10	2*U250	3* Virtex Ultrascale	5*AWS F1	3*xcku5p
Platfprm	Single FPGA			Multiple FPGAs			
Model	ResNet-50	ResNet-50	ResNet-50	ResNet-50	ResNet-152	ResNet-18	ResNet-50
FMax	200 MHz	125 MHz	212 MHz	217 MHz	150 MHx		246 MHz
Precision (fixed)	w16a16	w16a16	w8a8	w1a2	w16a16	w16a16	w1a4
DSP Blocks	1,518 (100%)	80.25%	33%	-	(19%, 18.8%, 9.3%, 30.3%)	-	((2.96%, 5.12% 1.05%, 3.82%)
LUTs	218.6K (51%)	64%	34%	-	(77.4%, 74.5%, 83.4%, 88.6%)	-	(3.57%, 6.36% 4.69%, 27.3%)
BRAM (M20K)	1,927 (71%)	83%	48%	-	(81.4%, 81.3%, 82.1%, 86%)	-	(41.2%, 50.6%, 23.42%, 60.3%)
Latency/Image (ms)	12.51	13.9	20.9	2.3	-	2.1	4.2
Throughput							

TABLE II: ResNet Performance Comparison with state-of-art approaches

	Layer-based ResNet			Block-based ResNet			Baseline ResNet		
	KFF	KLUTs	BRAM	KFF	KLUTs	BRAM	KFF	KLUTs	BRAM
Resources	741 (↓ 26%)	421 (↓ 24%)	821.5 (≅)	801 (↓ 16%)	479 (↓ 10%)	761.6 (↓ 7%)	935	526	822
Latency (ms)	4.8 (↓ 31%)			4.2 (↓ 33 %)			6.3		
Frequency (MHz)	276 (↑ 37%)			252 (↑ 25.3%)			201		
Avg. Power (W)	208			225			235		
Energy Efficiently	29.79			27.98			22.15		

TABLE III: Granularity Exploration of the ResNet implementation on 3 FPGAs

the model quantization, the hardware optimization objectives, RTL vs. HLS design, the evaluation methodology, and which performance metrics are reported. Table II compares our work with prior work of CNN inference on FPGA. Works are grouped into single and multi-FPGA implementations. The demand in terms of BRAM makes it impossible for ResNet, even with the smallest parameters, to fit on a single FPGA. Among the single-FPGA, Ma et al. [8] report the smallest latency of 12.51 ms by integrating optimized RTL components within an automated CNN compiler for various inference tasks. Elastic-DF is the closest work to ours regarding multi-FPGA implementations and achieves a latency of 2.1 ms. However, there is no report of resources for comparison. Concerning the throughput, Zhang et al. [17] has a 5% higher throughput. Nonetheless, we report a lower resource usage and a higher frequency.

D. Productivity

With the continuous growth of CNNs parameters and depth, improving productivity is an important factor in hardware design. This section shows how the proposed flow can leverage component reuse to reduce compile-time and implementation cycles. Table IV presents the time in hours to generate the design checkpoint with both rapidwright and vivado. ResNet topology reuse 69% of its layers. The proposed framework takes advantage of that properties and can achieve up to 3.3× productivity improvement. Nevertheless, with a higher number of blocks, the complexity of the problem also increases, resulting in a longer implementation time for the layer granularity.

	Layer Granularity		**ResNet**	
	Custom API	Inter-node Routing	Synthesis	P&R
Time (hours)	1.14	4.82	4.16	8.9
Ratio	∼ 17.4%	82.6%	35.6%	64.4%
Total (hours)	**5.96** (2.18× ↓)		**13.02**	

	Block Ganularity	
	Custom API	Inter-node Routing
Time (hours)	0.32	3.63
Ratio	8.10%	91.8%
Total (hours)	**3.95** (3.21× ↓)	

TABLE IV: Design Generation Time for implementation of ResNet with vivado and the proposed framework in hours.

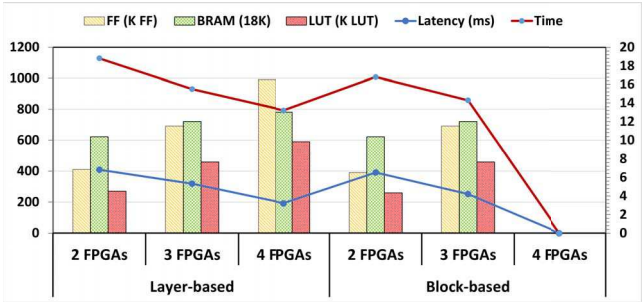


Fig. 6: Performance comparaison of different granularity when scaling the number of FPGAs.

V. Conclusion

This paper proposes a framework to accelerate model inference on a multi-FPGA Cloud Platform. The framework takes as input the computational graph of the CNN model inference. It performs an intensive search in the form of a quadratic optimization problem to determine each layer's highest degree of parallelism considering the platform constraints. The graph is then partitioned, and the resulting sub-graphs are allocated to the FPGAs' SLR such that the communication latency is minimized. Experiments and results show that our approach improves latency and maximum frequency, with little to no

impact on the number of resources used. Our workflow is designed in a modular fashion, allowing easy integration for new layer types. In future works, we intend to expand to a wider variety of neural networks and report power and energy consumption.

REFERENCES

[1] H. Kung, B. McDanel, S. Q. Zhang, X. Dong, and C. C. Chen, "Maestro: A memory-on-logic architecture for coordinated parallel use of many systolic arrays," in *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, vol. 2160. IEEE, 2019, pp. 42–50.

[2] Y. Chen, Y. Xie, L. Song, F. Chen, and T. Tang, "A survey of accelerator architectures for deep neural networks," *Engineering*, vol. 6, no. 3, pp. 264–274, 2020.

[3] B. McDanel, S. Q. Zhang, H. Kung, and X. Dong, "Full-stack optimization for accelerating cnns using powers-of-two weights with fpga validation," in *Proceedings of the ACM International Conference on Supercomputing*, 2020, pp. 449–460.

[4] S. Mittal, "A survey of fpga-based accelerators for convolutional neural networks," *Neural computing and applications*, pp. 1–31, 2020.

[5] S. I. Venieris and C. S. Bouganis, "fpgaConvNet: Mapping Regular and Irregular Convolutional Neural Networks on FPGAs," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–17, 2018.

[6] L. Petrica, T. Alonso, M. Kroes, N. Fraser, S. Cotofana, and M. Blott, "Memory-efficient dataflow inference for deep cnns on fpga," in *2020 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2020, pp. 48–55.

[7] 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, pp. 65–74.

[8] Y. Ma, Y. Cao, S. Vrudhula, and J.-s. Seo, "An automatic rtl compiler for high-throughput fpga implementation of diverse deep convolutional neural networks," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2017, pp. 1–8.

[9] C. Lavin and A. Kaviani, "Rapidwright: Enabling custom crafted implementations for fpgas," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2018, pp. 133–140.

[10] 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 Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 15, no. 2, pp. 1–34, 2021.

[11] A. Khodamoradi, K. Denolf, and R. Kastner, "S2n2: A fpga accelerator for streaming spiking neural networks," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2021, pp. 194–205.

[12] F. K. Mohammad Ghasemzadeh, Mohammad Samragh, "Rebnet: Residual binarized neural network," in *Proceedings of the 26th IEEE International Symposium on Field-Programmable Custom Computing Machines*, ser. FCCM '18, 2018.

[13] 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, pp. 1–23, 2018.

[14] xilinx, "Ultrascale architecture configurable logic block," https://www.xilinx.com/support/documentation/user_guides/ug574-ultrascale-clb.pdf, 20178.

[15] Y. Chen, J. He, X. Zhang, C. Hao, and D. Chen, "Cloud-dnn: An open framework for mapping dnn models to cloud fpgas," in *Proceedings of the 2019 ACM/SIGDA international symposium on field-programmable gate arrays*, 2019, pp. 73–82.

[16] S. Biookaghazadeh, P. K. Ravi, and M. Zhao, "Toward multi-fpga acceleration of the neural networks," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 17, no. 2, pp. 1–23, 2021.

[17] W. Zhang, J. Zhang, M. Shen, G. Luo, and N. Xiao, "An efficient mapping approach to large-scale dnns on multi-fpga architectures," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 1241–1244.

[18] J. Shan, M. T. Lazarescu, J. Cortadella, L. Lavagno, and M. R. Casu, "Cnn-on-aws: Efficient allocation of multikernel applications on multi-fpga platforms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 2, pp. 301–314, 2020.

[19] T. Benoist, B. Estellon, F. Gardi, R. Megel, and K. Nouioua, "Localsolver 1. x: a black-box local-search solver for 0-1 programming," *4or*, vol. 9, no. 3, p. 299, 2011.

[20] LocalSolver, "Benchmark quadratic assignment," retrieved November 26, 2020 from https://www.localsolver.com/benchmarkqap.html, 2020.