Enhancing the scalability of multi-FPGA stencil computations via highly optimized HDL components

ENRICO REGGIANI, Barcelona Supercomputing Center, Spain and Politecnico di Milano, Italy
EMANUELE DEL SOZZO, Politecnico di Milano, Italy
DAVIDE CONFICCONI, Politecnico di Milano, Italy
GIUSEPPE NATALE, Xilinx Research Labs, Ireland and Politecnico di Milano, Italy
CARLO MORONI, E-lysis, Italy
MARCO D. SANTAMBROGIO, Politecnico di Milano, Italy

Stencil-based algorithms are a relevant class of computational kernels in high-performance systems, as they appear in a plethora of fields, from image processing to seismic simulations, from numerical methods to physical modeling. Among the various incarnations of stencil-based computations, Iterative Stencil Loops (ISLs) and Convolutional Neural Networks (CNNs) represent two well-known examples of kernels belonging to the stencil class. Indeed, ISLs apply the same stencil several times until convergence, while CNN layers leverage stencils to extract features from an image. The computationally intensive essence of ISLs, CNNs, and in general stencil-based workloads requires solutions able to produce efficient implementations in terms of throughput and power efficiency. In this context, FPGAs are ideal candidates for such workloads, as they allow design architectures tailored to the stencil regular computational pattern. Moreover, the ever-growing need for performance enhancement leads FPGA-based architectures to scale to multiple devices to benefit from a distributed acceleration. For this reason, we propose a library of HDL components to effectively compute ISLs and CNNs inference on FPGA, along with a scalable multi-FPGA architecture, based on custom PCB interconnects. Our solution eases the design flow and guarantees both scalability and performance competitive with state-of-the-art works.

CCS Concepts: • **Hardware → Hardware accelerators**; *Buses and high-speed links*; • **Computer systems organization → Interconnection architectures**; *Reconfigurable computing*; • **Theory of computation →** Streaming models.

Additional Key Words and Phrases: multi-FPGA, stencil computation, HDL, CNN

ACM Reference Format:
Enrico Reggiani, Emanuele Del Sozzo, Davide Conficconi, Giuseppe Natale, Carlo Moroni, and Marco D. Santambrogio. . Enhancing the scalability of multi-FPGA stencil computations via highly optimized HDL components. , 1 ( ), 32 pages.

1 INTRODUCTION

Multimedia and image processing applications like computer and robot vision [27], Gaussian smoothing [63], and Sobel edge detection [3], numerical methods to solve linear eigenvalue problems [67] or model and simulate various natural phenomena and behaviors [21, 46], classical and quantum dynamics [11, 18, 50], cellular automata [81], pricing of financial options [32]. Although different at first glance, all these fields of application and many others share the same computational pattern and are examples of *stencil-based* algorithms. Given a system of points on a multi-dimensional

Authors' addresses: E. Reggiani, Barcelona Supercomputing Center, Barcelona, Spain; email: enrico.reggiani@bsc.es; E. Reggiani, E. Del Sozzo, D. Conficconi, G. Natale, and M. D. Santambrogio, Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Via Ponzio 34/5, 20133, Milano, Italy; emails: enrico2.reggiani@mail.polimi.it, {emanuele.delsozzo, davide.conficconi, giuseppe.natale, marco.santambrogio}@polimi.it; G. Natale, Xilinx Research Labs, Dublin, Ireland; email: gnatale@xilinx.com; C. Moroni, E-lysis, Via Roma 51, 20863, Concorezzo (MB), Italy; email: carlo.moroni@e-lysis.com.

. XXXX-XXXX//-ART
https://doi.org/

grid, the update of each point value depends on the weighted contributions from a fixed pattern of neighbors. This regular pattern is the stencil, and the update function applied to update each point of the grid is called *transition function*.

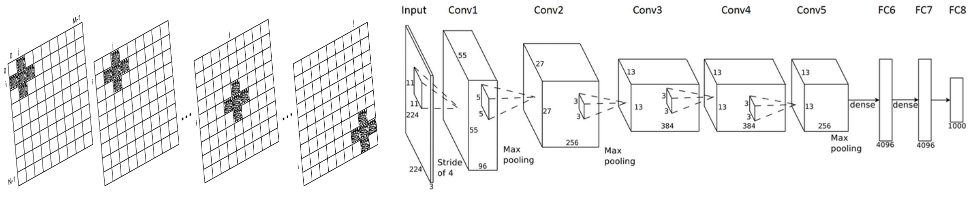
Among the multiple fields of application, two of the most prominent and relevant examples of stencil-based computations are Iterative Stencil Loops (ISLs), and Convolutional Neural Networks (CNNs) [30, 39, 82, 83]. Indeed, ISLs iteratively apply the same stencil on a multi-dimensional grid until the system of points reaches convergence. Similarly, the feature-extraction stage of CNNs employs several convolutional filters to extract specific features from the input image. In general, the computationally intense nature of stencil-based workloads makes them suitable for optimized implementations on high-performance systems [7]. The same concept holds for both ISLs and CNNs. In particular, both these kernel classes have proved to benefit principally from hardware accelerations [4, 12, 14, 17, 25, 41, 51, 61, 62, 75, 78, 85, 88, 90]; indeed, they require optimized computing to be used in real-life scenarios, as well as *tightly optimized architectures* in terms of both performance and power efficiency metrics. Besides, these kernel classes, especially CNNs, are continuously upgrading to improve their accuracy and require architectures whose *flexibility* and *design-productivity* are high enough to keep pace with the algorithm evolution. To this purpose, Field Programmable Gate Array (FPGA) [13, 29] are good candidates to handle these constraints against GPUs and ASICs, as they can provide good computational and power efficiency, while their reconfigurability properties satisfy the system flexibility needs. The principal gaps to cope with when using FPGAs as hardware accelerators are, however, their long design time and their steep learning curve. High-Level Synthesis (HLS) can ease the design process but often leads both to suboptimal implementation results and restricted architectural choices, also preventing the design to efficiently scale when distributed architectures are adopted [2, 22, 42–44, 61, 64, 79, 84, 90]. The architecture scalability is indeed a key aspect of accelerating heavily compute-intensive applications, as they usually need distributed systems to fulfill the performance requirements.

This work proposes a set of Hardware Description Language (HDL)-written Intellectual Property (IP) libraries that leverage the Streaming Stencil Timestep (SST) micro-architecture [51] to improve the scalability of stencil-based computations [57, 58]. Specifically, the proposed libraries are devised to design both ISLs and the feature extraction stage of CNNs. Besides, this work implements a multi-FPGA based system, composed of serially connected FPGAs. To improve the scalability, the proposed architecture leverages a custom Printed Circuit Board (PCB) interconnect, which provides a point-to-point connection with an overall bandwidth of 7.75 GB/s. The experimental evaluation shows that our multi-FPGA system achieves performance comparable to or better than the literature for both ISLs and CNNs. This work main contributions can be summarized as follows:

- highly-optimized set of HDL IPs based on an improved version of SST architecture to efficiently implement both ISLs and the feature extraction stage of CNNs on FPGA;
- design and implementation of a scalable multi-FPGA system;
- implementation and scalability validation of multiple ISL algorithms on the proposed system;
- implementation and estimation of two state-of-the-art CNNs, namely AlexNet [37] and VGG16 [66], on the proposed system.

2 BACKGROUND

This Section describes the technical background of this work. Section 2.1 and Section 2.2 explain the structure of ISLs and CNNs, respectively. Section 2.3 illustrates the fundamentals of FPGA Input/Output (I/O) this work builds upon for the multi-FPGA system.



(a) Single time-step of a cross-shaped ISL

(b) The AlexNet architecture

Fig. 1. ISL and CNN examples

2.1 Iterative Stencil Loops

ISLs represent a class of computational kernels used in a wide range of scientific and industrial applications, such as methods for partial differential equations [67], computational finance [32], and cellular automata [81]. These kernels usually constitute the computational bottleneck of a given application due to their iterative nature. Indeed, in an ISL algorithm elements of a multi-dimensional array are iteratively updated through a fixed pattern of weighted neighbors, called *stencil*. Depending on the ISL, the shape of the stencil changes, as well as its weights distribution, which can be composed either by a scalar or variable coefficients. A complete update of the multi-dimensional array, called *timestep*, is usually performed several times to output the final kernel result. Since stencils are composed of neighbor elements of the array, ISL algorithms have ghost zones on the array boundaries: depending on the nature of the algorithm, boundaries can thus be filled either with constant or periodic values. The filtering action performed by the stencil on a single *timesteps* is highlighted in Figure 1a, where the stencil is slid over the whole input array to iteratively isolate the reference element together with its north, south, west, and east neighbors. Since reading multiple data elements – *i.e.* the stencil – creates an output element, ISL computation is often bounded by memory transfers, which create stalls in the data processing path and worsen the performance. Regardless of the hardware architecture used, one of the major implementation challenges is to efficiently manage memory accesses and allow continuous data processing. Finally, ISL parallelization possibilities are limited by data dependencies between subsequent time update, as the next *timestep* can be computed only when the whole array is updated.

2.2 Convolutional Neural Networks

Firstly introduced in 1998 [39], CNNs take inspiration from the visual cortex biological process to perform recognition and classification of images and nowadays are the state-of-the-art class of deep learning algorithms for computer vision, and one of the hottest topics for both academics and industrial research [30, 82, 83]. CNNs are composed of two main stages of computation, named *Feature Extraction Stage* and *Feature Classifier Stage*, each of which can be divided into several layers. The *Feature Extraction Stage* is the first stage of a CNN, and it is composed of a sequence of Convolutional Layers, usually interleaved by activation and/or pooling layers, whose purpose is to evaluate and highlight the presence of a set of features of the input image. The *Feature Classifier Stage* is constituted by one or more Fully-Connected Layer (FCL). It elaborates the output of the last Feature Extraction layer, determining its affinity against the CNN classes. One of the most used CNNs in literature is AlexNet, reported in Figure 1b. Throughout this work, we do not focus on the CNN training, i.e., the phase in charge of equalizing the CNN coefficients used during inference, usually performed off-line only once, but on the efficiency of the inference part, used in real-time scenarios and having stringent throughput and energy requirements.

2.3 FPGA I/O

The Input/Output (I/O) capabilities of an architecture are crucial for its flexibility and performance. The exchange of data between different chips or boards divides into two main transmissions types: Parallel and Serial. In the former, data is sent simultaneously from a Transmitter (TX) to a Receiver (RX) on separate communication lines and synchronized with a common clock. In the latter, data are bit-wise serialized and transmitted sequentially on the same channel. The communication can be either synchronous when the transmission clock is provided by RX and sent to TX, or asynchronous, when TX transmits data asynchronously to RX (i.e., there is not a communication media containing a synchronization clock), which takes care of recovering data and clock.

Historically, parallel buses were a standard for chip-to-chip and board-to-board communications, as serial communication exhibited slower speed and overhead due to serialization and deserialization of data. To satisfy an ever-increasing bandwidth demand, the size of parallel data buses increased, and the communication clock rate. The increase in cost and dimension of parallel buses, together with issues related to clock and data skew and signal degradation over long distances, made room for serial transmissions, which reduced the overall pin and wires count, as well as Electromagnetic Interferences and costs. Nowadays, improvements like differential signaling, Double Data Rate transfers, or multi-lane serial transmissions have been adopted to improve both parallel and serial systems, and the performance gap between parallel and serial communications has been almost filled. Therefore, serial transmissions have become advantageous for off-chips, board communications, as they can provide higher performance while minimizing pin counts, simultaneous output switching noise, and overall system costs. Ideally, perfect data transmission between TX and RX interfaces would require an ideal clock synchronization in terms of both frequency and phase. To satisfy this constraint while reducing the overall pin counts, a Synchronous Serial Interface may seem the natural way to go, as the same clock is forwarded from RX to TX to handle data transmission; however, non-idealities introduced by the communication media add jitter, distortions, and attenuation to the transmitted clock, while increasing the communication media quality results in a higher overall cost of the system. Thanks to their low cost, a wide range of electronic systems (e.g., FPGAs) include dedicated hardware for serial links, known as Serializer/Deserializer (SerDes), which provides a high-speed Asynchronous Serial Interface.

In this context, Multi-Gigabit Transceivers (MGTs) are SerDes capable of reaching bit rates above 1 Gigabit/second. Thanks to their high bit rate capabilities, MGTs are increasingly used for high-speed data communications. The MGT main building block is a Parallel In Serial Out (PISO) / Serial In Parallel Out (SIPO) couple, where parallel data coming from the data bus are serialized by the PISO and transmitted through a communication media to the SIPO, which is in charge of recovering serial data in its original form. A method to correctly recover data at the receiver side of the transmission link is the Clock and Data Recovery (CDR) circuit. This circuit creates a recovered clock that is phase-aligned with the input transition position and used to sample the incoming data stream. The CDR is usually composed of a Phase-Locked Loop, a negative feedback circuit that, through a Voltage Controlled Oscillator (VCO), generates an output frequency locked to the incoming data streams in both frequency and phase. The Phase Detector (PD) behavior can be modeled as an XOR gate, as it outputs a signal that is proportional to the phase difference of its inputs. This signal is averaged by a low-pass filter and sent to the VCO, which generates a clock whose frequency is proportional to its input amplitude. The negative feedback loop reacts to generate a stable recovered clock, which is locked to the input stream in phase and frequency. It is important to guarantee enough transitions for the clock recovery circuit to operate: long sequences of ones or zeros at the input of the PD circuit would cause a Loss Of Lock that prevents the CDR to works correctly. Line encoding schemes, such as 8b/10b or 64b/66b, are thus used to ensure an adequate number of transitions on the serialized stream. It is worth noting that the CDR circuit must receive a data stream whose frequency is not affected by jitter to work correctly. Indeed, a jittered input signal would prevent the CDR from generating a stable recovered clock, causing a wrong sample of the stream and communication errors. Thanks to their high working frequency, modern MGTs allow extremely high-throughput communications and are widely used to interconnect FPGAs [13, 29, 45, 72], thus increasing the hardware accelerator performance.

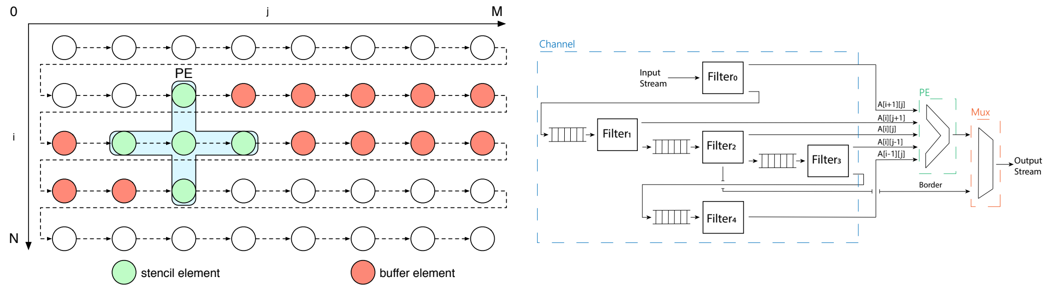
3 STATE OF THE ART

This Section describes the most relevant works about hardware acceleration, mainly on FPGA, of ISLs and CNNs. Specifically, Section 3.1 analyzes the literature about ISLs, Section 3.2 focuses on the State of the Art of CNNs, and Section 3.3 reviews relevant ISL and CNN multi-FPGA systems.

3.1 Iterative Stencil Loops

ISLs are widely used in scientific computations, but their low computational intensity limits the performance whenever not coupled with tailored memory optimizations. For this reason, efforts based on multi-core processors [6, 15, 71] and GPUs [28, 47] lead to sub-optimal results. The flexibility of reconfigurable devices such as FPGAs leaves large room for ISL performance improvements, since improving data reuse reduces off-chip memory accesses. Kobori and Maruyama start exploiting temporal parallelism of ISLs employing FPGAs for cellular automata [35]. In [59], a flexible RTL-based template for tri-dimensional stencil algorithms is proposed. To solve the memory-boundedness of ISLs, authors minimize off-chip data transfer by storing all data on local memories, though requiring large on-chip buffers as the problem size increases. Nacci et al. [49] present an automatic flow to generate the ISL hardware design. They merge multiple iterations in a tile, improving the data locality, which in turn is instantiated multiple times to work concurrently on different data, and subsequently processed by the following stages, creating a cone-shaped structure. Though improving on the off-chip boundedness, large stencil windows create on-chip memory port contention. Due to data dependencies, it must introduce redundant computation between neighboring tiles, thus limiting the overall performance and memory efficiency. Besides, [78] proposes a framework that leverages OpenCL pipes to share data between neighboring tiles, reducing the overall redundant computation. The framework traduces an OpenCL code in a cone-shaped architecture, where pipes bridge neighboring tiles to remove data redundancy among them. However, the on-chip port contention is still present, resulting in performance degradation when the number of merged iterations increases. SODA [14] is an automated framework devised to implement dataflow architectures for stencil computations on FPGA. Starting from a high-level input, SODA produces an efficient micro-architecture that minimizes external memory transfers and on-chip reuse buffer size. Moreover, the SODA framework includes resource and performance models to effectively and quickly explore the design space and identify performance-optimized configurations. The authors implemented different stencil computations on an AlphaData ADM-PCIE-KU3 board and obtained up to 3.28X speedup over a 24-thread optimized CPU implementation.

3.1.1 Streaming Stencil Timestep. The SST is an FPGA-based acceleration methodology that models ISLs as data-flow algorithms able to execute a single *timestep* update. The SST micro-architecture, firstly proposed in [12, 51], relies on the polyhedral model [20] to automate the design of a generic ISL algorithm from a C/C++ code. To provide a dataflow data-access, the SST micro-architecture uses a non-uniform memory partitioning, first proposed in [16], to perform on-chip buffering and provide concurrent accesses to all the elements of the stencil window at each clock cycle. In particular, the memory sub-system leverages a series of channels, one for every stencil element involved in the computation, composed of a series of chained filters, and interleaved by First In



(a) Stencil access pattern (b) Architecture of the SST

Fig. 2. SST access pattern and architecture (example on a $N \times M$ array indexed with variables i and j)

First Out (FIFO) buffers that realize the on-chip buffering relying on a blocking read and write access. These filters select from the incoming data stream the elements that belong to their data domain, *i.e.* , the elements accessed by the specific array reference, and send them to the Processing Element (PE), while also passing every element to the subsequent filter within the chain. Each filter has its own set of counters in charge of tracking the array spatial position and indicating whether to forward the data to the PE in any given clock cycle, according to the stencil shape. Figure 2a shows an example of a bi-dimensional cross-based stencil: the SST architecture handles input data, performing on-chip buffering on the elements depicted in range, and forwarding the stencil elements to the PE. The PE is a fully pipelined unit such that, at steady state, it produces a new output at each clock cycle. A Multiplexer (MUX) collects both the output from the PE and the array border, the elements that are not updated during the computation, and reconstructs the output data stream with every element in the same spatial position as the original input stream.

The corresponding SST micro-architecture, as in Figure 2b, is composed of one channel having five filters. Each filter forwards every element to the subsequent filter, and the elements belonging to its data domain to the PE. To preserve coherency among data, FIFOs between filters store every element whose spatial location is between two subsequent filters data domain. Since the MUX reshapes the stream to be spatially coherent with the input array, it is possible to chain several SSTs together to accelerate multiple *timesteps* within a single sweep. The queue of SSTs acts therefore as a high-level pipeline, where, at steady state, each SST is concurrently active and computing on a portion of the data stream. This technique requires constant bandwidth, which is independent of the number of iterations queued. Indeed, data transfers between off-chip and on-chip memories happen only at the beginning and at the end of the SST chain, and they do not vary with the length of the queue. Thus, it is possible to scale the size of the accelerator and boost performance [12, 51], with the only limits being the total number of timesteps and the available target FPGA resources.

3.2 Convolutional Neural Networks

The CNN computational intensity makes such networks usually suitable for large High Performance Computing (HPC) systems, data centers and highly performing hardware accelerators like GPUs [69], FPGAs [30] or even ASICs [34]. While GPUs are the current *de-facto* standard for CNN training, they are less attractive for the inference task due to the high power they drain. Indeed, considering other accelerators, FPGAs offer great advantages in terms of both power efficiency and flexibility, and newer FPGA generations exploit several Digital Signal Processing (DSP) blocks and MegaBytes of on-chip memory for CNN requirements. For these reasons, FPGA-based CNN acceleration is becoming one of the most attractive alternatives in both the embedded [19, 55, 60], and the HPC domains [56, 85, 88], as described by state-of-the-art surveys [25, 75]. Indeed, in [80], the authors use a bi-dimensional systolic array pattern to implement an end-to-end automation flow for CNNs targeting FPGAs. The architecture is composed of a grid of pipelined PEs, exploiting meshed local connections to exploit FPGA routing resources better, achieving higher working frequencies. This work uses a two-phase Design Space Exploration (DSE), where the first phase relies on performance and resource models to evaluate the best over several pre-designed templates, while the second uses the hardware generation flow of the target device to reach the best performance. The work is evaluated on an Arria 10 GT 1150 board against AlexNet, with a throughput of 360.4 GFLOPS.

Relying on Intel FPGA SDK for OpenCL, the authors of [4] introduce a CNN architecture targeting an Intel Arria 10 FPGA able to cache on-chip both CNN features and weights. Moreover, they use a 1D Winograd transform to reduce the convolution arithmetic complexity. They introduce a shared exponent technique to convert half-precision floating-point data to 18-bit fixed-point. This enables a fully fit of the Altera DSP blocks, which can compute two 18x18 fixed-point multiplications in a single unit, saving memory resources while increasing the computational efficiency, without accuracy losses. They test AlexNet network by implementing every single stage of the network on the whole FPGA, reaching a maximum of 1382 GFLOPS at 303 MHz, without taking into account the FPGA reconfiguration time. In [41] the authors implement the 2D Winograd algorithm [38] to improve their FPGA-based CNN design. This algorithm offers arithmetic complexity reduction, and it is one of the fastest known matrix multiplication algorithms, enabling resource savings. The authors design line-buffers to cache feature maps before and after the Winograd transformation. Moreover, they propose a DSE, from which they derive an automatic tool flow, where their AlexNet implementation on a Xilinx ZC706 platform reaches 1006.4 GOPS running at 200 MHz.

Other works exploit the Winograd transformation [1, 2]. In particular, Ahmad et al. [2] rely on this technique to implement an optimized convolution engine obtaining significant throughput focusing on single layers of AlexNet (3331.6 GOPS) and groups of layers of VGG16 (3623.9 GOPS). Zhang et al. [86] propose a DSE technique to identify the optimal architectural solution, in terms of both performance and resource usage, over a large design space and leveraging on both roofline model and the Computation to Communication Ratio. The design space is created by applying to the described network a set of optimizations such as on-chip reuse buffering and loop transformations, e.g., tiling, pipelining, and unrolling. This methodology is applied only to the CNNs convolutional layers rather than the full network. Implementing AlexNet on a Xilinx Virtex-485T with an operating frequency of 100MHz, this work achieves 1.33GFLOPS. In [70], the authors describe a design space methodology to accelerate CNNs on OpenCL-based FPGA. Such a methodology aims to maximize network throughput by considering the FPGA resources and memory bandwidth. The authors evaluated their approach on both AlexNet and VGG on two Altera boards, namely DE5-Net and P395-D8. On the other hand, Zhang et al. [87] leverage on Fast Fourier Transform and Overlap-and-Add to optimize the convolutional layers of CNNs and reduce their computational requirements. In this way, the authors significantly reduced the number of required floating-point computations. The experimental evaluation, performed on an Intel Quick-Assist QPI FPGA Platform, reached 123.48, 83.00, and 96.60 GFLOPS on VGG16, AlexNet, and GoogLeNet, respectively. Li et al. [40] propose an end-to-end CNN accelerator where all the network layers are simultaneously mapped on the same FPGA. This work takes advantage of both a batch-based approach and a design space methodology to optimize external memory utilization and identify the proper parallelism level. The authors implemented AlexNet on a Xilinx VC709 and reached 565.94 GOPS as peak performance.

State of the Art also contains various works [22, 42, 44, 64, 74, 79, 84, 85, 90] describing automatic toolchains to map CNNs on FPGAs. Usually, these tools start from a high-level representation of the network described using frameworks like Caffe [33], PyTorch [53], TensorFlow, etc., and then apply a sequence of transformations and optimizations to produce efficient FPGA designs. For instance, Caffeine [85] is a framework integrated with Caffe that exploits a systolic-like computational architecture to evaluate CNNs on FPGAs using a matrix-multiplication pattern. This framework allows users to configure different CNNs directly in software without changing the FPGA bitstream. They use a weight-major mapping technique to exploit data locality for both convolution and FCL execution. As a case study, they implement both VGG16 and AlexNet on various FPGAs peaking 636 and 338 GOPS, respectively. DNNBuilder [90] is another relevant example of automatic toolchains. DNNBuilder takes as input Deep Neural Network (DNN) description from Caffe and TensorFlow frameworks and exploits an automatic design space exploration to produce an optimized FPGA design. In particular, the DSE tool takes into account the target board features (e.g., available FPGA resources and external memory bandwidth), as well as the DNN complexity. The authors evaluated DNNBuilder on four different DNNs, namely AlexNet, ZF, VGG16, and YOLO, on both edge and cloud FPGAs, outperforming state-of-the-art FPGA accelerators.

Finally, in [5] the authors propose a data-flow accelerator template that relies on the SST architecture, achieving 5.2 and 28.4 GFLOPS on two CNNs based on USPS and CIFAR-10 datasets. On top of [5], the authors present Condor, an end-to-end framework [56] integrated with Caffe [33]. Albeit already proposed in [85], [56] enables the accelerator cloud integration through the Amazon AWS F1 instances, reporting 8.36 GFLOPS for USPS and 53.51 for the LeNet feature extraction stage.

3.3 ISLs and CNNs Multi-FPGAs

Both ISLs and CNNs are large-scale computational kernels that could benefit from multi-FPGA platforms on the execution times and power efficiency sides. Here, we review multi-FPGAs implementing ISLs and CNNs of interest to this work.

Sano et al. [61] present a multi-FPGA domain-specific programmable architecture for ISLs that guarantees a flexible accelerator by leveraging run-time configurable processing elements. This architecture exploits both time and spatial parallelism and applies a streaming computational pattern, allowing linear performance scalability. This work has been prototyped on a cluster of nine Altera Stratix III, achieving 260 GFLOPS and 236 GFLOPS for 2D and 3D Jacobi kernels, while they model the performance of 100 chained Altera Stratix V with peak estimation of 17 TFLOPS and 17.7 TFLOPS. Waidyasooriya et al. [77] proposed an FPGA-platform based on OpenCL to accelerate stencil computations. Such a platform consists of a pipeline of computation modules, each performing one iteration of the stencil computation. The authors also developed an optimization methodology that identifies the optimal architecture to implement based on the devised platform, starting from a given application. The experimental evaluation, performed on two FPGA boards, namely a DE5 and 395-D8, covers both 2D and 3D single-precision stencil computations and shows performance ranging from 119 to 237 GFLOPS. Starting from this work, Waidyasooriya et al. [76] designed a multi-FPGA accelerator to scale stencil computations in time and space dimensions. The authors evaluated their approach on a two-FPGA system, where they connected two Nallatech 385A accelerator boards to the host via PCIe and to each other via QSFP interconnect. Finally, the author executed 2D and 3D stencil benchmarks using *skewed grids* and *non-skewed grids*, reaching up to 950 GFLOPS (single FPGA) and 1861 GFLOPS (two FPGAs). Considering the SST micro-architecture, Cattaneo et al. evaluate their framework [12] on a multi-FPGA system [51] composed of two Xilinx Virtex-485T, reaching 30 GFLOPS and 6 GFLOPS for a 2D and 3D Jacobi solver, respectively, and up to 10x power-efficiency improvement when compared to [6].

Considering the CNNs, Zhang et al. [88] propose a pipelined FPGA cluster, where seven FPGAs are connected through a ring topology, reaching single-link bandwidth of 750 MB/s to improve the energy-efficiency of FPGA-based CNN accelerators. The authors then propose an analytical model to easily map CNNs on a multi-FPGA architecture by choosing the selected objective, *i.e.* , energy, latency, or throughput . The result of their AlexNet implementation shows a maximum throughput of 825.6 GOPS relying on a cluster of four Xilinx Virtex VX690t. Thereby, a multi-FPGA system for ISLs, or CNNs, is not a novelty per se. Still, this work proposes a system with one of the best bandwidths for chip-to-chip communication against related works, as Section 6.1 will show. The high bandwidth feature is the enabling improvement for achieving performance and scaling CNNs and ISLs in a multi-FPGA system, as shown in Section 7.

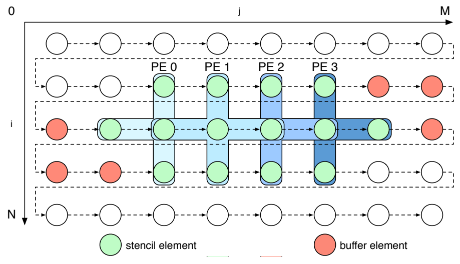
Summary. This Section focused on relevant implementations available in the literature about both ISLs and CNNs. Among them, the SST approach [12, 51] applied to ISL computational pattern has shown exciting results, as the authors can to both efficiently partition the on-chip memory and perform data-flow computations on the stencil. On the other hand, [5, 56] showed that the same micro-architecture applies to CNNs as well. Indeed, ISLs and CNN layers share similar characteristics in data locality patterns. However, all these works show significant limitations on throughput, scalability, and resource utilization efficiency, which prevents them from reaching performance comparable with the literature. Besides, they leverage HLS tools and off-the-shelf IP components to generate the hardware design, resulting in a sub-optimal architecture in terms of both timing and resource consumption. For these reasons, the proposed work aims to improve the above-mentioned design methodology by exploiting scalable and parameterizable HDL modules to describe the most common ISL and CNN computational blocks, providing tailored components that alleviate the user effort to describe specific networks in HDL. Moreover, since both computational domains are moving toward the usage of multi-FPGA systems [12, 30, 61, 88], this work also proposes a custom multi-FPGA architecture that leverages modular ad-hoc PCBs optimized for data-flow applications.

4 HARDWARE ACCELERATION METHODOLOGIES

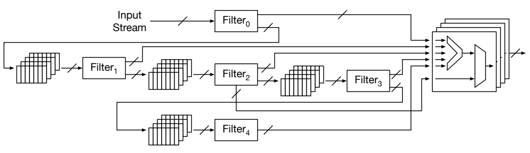
This Section presents the fine-grained structure of our HDL IPs for stencil-based computations. As stated before, various application fields rely on stencil-based kernels. In this work, we focus on two particular instances of algorithms that build upon the stencil model: ISLs and CNNs. Although they may look different, these algorithms share a similar data locality pattern, based on applying one or more stencils to the input data. Our accelerators for these workload classes rely on the SST micro-architecture [16, 51], as in Section 3.1.1, to perform on-chip memory buffering while providing a data-flow like computation. Our enhanced and reusable SST-based architecture exploits both inter-iteration parallelism and a new degree of parallelism, namely intra-iteration parallelism, performing better than [57] for ISLs. Concerning CNNs, we combine inter-layer, intra-layer, intra-FM parallelism, DSP time-sharing, and clock gating. To the best of our knowledge, we are the first to apply the combination of DSP time-sharing and clock gating to CNN computations. These techniques considerably increase parallelism, throughput, and weight reuse. Finally, given the benefits of a distributed system, Section 6 proposes a scalable multi-FPGA architecture, where an arbitrary number of serially connected FPGAs provides a pipeline of computational resources.

4.1 ISL Hardware Architecture

The work proposed in [51] keeps large rooms for improvements. Indeed, the authors do not apply intra-iteration parallelization to fully exploit the system throughput capabilities and leverage HLS tools to create the architecture, with a consequent sub-optimal resources handling. This work builds on top of [51] to overcome these limitations. Specifically, it provides a set of architectural templates to minimize application design time and exploit specific optimizations to improve the overall performance. Indeed, the proposed solution implements both *inter-* and *intra-iteration* parallelization, while keeping the on-chip memory requirements to a (analytically optimal) minimum [17]. The proposed hardware accelerator components implement a data-flow model of computation, which relies on three fundamental elements: *filters*, *FIFOs*, and *PE*. The templates can be examined



(a) Improved stencil access pattern



(b) Improved micro-architecture block diagram

Fig. 3. Enhanced SST design

at two levels of hierarchies: at the level of the single timestep and the entire accelerator level, able to accelerate multiple timesteps in a single sweep.

4.1.1 Micro-architecture for Single Timestep. The non-uniform memory partitioning, proposed in [16], performs on-chip data buffering, providing concurrent accesses to all the stencil window elements at each clock cycle. As described in Section 3.1.1, the SST memory sub-system is made by a series of channels, composed of a series of filters connected in a chain, and interleaved by FIFO buffers. These filters select from the incoming data stream the elements that belong to their data domain and send them to the PE, while also passing every element to the next filter within the chain. Addressed in the same Section, the micro-architecture exploits counters to perform data filtering, adapted to support intra-iteration parallelism.

While the fully pipelined PE converges to the new value production at each clock cycle, a *multiplexer engine* collects all the elements to reconstruct the output stream positions as the input stream. The proposed solution exploits intra-iteration parallelization to enhance computational intensity and bandwidth of the stencil kernels while minimizing on-chip memory usage. It processes consecutive element updates using multiple parallel PEs, maximizing thus data reuse[1] while having a minimal impact on on-chip memory usage. Figure 3a illustrates the implemented computational pattern, easily comparable with Figure 2a. The highlighted stencil windows and all the elements included between the stencil boundaries – north and south elements if columns slide the matrix –are the ones stored on-chip at the instant depicted in figure. In general, every parallel PE added to improve computational intensity and bandwidth of the stencil kernels requires an additional number of elements to be stored on-chip equal to the stride of the stencil kernel, which is commonly 1. Comparing Figure 3a and Figure 2a, only 3 additional elements need to be indeed stored on-chip to enable 4 concurrent updates. This means that the width of the on-chip memories has to be increased to store the required elements. Consequently, we increased the width of the on-chip memories to store the required elements concurrently, thus guaranteeing contention-free data accesses. Increasing the intra-iteration parallelization improves the accelerator bandwidth but complicates the SST synchronization mechanism and increases the resources required to perform the computation. Hence, our implementation exploits an intra-iteration parallelization of four. We chose this solution to balance the parallelization benefits in throughput and resource utilization.

For what concerns the single timestep, this solution leads to a performance increase of a factor 4 against the solution proposed in [12, 51]. Moreover, intra-iteration parallelization enhances the overall bandwidth, since to sustain the datapath rate, multiple elements per clock cycle need to be fetched from off-chip memory in a number equal to the number of parallelized PEs. Thus, this

[1]the stencil windows of consecutive elements are overlapped and therefore share common data that can be reused

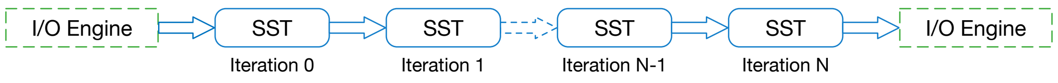
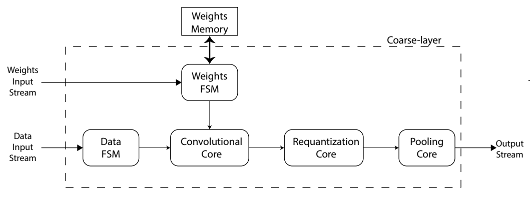


Fig. 4. Scheme showing the structure of the proposed ISL hardware accelerator.

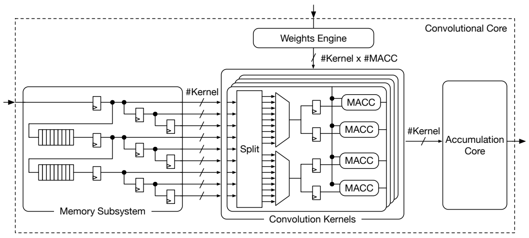
parallelism allows to match the off-chip bandwidth with the computational ones and respect the data-flow computational pattern, where consumed and produced data must be tuned to avoid stalls. Specifically, the input data stream is firstly processed by different filtering stages, each having as many filtering conditions as the number of parallel PEs, as depicted in Figure 3b. This set of filters' main purpose is to orchestrate data to guarantee a continuous flow towards the PEs without other control structures that may interfere with the stream of data. Each filter outputs data to both the PEs and the subsequent filter within the channel; to guarantee coherency among data, consecutive filters are interleaved by read- and write-blocking FIFOs, used as on-chip memory buffers to store the current input elements. Since multiple elements are updated concurrently, the structure of a channel has been updated by inserting multiple FIFOs between filters, equal to the number of elements updated in parallel, allowing the transfer of multiple separate elements concurrently. More specifically, if N is the width of the input matrix, while [51] requires one FIFO to store $N-1$ elements - which represent the maximum distance between two points belonging to the SST window - the proposed solution exploits Y FIFOs to store the same data, each containing $(N-1)/Y$ elements. This change has no theoretical impact on memory resource usage since the difference lies in how these FIFOs are written/read. Nonetheless, there may be a minimal difference in the resource usage that mainly depends on the synthesizer. On the other hand, it may be easier for a synthesis tool to place Y smaller FIFOs on the FPGA instead of a larger one. Valid data at the output of the last filter notify a fullness of the memory pipeline. From that moment forward, all the PEs start consuming data concurrently and computing.

The proposed micro-architecture is implemented in HDL. This design choice allows performing tailored optimizations to improve both resource utilization and the working frequency, dramatically improving the system performance. Therefore, each PE exploits a mix of LUTs and DSPs to balance resource utilization, while input and output operator registers guarantee to meet timing constraints even with a high working frequency. PE results and the input border, are then ordered by a *multiplexer engine* to spatially reconstruct the array and provide meaningful data to the next stencil iteration. A PE is implemented by replicating the algorithm behavior, either using fixed or floating-point notations, exploiting elementary arithmetic operations, such as additions and multiplications. The created PE is automatically replicated and connected to both the SST and the multiplexer engine to match the imposed intra-iteration parallelism. The proposed library implements the most common stencil windows for several data dimensions, *i.e.* , 1D, 2D, 3D. As a result, the user just has to select the desired stencil window and specify the operation that the PEs have to implement.

4.1.2 ISL Hardware Accelerator. The described micro-architecture contains all the functional characteristics to evaluate the target iterative stencil over a given number of timesteps. However, implementing just a single iteration engine core would lead to poor performance as data would be transferred from off-chip memory T times, where T is the number of timesteps. Since, as in [51], data exiting from a single iteration engine are ordered to be spatially coherent with the input array, multiple engines can be serially connected to accelerate multiple timesteps in a single Input/Output (I/O) transfer, as shown in Figure 4. The streaming data pattern and the data-flow computational model of the proposed accelerator allow achieving a pipelined execution within the chain, where each iteration can be seen as a "pipeline stage". The overall accelerator comprises N



(a) Coarse-layer architecture



(b) Convolutional Core architecture

Fig. 5. CNN accelerator architecture

serially connected engines, reducing the completion time while increasing resource utilization by a factor of N against the single iteration design. The ISL computational pattern, along with the presented SST architecture, allows a throughput increase that scales with the number of chained timesteps. Thus, the performance of the proposed multi-timestep accelerator is just limited by the available FPGA fabric resources and the bandwidth available from the FPGA I/O. An approach to further improve the system scalability consists of fully exploiting the SST data-flow paradigm through a point-to-point multi-FPGA system, where multiple FPGAs are chained to compose a deep pipeline of acceleration engines. Indeed, this solution allows extending the data-flow approach that characterizes the SSTs to an arbitrary number of accelerators, where every SST performs only one timestep on the same input data. The hardware architecture discussed in Section 6 aims to cope with these considerations, thus extending the stencil acceleration to a distributed system scale.

Finally, it is worth noting that there is no control step between two SSTs. Although this control could be useful to identify an early convergence of a stencil computation, it would include additional and unnecessary logic that could limit the scalability of the multi-timestep approach. Moreover, this would add complexity and contention when processing a batch of data, as these data could potentially converge at different stages of the multi-timestep pipeline. Therefore, two control steps could simultaneously require access to the I/O interface.

4.2 CNN Hardware Architecture

Moving to CNN, we propose a flexible and scalable methodology to accelerate any CNN inference on FPGAs, by combining HDL and HLS design flows and exploiting data quantization and time-sharing. In the context of CNN accelerations, Convolutional Layers (CNVLs) represent the most critical computational kernel, as they contain most of the network operations. Thus they consume most of the accelerator resources. Indeed, an inaccurate design of the CNVL quickly leads to high latency and system performance degradation. For this reason, it is crucial to fully exploit the FPGA parallel architecture. In general, after CNVLs follow an activation layer, a requantization layer – if working with quantized data –, and, sometimes, a pooling layer. In this work, the aforementioned quartet – or trio, if the pooling layer is absent – is named *coarse-layer*, and the features extraction part of a CNN can be viewed as a chain of coarse-layers.

The proposed IP library can leverage the same hardware resources to implement several subsequent *coarse-layers*, namely *stages*. A coarse-layer consists in the components shown in Figure 5a: the *Convolutional Core (CNVC)*, the *Pooling Core*, the *Data Movements Finite State Machine (FSM)*, and the *Requantization Core*. It is worth to note that the proposed HDL IP library only implements the Convolutional Core and the Pooling Core. On the other hand, we designed both the Requantization Core and the FSMs using HLS tools – *i.e.* Vivado HLS –. The FSMs are lightweight components – *i.e.* every FSM occupies just some hundreds of Look-Up Tables (LUTs) and Flip Flops (FFs) – and would not benefit from an HDL-based implementation. Therefore, relying on HLS to implement these cores does not affect accelerations performance but improves design productivity. On the other way, the Requantization Core needs a moderate number of DSPs to perform bias addition and data quantization. Nevertheless, the integration of this core in the HDL IP library is future work.

4.2.1 Convolutional Core. The CNVC contains most of the CNN operations, and results to be compute-intensive even when high parallelism imposes a substantial data movement from the off-chip memory. The outcome of this core is a volume composed of a set of bi-dimensional matrices, named output feature-maps. Specifically, the core input feature-maps are convolved with a set of filters. The resultant volume is summed element-wise to obtain a single bi-dimensional output feature-map. The described computational pattern keeps room for two principal parallelism opportunities, named *intra-FM* and *intra-layer* parallelisms. Intra-FM parallelism relies on the data dependencies absence to compute the output feature-map; thus, a set of convolutions are computed concurrently. A reduction tree is used to create the output feature-map element. Intra-layer parallelism exploits data reuse to convolve the same layer input feature-maps with different weights-set to compute several output feature-maps in parallel.

The above-mentioned parallelism paradigms can easily saturate the FPGA computational resources. Even when low-precision fixed-point arithmetic is adopted, DSPs are not able to fulfill the parallelism possibilities offered by this core. Consequently, a fine-grained tuning of the CNVC parallelism dictates both the overall performance and the parallelism to infer on the other components. To exploit both intra-FM and intra-layer parallelisms while saving DSP resources, this library implements a *time-sharing* approach to compute each convolution with one DSP primitive. Each DSP is used as Multiply and Accumulate (MACC) unit to perform an element-wise multiplication between the input feature-map and the correspondent filter while accumulating these partial results. To allow a single DSP to perform this computation, the input datapath flows at a sub-multiple of the clock frequency, whose value changes according to the filter dimension. In particular, we are clock-gating the datapath as its clock is enabled once every $N \times N$ clock cycles, where N is the filter dimension. This solution offers a substantial DSP count reduction and can exploit larger intra-layer and intra-FM parallelism degree, resulting in less off-chip data movement to process the network. Moreover, the design relies on parameterized HDL-based cores easily retailored for the target network and device through the following parameters: filter and input feature-map dimensions, bit-width of data and weight buses, padding, and stride factors, intra-FM and intra-layer parallelisms. Figure 5b represents the CNVC. The Memory Subsystem feeds a set of *Convolution Kernels*, guaranteeing a data-flow communication pattern, while a *Weights Engine* loads and distributes the weight windows to each *Convolution Kernel*, whose output is hierarchically accumulated by the *Accumulation Core* to create the corresponding output feature-maps.

Memory Subsystem. The CNVLs require concurrent access to several neighbor elements of the input feature-maps: to perform the convolution, these elements need to be slid and processed over a filter. The presented methodology relies on bi-dimensional line buffers to concurrently access all the filter elements while performing a data-flow computation. Specifically, an input channel streams sequentially the input feature-maps elements, while a chain of FIFOs is used to cache a suitable number of rows and output concurrent elements, preserving data locality. The concept of on-chip buffering implemented in the Memory Subsystem leverages the SST architecture [51]: indeed, as highlighted in [5], the data locality needed to perform a convolution requires the extraction of a bi-dimensional square stencil from the input feature-maps, and the SST architecture can be applied to allow on-chip memory buffering on the Convolution Kernel input data. To enhance flexibility, this work automatically infers a suitable number of FIFOs to implement the chain and the optimal FIFO depth, which has to store an entire input feature-map row.

The Memory Subsystem exploits intra-FM parallelism, as the bit-width of its data-bus can be set to buffer several input feature-maps concurrently. Intra-layer parallelism is extracted from the Memory Subsystem outputs, as the same data are used to compute different output feature-maps. All datapaths of these blocks flow at a sub-multiple of the clock frequency, whose value depends on the filter dimension, to allow all the Convolution Kernels to compute data through the *time-sharing technique*. When the stride factor is greater than one, the updating frequency dynamically changes depending on the spatial location of the elements within the FIFO chain. This optimization allows freezing the FIFOs chain only when the Convolution Kernels must process its output to produce valid elements while running at a full frequency to skip dummy window elements. Finally, to orchestrate the data stream, a controller is in charge of handling FIFO read and write enables.

Convolution Kernels. The Memory Subsystem outputs are consumed by a set of Convolution Kernels that perform the element-wise multiplications between the input feature-map elements and weights, and accumulate the results. MACC units and MUXs perform this computation: both these functional blocks work at a full frequency and leverage the datapath slowness to consume data while sharing computational resources. In particular, each clock cycle of the time-sharing period is used to multiply the *i-th* element of the input window. The MUX output is fed to the MACC unit, which performs the multiplication with the corresponding weight and accumulates the partial result. After $N \times N$ clock cycles, where $N \times N$ represents the dimension of the stencil – *i.e.* , the dimension of the filter –, the accumulator outputs the final result and resets its content. To enhance data locality, Convolution Kernels can be replicated to compute output feature-maps in parallel, exploiting intra-layer parallelism. In this case, the same data are streamed to different MACCs and multiplied with different weights.

Weights Engine. In CNVLs, the same weight window is slid over the whole input feature map. This pattern improves data locality by saving the weight window in small memories until the input feature map is completely processed. This solution requires a high peak bandwidth: when a new set of feature-maps has to be computed, slowness in weight loading from off-chip memory implies computation stalls. Moreover, depending on intra-FM and intra-layer parallelisms, the number of weights loaded in each iteration widely varies. Weights are loaded and saved on-chip before the Memory Subsystem starts to forwards valid data to the Convolution Kernels to avoid stalls. To properly orchestrate the weight loading, a tailored *gearbox engine* has been implemented. The gearbox automatically distributes data from off-chip to on-chip memories, providing weights to each MACC unit until the input feature-maps are entirely processed. All on-chip memories share the same controlling structure: memories must be written to store the gearbox data at the beginning of each Convolution Kernel run and must provide data to the MACC units during the computation. Suppose the off-chip memory bandwidth is not enough to avoid computational stalls. In that case, the Memory Subsystem freezes and stops sending data to the Convolution Kernels until the Weights Engine completes weight loading. From that moment, the Weights Engine starts writing data to the MACC units, guaranteeing synchronism and coherency with the input datapath.

Accumulation Core. The results of several Convolution Kernels are summed element-wise to create the output feature-map elements based on intra-FM and intra-layer parallelisms. To this purpose, the proposed methodology exploits a hierarchical accumulator strategy, where each hierarchy level performs a partial data reduction until the output feature-map element is generated.

4.2.2 Pooling, Data Movement, and Requantization Cores. The proposed *Pooling Core* architecture shares the same memory access pattern of the CNVC. A FIFO chain collects the output feature-maps produced by one of the Requantization Core, while tailored Pooling Kernels consume its output. The

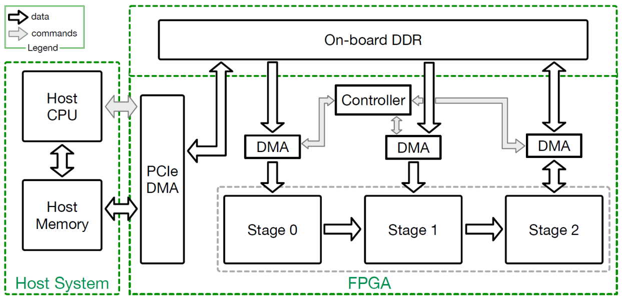


Fig. 6. High-level overview of the proposed CNN hardware accelerator

Pooling Kernel does not require specific computational resources, as it just needs one comparator working at a full frequency to find the maximum value of the window.

The *Data Movement* cores are HLS-based FSMs, given their simplicity, and handle data movement inside each stage and counting number of elements and number of iterations missing to end of the coarse-layer computation. In particular, there are two main cores: *Data FSM* and *Weights FSM*. The *Data FSM* is interposed between two subsequent CNVLs to perform data collection and reordering and pre-process data for the subsequent Convolution Kernel, as data between adjacent layers need to be normalized. Specifically, data coming from the previous coarse-layer are saved on-chip and distributed to the Convolution Kernel. Data reordering is necessary when the CNVC computes a fraction of output feature-maps simultaneously. Therefore, data in input to the Data FSM are composed of fragmented output feature-maps and must be appropriately reordered to feed the following coarse-layer correctly. The *Weights FSM* takes weights from the off-chip memory and simply perform an element-wise normalization of the CNVC weights.

As discussed in the literature [24–26, 90], CNN inference with lower-precision quantized networks can achieve results comparable to floating-point arithmetic, with an accuracy loss of only a few percentage points. Quantized networks are beneficial for FPGA devices, as fixed-point arithmetic exploits reduced bit-width data types than the floating-point one, reducing latency, resource consumption, and memory bandwidth. As a direct consequence of these considerations, this work leverages 8-bit integer arithmetics to perform the CNN computations. Indeed, 8-bit quantization has proven to be the right trade-off between lowering precision and maintaining a level of accuracy comparable with the uncompressed model and is the quantization choice of several industry leaders [36, 52] and works in literature [24, 26, 42, 43, 70, 73, 90]. Specifically, the 8-bit quantized data and weights feed the CNVC, which performs the convolution and outputs data whose bit-width is larger than 8 bits – *i.e.* , large enough to avoid overflows during the computation –. As a result, a *Requantization Core* is necessary to correctly requantize the output feature-maps to 8 bits.

The *Requantization Core* is placed after the accumulation hierarchy and performs data requantization, Rectified Linear Units (ReLU) activation, and biases addition. Indeed, from the Convolution Kernel input and the Accumulator outputs, data bit-widths have to be increased to avoid overflow during the computation. Hence, once feature-maps are available, the Requantization Core rescales these elements at their original bit-widths. Lastly, this core might consume a (negligible) part of the available DSPs, which depends on the HLS process and the provided directives.

4.2.3 CNN Hardware Accelerator. The proposed architecture follows a classical host-accelerator scheme where the *host* – *i.e.* , the CPU – is responsible for the initial data pre-processing – if needed – and the final results gathering after hardware acceleration takes place. Data is first sent

Table 1. List of parameters for ISL and CNN HDL IPs

ISL Parameters	CNN Parameters Convolution Kernel	Accumulation Core	Pooling Core	Description
D_WIDTH	D_WIDTH	D_WIDTH_IN	D_WIDTH	Input data bitwidth
INPUT_DIM	-	-	-	Input dimension
PE_LATENCY	-	-	-	Number of clock cycles used by the PE to compute the output result
FILTER_SHAPE	-	-	-	Shape of the filter (square, cross, ...)
CHAIN_LENGTH	-	-	-	Length of SST chain
-	W_WIDTH	-	-	Weight bitwidth
-	-	ACC_WIDTH	-	Accumulation bitwidth
-	MACC_WIDTH	D_WIDTH_OUT	-	Output bitwidth (if < ACC_WIDTH, data are truncated)
-	FM_PARAL	FM_PARAL	FM_PARAL	Intra FM parallelism
-	LAYER_PARAL	LAYER_PARAL	LAYER_PARAL	Intra-layer parallelism
-	KERNEL_DIM	KERNEL_DIM	KERNEL_DIM	Layer stencil dimension
-	DMA_WIDTH	-	-	DMA bitwidth
-	NUM_LAYERS	NUM_LAYERS	NUM_LAYERS	Number of layers on a given stage
-	SST_FIFO_DEPTH	-	SST_FIFO_DEPTH	Number of elements to be stored in the SST FIFO (at least images/input FMs)
-	SST_FIFO_GROUP	-	-	Implement each SST FIFOs as N FIFOs of width D_WIDTH·NUM_KERNEL/N
-	MACC_PIPE	-	-	Number of pipeline stages between the SST chain and MACC input
-	NUM_MACC_DSP	-	-	Number of MACCs (in total NUM_KERNEL·NUM_MACC) to be implemented using DSPs
-	MACC_SPREAD	-	-	Number of MUXs between the SST chain and MACC input
-	-	KERNEL_GROUP	-	Number of NUM_KERNEL to be grouped together
-	-	-	POOL_TYPE	Max or Min pooling

to the accelerator through a Peripheral Component Interconnect Express (PCIe) interface to copy both weights and input data to the FPGA external memory. The computation starts when all the weights are transferred, and the first image is streamed to the accelerator. This transfer happens only at the beginning of the first computation, after which the weights reside in the FPGA external memory, and a controller feeds the Weights Engine periodically. This solution allows to easily process multiple input frames in a batch, a common scenario in real applications, where models are used to process a collection – sometimes a continuous stream – of frames. As a result, the proposed acceleration methodology implements *inter-layer parallelism*: such parallelism is realized when the accelerator executes layers in a pipelined fashion, which intuitively means that while layer l is computing on a frame, layer $l-1$ is computing on the subsequent frame in the batch.

This is the general idea beyond inter-layer parallelism, though the actual implementation differs. Indeed, our architecture allows to cluster subsequent layers into *stages* according to resource and performance requirements. Each pipeline stage s implements one or more subsequent coarse-layers and has a certain total latency λ_s. Therefore, the total latency to process a single frame will be $\sum_{s \in S} \lambda_s$ as the frame needs to pass through all the S stages to be fully processed. However, as for a classical pipeline, at steady state each stage s will be processing a different input frame, therefore increasing the throughput since the slowest stage will give the time-per-frame, *i.e.* , $\max_{s \in S} \lambda_s$. It is thus important to keep $\max_{s \in S} \lambda_s$ as low as possible and balance all $\lambda_s \forall s \in S$, as will be detailed in Section 5. Figure 6 depicts the overall system, showing the accelerator pipelined architecture.

4.3 Parameters of HDL IPs

After providing a high-level description of the ISL and CNN architectures, we detail and summarize the customizable parameters of the proposed HDL IPs. Table 1 reports the parameters for ISL (column 1) and CNN (column 2, 3, and 4) IPs along with a brief description. The ISL IPs allow the user to implement 1D, 2D, or 3D stencil computations with different shapes and describe the PE behavior (i.e., the calculation). The user can also specify the latency of the PE. The IP relies on this parameter to synchronize filters, PE, and the output multiplexer, avoiding additional buffering and data stalls. After defining the configuration parameters, the IP setups the SST infrastructure.

The IPs for CNNs offer a large variety of customizable parameters. In particular, the user can specify and tailor the behavior of Convolution Kernel, Accumulation Core, and Pooling Core. After configuring such components, the remaining ones (e.g., the Memory Subsystem) are consequently adapted. The provided parameters permit tackle different aspects of the resulting architecture. For instance, parameters like FM_PARAL and LAYER_PARAL enable to increase the level of parallelism, and, consequently, the performance. On the other hand, MACC_PIPE and MACC_SPREAD target timing and place & route phase at the cost of resource usage. Finally, NUM_MACC_DSP is an example of a parameter oriented to resource balancing. Indeed, it defines the number MACCs to be implemented using DSPs, while the remaining ones are implemented using LUTs and FFs.

In conclusion, the proposed IPs supply multiple knobs to tune either the ISL or CNN architecture according to the user's needs. However, exploring such a large design space may be time-consuming, especially for CNN designs, given the number of parameters. Therefore, in the next Section, we describe a resource and performance model to support the design of a CNN accelerator.

5 RESOURCE AND PERFORMANCE MODEL

The proposed design methodology relies on different parallelism levels to efficiently exploit the available resources and guarantee low latency and high throughput. Given the vast number of available target platforms and CNN models, a careful analysis of both resources and performance is mandatory to create an efficient design. Indeed, a proper coarse-layer clustering and intra-layer parallelism are crucial to balance latency and resource utilization. Since CNVCs represent the key component of the design (they are the most resource-hungry and the actual performance bound), determining the number of *stages* and the degree of internal parallelism takes precedence over other Cores (their impact on the overall execution time is negligible). DSPs usage is a critical aspect of CNVC design since they represent the most constraining resource. Thus, the proposed model relies on their estimation for a good assessment of the feasibility of the design. Thanks to the proposed time-sharing, just one DSP is needed to perform an entire convolution, independently from the filter dimension. For a given CNVC, the total DSP usage is a function of the chosen intra-FM parallelism δ and intra-layer parallelism κ:

$$DSP_c = \delta \cdot \kappa \tag{1}$$

The designer could also offload part of the MACC computations inside Convolution Kernels to logic and consequently decrease the number of used DSPs. However, DSPs usage would still be significant. Hence Equation 1 remains a good approximation for evaluation purposes. The following Equation estimates the total number of clock cycles τ_c of a given CNVC:

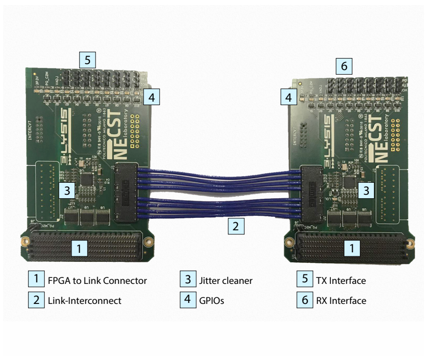
$$\tau_c = \begin{cases} \frac{(m_{l-1} \cdot n_{l-1}) \cdot (\rho_l \cdot \rho_l) \cdot |F_{l-1}| \cdot |F_l|}{\delta \cdot \kappa}, & \text{if } \sigma_l = 1 \\ \frac{(m_{l-1} \cdot n_{l-1}) \cdot (\rho_l \cdot \rho_l) \cdot |F_{l-1}| \cdot |F_l|}{\delta \cdot \kappa \cdot \sigma_l^2}, & \text{if } \sigma_l > 1 \end{cases} \tag{2}$$

where (m_{l-1}, n_{l-1}) represent the size – m rows and n columns – of the padded input feature-maps, σ_l the stride, $|F_{l-1}|$ and $|F_l|$ the cardinalities of respectively the input and output feature-maps and $(\rho_l \cdot \rho_l)$ the filter size. The total execution time T_c will simply be given by dividing τ_c by the target frequency H_c: $T_c = \tau_c / H_c$. Notice that the term $(\rho_l \cdot \rho_l)$ stems from the fact that the proposed implementation uses time-sharing. Equation 2 is valid as long as performance is computation-bound, *i.e.* , weights are loaded before the Memory Subsystem starts sending data to the Convolution Kernels. The following Equation clarifies this requirement:

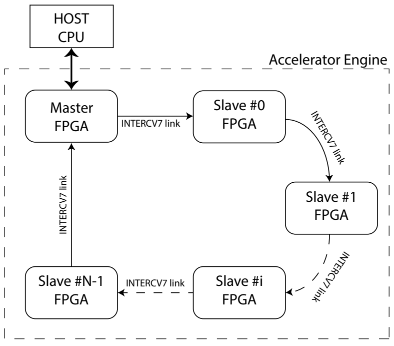
$$\tau_{w_c} \leq (\rho_l - 1) \cdot m_{l-1} \cdot (\rho_l \cdot \rho_l) \tag{3}$$

with τ_{w_c} being the total number of clock cycles needed to load a new set of weights η_{w_c} for a new iteration. η_{w_c} depends on the intra-FM and inter-layer parallelisms selected for the CNVC, *i.e.* $\eta_{w_c} = (\rho_l \cdot \rho_l) \cdot \delta \cdot \kappa$. The term τ_{w_c} will simply be given by dividing η_{w_c} by the number of weights ϵ_{w_c} the Weights Engine can transfer at each clock cycle, under this constraint:

$$\epsilon_{w_c} \geq \frac{\delta \cdot \kappa}{(\rho_l - 1) \cdot m_{l-1}} \tag{4}$$



(a) INTERCV7 link description



(b) Accelerator Block Diagram

Fig. 7. Multi-FPGA architecture and custom communication system

In other words, Eq. 4 defines the required off-chip memory bandwidth such that a given stage is computation-bound. Indeed, if Eq. 4 is not satisfied, then the Memory Subsystem will stall until every weight is loaded in the corresponding Convolution Kernel.

The designer can attempt a first subdivision into stages using a greedy approach to design the entire accelerator, clustering coarse-layers according to common features. The first is the size of the filter of the respective CNVLs, as it allows to reuse the same CNVC to implement different CNVLs. If the result is still too coarse, the designer can use the number of input and output feature-maps to break these bigger stages further. The intra-FM parallelism of each stage can be first set to match the highest number of input feature-maps among all the CNVLs belonging to the related stage. To obtain the final design, further refinements can be applied, such as swaps of coarse-layers between stages, tweaking the different internal parallelism, following these principles: minimizing the highest latency among stages, maximizing the achievable throughput, keeping the stage latency balanced, and the stages count as high as possible considering a feasible number of DSPs.

The model does not entirely take into account overheads introduced by off-chip memory data movements or host-to-accelerator transfers. The limited off-chip DDR memory bandwidth can indeed saturate when large batches are computed and can consequently increase the overall network latency. Nevertheless, it is crucial to understand that the purpose of the model is to guide the designer through the implementation process. Small deviations from the actual system performance are tolerable, especially if they are due to technology-dependent reasons. Moreover, the model can help estimate when a single FPGA cannot accommodate a specific CNN stage configuration. The model is resolved through the algorithm described in [58], which just solves the model described to test the mapping feasibility on a single device. If the single device mapping is unfeasible, a possible solution is to map the stages on multiple FPGAs according to both the available resources (especially DSPs) on each device and the model estimations.

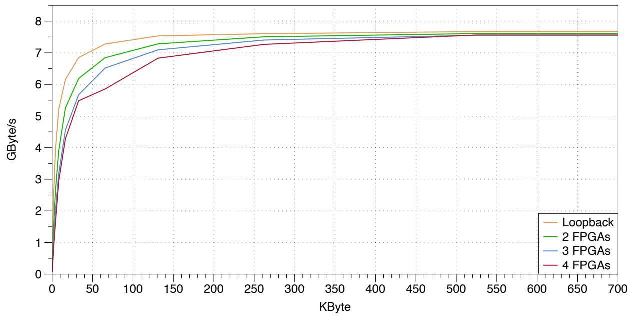
6 DISTRIBUTED HARDWARE ARCHITECTURE

So far, we have described optimized hardware accelerators for both ISLs and CNNs. These accelerators are a good fit for a single FPGA system, but their performance can significantly scale when implemented on *multi-FPGA systems*. Indeed, a multi-FPGA system represents an efficient way to boost the performance of hardware accelerators, as demonstrated by both industry [13, 72] and academic [61, 88] solutions. The computational pattern of both ISLs and CNNs considerably benefits from a heavy pipelining, as we show in Section 7. For this reason, we implemented a distributed architecture composed of multiple FPGAs interconnected through a ring-based topology. The interconnection between subsequent FPGAs is implemented through a custom communication system, named *INTERCV7 link*. Given the fundamentals described in Section 2.3, this link, depicted in Figure 7a, is composed of one transmitter interface board *Tx*, one receiver interface board *Rx*, and one *high-speed link-interconnect* to connect *Tx* and *Rx*. We chose the *Samtec DCH-08-04.0-T-PF-1-1* High-Speed Press-Fit connector as a link-interconnect, which offers eight 100 ohm differential pair serial links, each supporting up to a 56Gbsp data rate. With this setup, we successfully transmit data at the target data rate (i.e., 8Gbps per lane), exploiting different cable lengths ranging from 15 to 30 cm. Both the *Tx* and *Rx* boards rely on a *FPGA to Link Connector* based on a FPGA Mezzanine Card (FMC) to communicate with the reference nodes: this connection leverages on eight Multi-Gigabit Transceivers (MGTs) to provide an aggregate peak bandwidth of 8 GB/s to the *INTERCV7 link*. The reference clock of each MGT is provided through a jitter cleaner mounted on the custom boards to enhance the signal integrity. Specifically, the transmission clock is initially generated on the FPGA fabric, forwarded to the jitter cleaner through the FMC, and routed to the MGT reference clock through the FMC. The jitter cleaner provides to the TX MGT a clock whose jitter respects the requirements to obtain an optimal signal transmission through a correct Clock and Data Recovery (CDR) on the receiver side. Along with the MGTs, the FMC is used to connect several FPGA General Purpose Input/Outputs (GPIOs), whose purposes vary from setting the jitter cleaner parameters – such as its reference clock frequency –, to controlling up to eight Light Emitting Diodes (LEDs) and GPIOs, as well as providing the board power supplies.

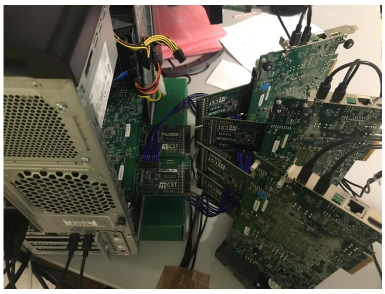
To allow the *INTERCV7 link* to be used by the accelerators, we implemented an IP core, namely *INTERCV7 CORE*, which leverages the Xilinx Aurora 64b/66b link-layer protocol to exchange data among nodes. In particular, the *INTERCV7 CORE* implements and correctly handles two Aurora 64b/66b IP cores, each providing a simplex communication to perform data reception from the preceding node and data transmission to the subsequent one. Furthermore, the *INTERCV7 CORE* is in charge of configuring the jitter cleaner parameters and status LEDs and performing data gearboxing to match the accelerator data rate. The total resource usage of the *INTERCV7 CORE* is very low, as it consumes roughly 3000 LUTs and 8000 FFs, on a *Xilinx Virtex-7 XC7VX485T* FPGA.

As Figure 7b shows, one *INTERCV7 link* performs a simplex connection between two subsequent *nodes* of the accelerator, where each node represents one FPGA. We chose a simplex-ring topology to maximize the throughput between subsequent nodes, creating a deep pipeline that fully exploits the benchmark computational pattern. Indeed, we devoted all the MGTs offered by the FMC connector for the simplex communication with the subsequent accelerator *nodes*. Although other topologies (e.g., mesh or full-duplex ring) could have been more versatile in terms of communication possibilities, they would have limited the total bandwidth achievable among subsequent nodes, thus limiting the overall performance. Moreover, the *INTERCV7 link* design has been chosen among a backplane-based one to guarantee modularity to the system. Since each *INTERCV7 link* is entirely independent of the others, it allows connecting an arbitrary number of accelerators to the chain.

Figure 7b also shows the complete distributed hardware architecture. The host CPU communicates with the acceleration engine through a PCIe interface. To this purpose, we implemented a multi-threaded application to bridge the communication of data buffers between the host code and the PCIe driver residing in kernel space. This application is in charge of allocating the number of packets, and the packet dimension, for proper data communication through the PCIe interface. The FPGA-to-host packet dimension is then forwarded to a HDL-written *PCIe controller engine*, which prepares suitable packets on the accelerator side and implements watchdogs to transmit dummy data in case of hardware stalls. The host processor can also reside inside the *Master FPGA* as soft-core. In this case, the host processor and the acceleration engine communicate through the off-chip DDR memory. The soft-core handles Direct Memory Access (DMA) interfaces to load/store



(a) INTERCV7 link bandwidth scaling with input size



(b) Multi-FPGA System view

Fig. 8. Bandwidth scaling and Real System Picture

data directly from/to the board off-chip DDR memory. This approach is more performing than the PCIe-based one, as the bandwidth offered my the onboard DDR is larger than the PCIe Gen2 x8 provided by the target device. As a downside, the FPGA resources used by the DDR-based solution are more significant than the PCIe-based one. In both cases, the Master FPGA is in charge of handling the communication with the host processor and hosting part of the accelerator logic and an *INTERCV7 link*. A series of *Slave FPGAs* are then serially connected to form the daisy-chain and contain both the accelerator logic and the *INTERCV7 CORE*.

In the proposed work, the CNN experimental results are collected exploiting the PCIe-based interface. In contrast, the ISL architecture relies on the *Xilinx Microblaze* soft-core to forward data to the acceleration engine. This choice permits to extract performance from both the considered accelerators fully. Indeed, even though a host could offload a single stencil-based problem or a batch to the multi-FPGA system via PCIe, the high bandwidth exploited by the soft-core based solution allows better to validate the ISL accelerator performance in terms of scalability. The CNN accelerator can instead exploit full performance even in smaller off-chip data bandwidth, as the adopted time-sharing approach highly lowers the off-chip bandwidth requirements.

6.1 Bandwidth Analysis

We analyzed the bandwidth of the proposed multi-FPGA architecture on a system composed of a set of *Xilinx Virtex VC-707* evaluation board, each containing a *Virtex-7 VX485T* FPGA. These FPGAs are daisy-chained through the *INTERCV7 link* to create a scalable computing pipeline, and the Master board also exposes a connection with the host processor. As stated before, the host processor can either reside on-chip (as a soft-core) or be an external CPU communicating with the Master board via PCIe. The proposed architecture is intrinsically scalable since the number of compute engines added to the chain is theoretically infinite. Each node is completely independent of its neighbors, as each FPGA couple is connected through an independent *INTERCV7 link*. If needed, as in the CNN architecture, both the Master and the Slave accelerators can rely on the off-chip DDR memory to load and store coefficients or partial results, streamed by the host processor at the beginning of the first application execution.

Each *INTERCV7 link* exploits 8 lanes, capable of providing an aggregate peak bandwidth of roughly 8 GB/s. On top of the physical MGTs, the *INTERCV7 link* uses the Xilinx*Aurora 64B/66B* IP core to perform Serial In Parallel Out (SIPO) and Parallel In Serial Out (PISO) data conversions, data encoding, decoding and data scrambling. The 64B/66B data encoding introduces a 3% transmission overhead, as detailed in [31]. For this reason, the inter-FPGA peak bandwidth is 7.75 GB/s. This

Table 2. ISL benchmarks

Benchmark	Computation	Problem size	Chained iterations
Jacobi-1D [8, 23]	$1/3 \cdot (A[i-1] + A[i] + A[i+1])$	1.04×10^6	362
Jacobi-2D [8, 23]	$1/5 \cdot (A[i-1][j] + A[i][j-1] + A[i][j] + A[i][j+1] + A[i+1][j])$	1024×1024	191
Jacobi-3D [68]	$1/7 \cdot (A[i-1][j][k] + A[i][j-1][k] + A[i][j][k-1] + A[i][j][k] + A[i][j][k+1] + A[i][j+1][k] + A[i+1][j][k])$	$64 \times 64 \times 64$	102
Heat-1D [8, 71]	$0.125 \cdot (A[i+1] - 2.0 \cdot A[i] + A[i-1])$	1.04×10^6	198
Heat-2D [8, 71]	$0.125 \cdot (A[i+1][j] - 2.0 \cdot A[i][j] + A[i-1][j]) + 0.125 \cdot (A[i][j+1] - 2.0 \cdot A[i][j] + A[i][j-1]) + A[i][j]$	1024×1024	89

bandwidth is in line with other works in the literature, like [48], whose multi-FPGA communication subsystem leverages on QSFP+ transceiver links available on Terasic DE5A-NET boards. Figure 8a shows the average bandwidth reached by the system, considering a chain whose length ranges from one – *i.e.* , loopback configuration – to four nodes. Moreover, the packet length needed to saturate the chain bandwidth increases with the number of the chained FPGAs. This is because each *INTERCV7 link* adds an average of $0.528\mu s$ latency to the chain, whose value affects the overall bandwidth with small packet length. While *Aurora 64B/66B* supports up to 16 lanes, each FMC connector on the *Xilinx Virtex VC-707* Evaluation board exposes 8 MGTs, and is the most equipped connector of the target platform; for this reason, the achieved inter-FPGA communication bandwidth coincides with the maximum allowed by the evaluation board.

7 EXPERIMENTAL SETUP AND EVALUATION

The experimental evaluation targets the system described in Section 6. We used *INTERCV7 links* to chain various *Xilinx Virtex VC-707* boards, as shown in Figure 8b: one master FPGA handling both the communication with a host system through a Gen.2 x8 PCIe and part of the computation, and multiple slave FPGAs. Each FPGA is equipped with 303600 LUTs, 607200 FFs, 1030 Block RAMs (BRAMs) and 2800 DSP. The host system exploits an *Intel i7-6700* Central Processing Unit (CPU) processor equipped with 32 GB of DDR4 RAM and runs Centos 7. We relied on the Vivado 2017.2 HLx design suite to obtain the final bitstream and extract power consumption estimates.

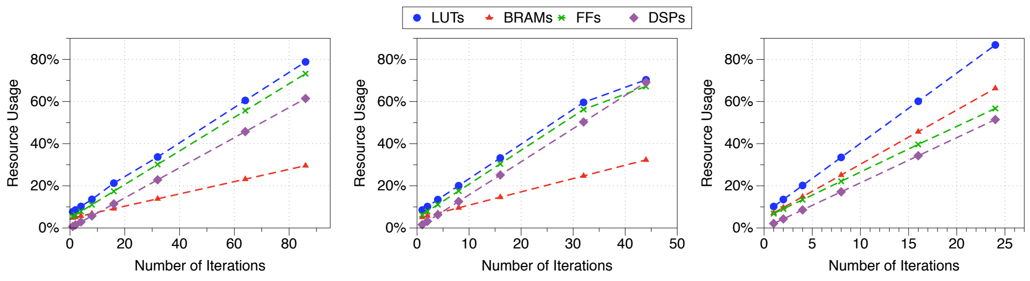
7.1 ISL Experimental Results

The proposed micro-architecture is designed to provide a set of templates able to compute several stencil algorithms. Hence, the same component can be used to create all the stencil algorithms having the same data dimension - *i.e.* , 1D, 2D, or 3D - and the same stencil shape. This experimental evaluation considers 5 of the most common ISL algorithms found in literature, having a total of three different data dimensions and three stencil shapes. In particular, we selected and adapted the following benchmarks from Polybench [23], Parboil [68], and Pochoir [71]: *Jacobi-1D*, *Jacobi-2D*, *Jacobi-3D*, *Heat-1D*, and *Heat-2D*. For each benchmarks, we considered both a single-FPGA and a multi-FPGA system composed of four boards. Table 2 reports the computation performed by each benchmark, the problem size, and the number of iterations we chained on 4 FPGAs. We empirically measured the number of SSTs each board could accommodate according to the number of resources required by a single SST, the I/O IPs, and the softcore (for the master FPGA).

Table 3 shows the performance and power efficiency – normalized against the software baseline – we obtained using the maximum number of iterations for 1 and 4 FPGAs. We experimentally extracted results for both *Jacobi-1D*, *Jacobi-2D* and *Jacobi-3D*, while we computed the performance of *Heat-1D* and *Heat-2D* starting from the *Jacobi-1D* and the *Jacobi-2D* execution times. The estimation performed on *Heat-1D* and *Heat-2D* is fair, as we implemented the algorithms starting from the same components used to compute *Jacobi-1D*, *Jacobi-2D*, and both *Jacobi* and *Heat* share

Table 3. ISL results and comparison with related works

	Architecture				Benchmarks									
					Jacobi-1D		Jacobi-2D		Jacobi-3D		Heat-1D		Heat-2D	
	Type	**Model and Technology**	**Freq.** [MHz]	**Designed with**	**Norm. Perf.**	**Pow. Eff.** [GFLOPS/W]	**Norm. Perf.**	**Pow. Eff.** [GFLOPS/W]	**Norm. Perf.**	**Pow. Eff.** [GFLOPS/W]	**Norm. Perf.**	**Pow. Eff.** [GFLOPS/W]	**Norm. Perf.**	**Pow. Eff.** [GFLOPS/W]
Baseline [6]	CPU (1 thread)	Xeon E5-2680 v2 (32nm)	2800	C	1	-	1	-	1	-	1	-	1	-
Optimized [6]	CPU (40 threads)	Xeon E5-2680 v2 (32nm)	2800	C	22.08	-	5.30	-	6.90	-	12.44	-	14.62	-
[78]	1 FPGA	ADM-PCIE-7V3 (20nm)	200	HLS	-	-	31.59	-	-	-	-	-	-	-
[12]	1 FPGA	Virtex-7 XC7VX485T (28nm)	200	HLS	-	-	6.67	-	0.88	-	-	-	-	-
[51]	2 FPGAs	Virtex-7 XC7VX485T (28nm)	n.a.	HLS	-	-	8.94	4.64	1.90	0.82	-	-	-	-
[61]	1 FPGA	Stratix III EP3SL150 (65nm)	133	HDL	-	-	9.74	1.52	10.67	1.31	-	-	-	-
[61]	9 FPGAs	Stratix III EP3SL150 (65nm)	133	HDL	-	-	74.61	1.3	78.81	1.07	-	-	-	-
[14]	1 FPGA	ADM-PCIE-KU3 (20nm)	250	HLS	-	-	30.12	-	10.32	-	-	-	-	-
This Work	1 FPGA	Virtex-7 XC7VX485T (28nm)	200	HDL	61.87	4.23	46.06	7.01	22.10	3.33	17.02	3.15	15.86	6.8
This Work	4 FPGAs	Virtex-7 XC7VX485T (28nm)	200	HDL	261.42	4.52	171.22	5.91	60.33	2.40	70.46	3.73	60.42	5.51



(a) Jacobi-1D (b) Jacobi-2D (c) Jacobi-3D

Fig. 9. Resource usage of the Jacobi stencils

a similar latency on the PEs in terms of clock cycles. We performed software benchmarks running code from [6] on a server-grade dual-CPU system consisting of 2 Intel Xeon E5-2680 v2 with 380GB of RAM. The baseline is single-thread, while the optimized one runs on 40 threads. The employed benchmarks exploit the polyhedral compiler Pluto [7, 9, 10] with diamond tiling enabled [6], with the best tile size found by empirical analysis. Regarding [12, 14, 51, 61, 78], we reported their best results for the common benchmarks presented in their works, and, when available, the power efficiency. We excluded works like [76, 77], as they use variants of the selected benchmarks.

The proposed hardware accelerators perform largely better than the optimized CPU implementation [6], even on a single FPGA. Our solution also considerably outperforms [12, 51], which rely on the SST micro-architecture to compute ISLs too, in terms of both performance and scalability. Indeed, while [12] is able to chain up to 8 *Jacobi-3D* timesteps on a single chip, the proposed accelerator hosts 24 iterations on the master FPGA. When tiling is applied, [78] shows performance similar to our single-FPGA. However, this optimization causes performance degradation when the number of fused iterations increases, mainly driven by port contention on the on-chip memory, limiting scalability to multiple devices, as performance can be quickly bound by data movement. Moreover, our solution performs better than the related works when considering multi-FPGA results: indeed, although Sano et al. [61] use 9 FPGAs (we use 4), they only reach a 1.3× speedup for *Jacobi-3D*, while they get a 2.3× slowdown for *Jacobi-2D*. In terms of power efficiency, the proposed approach surpasses all the other works [51, 61] both in the single and multi-FPGA case. It is worth to note that the previous works target either our very same FPGA family (*Xilinx Virtex 7*) [12, 51, 78], or similar devices [14, 78], which allow a fair comparison. Although Sano et al. [61] employ older technology, they achieve significant results on Jacobi-3D, both on 1 and 9 FPGAs, thanks to their HDL implementation. Therefore, our performance comes primarily from our HDL-based approach and secondly from improved technology. Finally, Figure 9 reports the resource usage of the master

Table 4. Parameters of the AlexNet network (estimated execution times with a clock frequency of 200MHz)

Stage	Coarse-Layer	Input FMs	Output FMs	Input Size	Output Size	Filter Size	Pad	Stride	Intra FM Paral.	Intra Layer Paral.	DSPs	Clock Cycles	Coarse-Layer Ex Time [ms]	Stage Ex Time [ms]
0	1	3	96	227	27	121	0	4	3	96	288	392909	1.96	1.96
1	2	96	256	27	13	25	2	1	96	16	1536	399776	2.00	2.00
	3	256	384	13	13	9	1	1				108000	0.54	
2	4	384	384	13	13	9	1	1	128	16	2048	162000	0.81	1.89
	5	384	256	13	13	9	1	1				108000	0.54	

FPGA (accelerator and I/O IPs) and shows the linear relation between resource usage and the chain length inside the acceleration engine.

7.2 AlexNet Design Space Exploration

To validate the proposed work, we first chose AlexNet, one of the most implemented CNNs in the literature. Thanks to its relatively large number of CNVLs, the feature-extraction stage of AlexNet can benefit from a multi-FPGA acceleration and fully exploit the flexibility of the IP library, as several layers use different hyperparameters. Besides, AlexNet architecture contains CNVLs with different filter dimensions, requiring the instantiation of multiple *stages*, as stated in Section 4.2. The proposed architecture performance tightly depends on the parallelism applied to each *coarse-layer*. AlexNet can be divided into 5 coarse-layers, each one composed of a convolutional, an activation, and a requantization layer, and, if available, also a pooling layer. The first coarse-layer requires an 11×11 filter, the second a 5×5, and the remaining ones a 3×3. The parallelism imposed on each coarse-layer must be selected to guarantee almost the same latency for all stages while maximizing the available chip resources. This is a crucial design choice to avoid resource wasting and computational stalls on faster layers, which may need to wait for data from slower ones. We can directly evaluate the latency of each stage using Equation (2). When we apply this expression, it is worth considering that the sum of the DSPs each stage requires – *i.e.* , the sum of Equation (1) applied to each layer – must be less or equal to the number of DSPs available on the target architecture. For the AlexNet network, the choice of intra-layer and intra-FM parallelism is driven in particular by layer 1, which is the performance bottleneck, a consequence of the size of its input feature-maps and the size of its filter.

The associated stage is completely parallelized, while the other stages are just set to achieve a similar latency. Specifically, since the first layer has a total of 3 input feature-maps and 96 output feature-maps, even setting both intra-layer and intra-FM parallelism to their maximum value – *i.e.* , 96 and 3 respectively – the total number of DSPs assigned to the first CNVL is, applying Equation (1), equal to 288. A small number of input feature-maps on the first CNVL is a typical case on CNNs, as they represent the three *RGB* channels of the input image. If some subsequent layer in the network has the same filter dimension as the first layer, we can mask this bottleneck by merging multiple coarse-layers in a single stage. However, this masking cannot be applied in AlexNet since the first and the second CNVLs have different filter dimensions. As a result, the first stage, namely *S0*, is only composed of the coarse-layer of the first layer and exploits the maximum allowable intra-FM and intra-layer parallelism. As stated in Section 5, the sizing of the remaining cores is directly inferred from the parallelism imposed to the CNVC, since it consumes most of the resources and is the biggest contributor to the total stage latency.

The second stage *S1* consists of the second coarse-layer, as its convolutional stage is the only one that implements a 5×5 filter: in this case, both intra-layer and intra-FM parallelism has been

[2]It refers to the implementation of a single convolutional stage, and not the entire network.

[3]It refers to the last three convolutional stages, each evaluated separately, and not the entire network.

Table 5. AlexNet results and comparison with related works

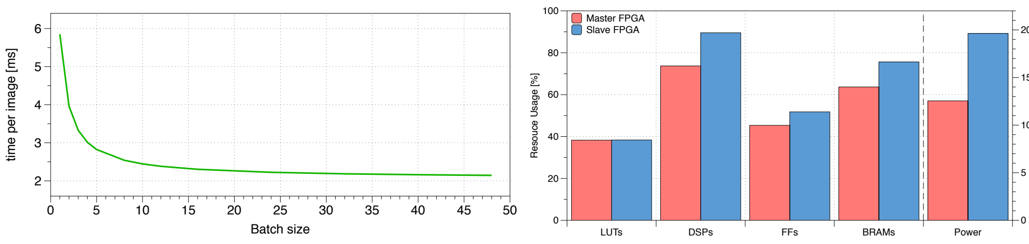
	Device and Technology	Designed with	Precision	Frequency [MHz]	Power [W]	DSP Util.	Batch Size	Latency [ms]	Throughput [GOPS]	Power Eff. [GOPS/W]
[80]	Arria 10 GT1150 (20nm)	HLS	32-bit float	239.62	-	3036	-	4.05	406.1	-
[41]	Zynq XCZU9EG (16nm)	HLS	16-bit fixed	200	23.6	2520	64	-	1006.4	42.64
[44]	Stratix V GXA7 (28nm)	HDL	16-bit fixed	100	-	512	-	9.92	134.10	-
[84]	Stratix V GXA7 (28nm)	HDL	16-bit fixed	200	-	512	-	-	780.60	-
[64]	Arria 10 GX115 (20nm)	HDL	16-bit fixed	200	-	3036	-	-	265.36	-
[64]	Zynq XC7Z020 (28nm)	HDL	16-bit fixed	150	-	220	-	-	20.16	-
[4]	Arria 10 A10-1150 (28nm)	HLS	32-bit float	303	-	1476	-	-	1382[2]	-
[74]	Zynq XC7Z045 (28nm)	HLS	16-bit fixed	125	-	900	-	8.22	161.98	-
[79]	Zynq XC7Z045 (28nm)	HDL	16-bit fixed	100	-	900	-	12.30	108.25	-
[22]	Zynq XC7Z045 (28nm)	HDL	16-bit fixed	250	9.48	-	-	10.20	116.5	12.3
[85]	UltraScale KU060 (20nm)	HLS	16-bit fixed	200	-	2760	-	-	163.00	-
[86]	Virtex-7 VX485T (28nm)	HLS	32-bit float	100	18.6	2240	-	22.01	61.62	3.31
[70]	Stratix V GS D8 (28nm)	HLS	8-bit fixed	-	19.1	727	-	20.1	72.4	3.80
[87]	Intel QuickAssist QPI (28nm)	n.a.	32-bit float	200	13.18	224	-	24.04	83	6.30
[40]	Virtex-7 XC7VX690T (28nm)	n.a.	16-bit fixed	156	30.2	2144	-	2.56	565.94	18.74
[42]	Virtex-7 XC7VX690T (28nm)	HDL	16-bit fixed	100	24.8	1436	-	-	222.1	8.96
[90]	ADM-PCIE-8K5 (20nm)	HDL	16-bit fixed	220	22.9	4854	-	-	1633	71.31
[2]	Virtex-7 XC7VX485T (28nm)	HDL	16-bit fixed	200	26.91	2304	-	0.20	3331.6[3]	123.81
[88]	**4x** Virtex-7 XC7VX690T (28nm)	n.a.	16-bit fixed	150	126	-	16	104	825.6	6.55
This Work	**2x** Virtex-7 XC7VX485T (28nm)	HDL	8-bit fixed	200	32.18	4182	48	2.14	1106.54	34.39

tuned to obtain a latency similar to the one of *S0*. A convenient design rule is to set the intra-FM parallelism equal to the number of input feature-maps of the CNVL. Thus avoiding data replication on the Data Movement FSMs or an additional accumulation stage to get a set of output feature-maps. Therefore, the *S1* intra-FM parallelism has been set to 96 and, by applying Equation (2) to its CNVC, the resulting intra-layer parallelism to match the *S0* latency is 16. By applying Equation (1), *S1* consumes a total of 1536 DSPs. The remaining coarse-layers can be implemented either with three separate stages or with a single stage that clusters them. Since treating the three stages separately would require three concurrent DDR accesses to perform weights loading, leading to sub-optimal results of DDR bandwidth and resource utilization, they have been clustered in one stage *S2*, as these three layers share the same filter dimension. A feedback mechanism, composed of MUXs and DEMUXs has thus been implemented to reuse the same CNVC to compute both the third, the fourth, and the fifth coarse-layers. Since these three layers have different input feature-map counts – *i.e.* , 256, 384, and 384 respectively –, a feasible solution has been to impose intra-FM parallelism of 128 and accumulate the partial CNVC outputs to obtain the output feature-maps. In other words, several iterations – *i.e.* , two for the third layer and three for the remaining – of the CNVC are needed to obtain a set of valid output feature-maps. To match the total *S2* latency with the other pipeline stages, its intra-layer parallelism has been set to 16, for a total DSP count of 2048.

Table 4 summarizes the details about each stage, along with the DSP count and the estimation of the needed clock cycle, obtained from Equation (1) and Equation (2) respectively. This configuration needs a total of two FPGAs: the first – *i.e.* , the Master – implements *S0* and *S1*, as well as the communication with the host system through the PCIe interface; the second FPGA – *i.e.* , the Slave – implements the stage *S2*. Following the performance model (Section 5), each *stage* should compute a complete image in, at most, 2.00 ms, considering a working frequency of 200 MHz; hence, at steady-state, the created pipeline should produce a new result every 2.00 ms. Indeed, the pipelined execution of the staged architecture reduces the time-per-frame to the slowest stage latency.

7.3 AlexNet Experimental Results

We evaluated AlexNet with an increasing image batch, ranging from 1 to 48 images, as the obtained latency remains constant when larger batches are applied (Figure 10a). The multi-FPGA system implements the CNVLs, while the host CPU executes the FCLs. The purpose of the CPU part



(a) Throughput with several batch sizes (b) Resource usage and power consumption

Fig. 10. Results of the proposed AlexNet implementation

is to complete the network flow to check the correctness of the classification. Thus, the latency measurement accounts only for the execution of the CNVLs, as it is the focus of this work.

At first, the host processor sends all the network weights through the PCIe interface. We do not consider this DDR initialization time when we measure the latency because it happens only once. After sending all the weights, the host starts to offload consecutive images to the acceleration engine and waits until all images have been processed. The average execution time is 5.85 ms for a batch size of one, representing the total pipeline latency, and it is perfectly in line with the sum of the execution times reported in Table 4. As the batch size increases, the computational pipeline fills up and increments the system throughput. The implemented network produces a valid set of output feature-maps every 2.14 ms at steady state, which deviates from the performance model by an additional 7%, probably caused by the off-chip data movement or host-to-accelerator overhead.

Figure 10b reports resource and power consumption of both the master and the slave FPGAs, and takes into account the resource utilization of the whole design, from the I/O interface to the accelerator. DSPs represent the critical resource. Indeed, as estimated through Equation (1), the first CNVC uses 288 DSPs, the second consumes 1536 DSPs while the third, hosted on the Slave FPGA, employs 2048 DSPs, with a total of 3872 DSPs to perform the convolution. The Requantization Cores use the remaining DSPs for requantization and bias addition.

Table 5 shows a comparison with the highest-performing subset of the works discussed in both Section 3.2 and state-of-the-art surveys [25, 75]. Results in terms of throughput – *i.e.* , Giga Operations per Second (GOPS) – have been evaluated as the total number of performed operations over the average execution time. The number of operations accounts for both the Convolution and the Requantization Cores. The performance reported in Table 5 refer to CNVLs only.

The proposed work outperforms, throughput-wise, the majority of the works reported in Table 5, but [2, 4, 90]. The throughput reported in [4] is the average result obtained by implementing each CNVC independently on the target FPGA, without considering FPGA reconfiguration time, which can take several tens of milliseconds. Ahmad et al. [2] followed a similar approach, implementing a convolution engine and evaluating its performance on the last three layers of AlexNet separately. On the other hand, we believe that the throughput difference between our work and [90] mainly relies on the different technology of FPGA. Indeed, the ADM-PCIE-8K5 board features a KU115 FPGA, which comprises two dies of resources. As a result, it offers almost as many DSPs as two Virtex-7 XC7VX485T (5520 versus 5600). Moreover, the KU115 FPGA features an Ultrascale architecture, more recent and performing than the one available on our board. Finally, it is worth to notice that works like [2, 4, 41] achieve a high throughput also thanks to the Winograd transformation, which simplifies complex operations and increases the efficiency of CNVL computations. The obtained results clearly show the advantages of the inter-layer parallelism as, once the computational pipeline

Table 6. Parameters of the VGG16 network (estimated execution times with a clock frequency of 200MHz)

Stage	Coarse-Layer	Input FMs	Output FMs	Input Size	Output Size	Filter Size	Pad	Stride	Intra FM Paral.	Intra Layer Paral.	DSPs	Clock Cycles	Coarse-Layer Ex Time [ms]	Stage Ex Time [ms]
0	1	3	64	224	224	9	1	1	3	64	2048	510760	2.55	7.66
	2	64	64	224	224	9	1	1	128	16		1021520	5.11	
1	3	64	128	112	112	9	1	1	128	16	2048	519840	2.60	7.80
	4	128	128	112	112	9	1	1	128	16		1039680	5.20	
2	5	128	256	56	56	9	1	1	128	16	2048	538240	2.69	8.07
	6	256	256	56	56	9	1	1	256	8		1076480	5.38	
3	7	256	256	56	56	9	1	1	256	8	2048	1076480	5.38	8.26
	8	256	512	28	28	9	1	1	256	8		576000	2.88	
4	9	512	512	28	28	9	1	1	512	4	2048	1152000	5.76	5.76
5	10	512	512	28	28	9	1	1	512	4	2048	1152000	5.76	5.76
6	11	512	512	14	14	9	1	1	512	4	2048	327680	1.64	4.92
	12	512	512	14	14	9	1	1	512	4	2048	327680	1.64	
	13	512	512	14	14	9	1	1	512	4	2048	327680	1.64	

is filled up, it can provide high performance. Indeed, using multiple batch of images is a common scenario for this type of application.

7.4 VGG16 Design Space Exploration

After evaluating AlexNet, we estimated the performance our CNN micro-architecture could reach on another well-known CNN, namely VGG16. As stated in Section 7.2, all the filters of VGG16 share the same dimension. This feature implies that we can approach the implementation of VGG16 on FPGA in two ways. On the one hand, we could leverage the shared filter dimension and implement the whole network with only one stage on a single FPGA. On the other hand, we could exploit the multi-FPGA system and implement VGG16 with multiple stages. According to our model, if we wanted to instantiate an optimal and balanced configuration of VGG, we would need 7 FPGAs. Unfortunately, we do not own that amount of FPGAs to evaluate this solution; therefore, we decided to estimate the performance of a 7-FPGA system using our model. This approach is a general solution in the literature to make performance projections, especially when the required hardware is not available [61, 85]. However, it is crucial to notice that we could also implement VGG16 either on a single board system, as stated before, or on a 4-FPGA system, as we did for the ISLs. Nevertheless, this would not allow us to fully exploit the potential of our CNN micro-architecture or the performance scaling of a multi-FPGA system.

We can divide VGG into 13 coarse-layers, each one comprising a convolutional, an activation, a requantization, and a potential pooling layer. Each coarse-layer requires a 9×9 filter. We followed an approach similar to the one applied to AlexNet and imposed proper parallelism on each coarse-layer to balance the latency of each stage and maximize resource usage. We estimated the latency of each stage with Equation (2). We completely parallelized the first coarse-layer, which requires a relatively small amount of DSPs, *i.e.* , according to Equation (1). Given the small latency of the first coarse-layer (2.55 ms), we combined it with the following coarse-layer and created *stage 0 (S0)*. Hence, we tuned the intra-layer and intra-FM parallelism of the second layer according to the available DSPs. S0 requires 2048 DSPs and its latency is 7.66 ms. We applied a similar approach for what concerns the following coarse-layers and balanced their parallelism to maintain a latency similar to *S0*. In particular, it was possible to distribute equally coarse-layers from 3 to 8 in three stages (*S1*, *S2*, and *S3*). On the other hand, we implemented coarse-layers 9 and 10 in two different stages (*S4* and *S5*), due to the number of required DSPs and their latency. Finally, *S6* contains the remaining three coarse-layers. This option was possible thanks to the similar structure of these coarse-layers and the relatively small latency we obtained. This configuration would need a total of seven FPGAs, one for each stage. In particular, the Master FPGA would implement not only

Table 7. Estimated VGG16 results and comparison with related works

	Device	Designed with	Precision	Frequency [MHz]	Power [W]	DSP Util.	Batch Size	Latency [ms]	Throughput [GOPS]	Power Eff. [GOPS/W]
[74]	Zynq XC7Z020 (28nm)	HLS	16-bit fixed	125	-	220	-	633.01	48.53	-
[74]	Zynq XC7Z045 (28nm)	HLS	16-bit fixed	125	-	900	-	239.5	155.81	-
[64]	Zynq XC7Z020 (28nm)	HDL	16-bit fixed	150	-	220	-	-	31.35	-
[24]	Zynq XC7Z045 (28nm)	n.a.	16-bit fixed	150	9.63	900	-	-	187.8	19.5
[24]	Zynq XC7Z020 (28nm)	n.a.	8-bit fixed	214	3.5	190	-	-	84.3	24.09
[85]	UltraScale KU060 (20nm)	HLS	16-bit fixed	200	-	1058	-	-	310	-
[85]	Virtex-7 XC7VX690T (28nm)	HLS	16-bit fixed	150	-	2833	-	-	488	-
[64]	Stratix V SGSD5 (28nm)	HDL	16-bit fixed	200	-	3180	-	-	157.39	-
[64]	Arria 10 GX115 (20nm)	HDL	16-bit fixed	200	-	3036	-	-	390.02	-
[44]	Stratix V GXA7 (28nm)	HDL	16-bit fixed	150	-	512	-	87.87[4]	352.24[4]	-
[44]	Arria 10 GX115 (20nm)	HDL	16-bit fixed	200	-	3036	-	42.98[4]	720.15[4]	-
[80]	Arria 10 GT115 (20nm)	HLS	16-bit fixed	231.85	-	3036	-	26.85[4]	1171.30[4]	-
[84]	Stratix V GXA7 (28nm)	HDL	16-bit fixed	200	-	512	-	-	669.10[4]	-
[41]	Zynq XCZU9EG (16nm)	HLS	16-bit fixed	200	23.6	2520	32	-	2601.3	110.22
[43]	Arria-10 GX 1150 (20nm)	HDL	8/16-bit fixed	150	21.2	1518	-	47.97[4]	645.25[4]	30.43
[54]	Intel QuickAssist QPI (28nm)	n.a.	32-bit fixed	200	8.04	256	-	142.3	229.2	28.5
[65]	Virtex-7 XC7VX690T (28nm)	HLS	16-bit fixed	150	25	1376	-	-	570[4]	22.8
[65]	XCVU440 (20nm)	HLS	16-bit fixed	200	26	1376	-	-	821[4]	31.57
[70]	Stratix V GS D8 (28nm)	HLS	8/16-bit fixed	120	19.1	-	-	-	136.5	7.15
[87]	Intel QuickAssist QPI (28nm)	n.a.	32-bit float	200	13.18	224	-	-	123.48	9.37
[89]	Arria10 GX1150 (20nm)	HLS	16-bit fixed	385	37.46	2756	-	-	1790[4]	47.78
[73]	Virtex-7 XC7VX485T (28nm)	n.a.	8-bit fixed	160	-	2688	-	46.9	660	-
[90]	ADM-PCIE-8K5 (20nm)	HDL	16-bit fixed	235	-	4318	-	-	2011	-
[2]	Virtex-7 XC7VX485T (28nm)	HDL	16-bit fixed	200	26.91	2304	-	8.47	3623.9[5]	134.67
[1]	Virtex-7 XC7VX485T (28nm)	n.a.	32-bit fixed	200	36.32	2736	-	28.05	1094.3	30.13
[88]	**6x** Virtex-7 XC7VX690T (28nm)	n.a.	16-bit fixed	150	160	-	2	200.9	1280.3	80.02
This Work	**7x** Virtex-7 XC7VX485T (28nm)	HDL	8-bit fixed	200	-	18432	-	8.84[6]	3669.39[6]	-

S0 but also the host system through the PCIe interface. According to the performance model, the steady-state pipeline at 200 MHz should produce a new output every 8.26 ms, *i.e.* , the latency of the slowest stage. Table 6 details the parameters of the discussed configuration of VGG.

Please note that it would theoretically be possible to improve the performance of VGG further. For instance, we could map each coarse-layer to a different FPGA and design a 13-FPGA system. However, this configuration would unbalance the pipeline of stages.

7.5 VGG16 Experimental Results

Table 6 reports the latency estimation of each stage of our multi-FPGA pipeline for VGG. When the batch size is one, the average estimated execution time is 48.23 ms. As shown in Section 7.3, the performance model well estimates the latency for a batch size of one; hence, we can assume this value is realistic. When the computational pipeline fills up due to bigger batches, our architecture time-per-frame should converge to 8.26 ms, reaching a steady-state throughput of 3926.24 GOPS. Even in this case, we measured the throughput results as the total number of performed operations over the estimated execution time. However, the experimental results on AlexNet in Section 7.3 indicate the performance model deviates from real latency at a steady-state for about a 7%. Therefore, to conduct a fair comparison with the works in the literature, we added a conservative correction of 7% to the estimated steady-state latency, obtaining 3669.39 GOPS.

Table 7 reports the performance of various state-of-the-art approaches discussed in Section 3.2 along with our estimation. We do not include power and power efficiency values, nor we estimate them since the design does not run effectively. Most of the works employ 16-bit quantization, while others (including ours) 8-bit. As described in Section 4.2.2, 8-bit quantization is generally a clear bound to ensure negligible accuracy loss for both AlexNet and VGG. In general, the performance

[4]It refers to the overall network, and not just the convolutional part.
[5]It refers to the single convolutional stages, each evaluated separately, and not the entire network.
[6]It includes an additional 7% correction.

in Table 7 regards CNVLs only. A footnote indicates when an entry applies to the overall network. As noticed for AlexNet, approaches like [2, 41, 90] confirm their high throughput, particularly the ones exploiting Winograd transformation [2, 41]. For instance, the convolution engine proposed by Ahmad et al. [2] reaches a remarkable throughput, but such an outcome depends on the separate evaluation of the accelerator on groups of VGG16 layers. Nonetheless, we can see that our work surpasses the state-of-the-art works in Table 7 in terms of throughput. This result significantly depends on the scalability of the multi-FPGA system and the large number of required DSPs, as it would not be feasible to use such an amount of DSPs on a single FPGA. Finally, even though our throughput is only an estimation, it shows that efficient and scalable multi-FPGA can help achieve meaningful performance when applied to heavy pipelined computations like ISLs and CNNs.

8 CONCLUSIONS

Stencil-based computations represent a common computational kernel involved in a wide range of applications, such as ISLs and CNNs. This work presents a set of HDL IPs, build upon the SST micro-architecture [51], to compute ISLs and CNNs on FPGA, exploiting fine-grained optimizations aimed to improve performance, while guaranteeing scalability and flexibility. This work also proposes a distributed hardware architecture, composed of a chain of FPGAs that communicate through custom interconnection PCBs exploiting an intra-node bandwidth of 7.75 GB/s. The multi-FPGA system allows to scale further the performance of both ISLs and CNNs.

The experimental evaluation of ISLs shows that our solution is competitive with the optimized CPU version [6] when running a single FPGA, reaching an average speed-up of 3.43×, while largely outperforms both software [6] and multi-FPGA [61] approaches when scaling to multiple boards, by an average factor of 12.54× and 1.53×, respectively. Regarding CNNs, this work proposes a multi-FPGA implementation of the AlexNet network, which exploits several parallelism degrees and specific architectural optimizations. The obtained results show an overall time-to-image of 2.14 ms, and throughput of 1106.54 GOPS, in line with the literature. Finally, we estimated the performance we could achieve by designing VGG16 on the proposed multi-FPGA system. The estimated throughput reaches 3669.39 GOPS, theoretically outperforming literature works.

REFERENCES

[1] A. Ahmad and M. A. Pasha. 2019. Towards Design Space Exploration and Optimization of Fast Algorithms for Convolutional Neural Networks (CNNs) on FPGAs. In *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*. 1106–1111.

[2] Afzal Ahmad and Muhammad Adeel Pasha. 2020. FFConv: An FPGA-Based Accelerator for Fast Convolution Layers in Convolutional Neural Networks. *ACM Trans. Embed. Comput. Syst.* 19, 2, Article 15 (March 2020), 24 pages. https://doi.org/10.1145/3380548

[3] Francesc Aràndiga, Albert Cohen, Rosa Donat, and Basarab Matei. 2010. Edge detection insensitive to changes of illumination in the image. *Image and Vision Computing* 28, 4 (2010), 553–562.

[4] Utku Aydonat, Shane O'Connell, Davor Capalija, Andrew C. Ling, and Gordon R. Chiu. 2017. An OpenCL™Deep Learning Accelerator on Arria 10. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Monterey, California, USA) *(FPGA '17)*. ACM, New York, NY, USA, 55–64. https://doi.org/10.1145/3020078.3021738

[5] M. Bacis, G. Natale, E. Del Sozzo, and M. D. Santambrogio. 2017. A pipelined and scalable dataflow implementation of convolutional neural networks on FPGA. In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 90–97. https://doi.org/10.1109/IPDPSW.2017.44

[6] Vinayaka Bandishti, Irshad Pananilath, and Uday Bondhugula. 2012. Tiling Stencil Computations to Maximize Parallelism. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (Salt Lake City, Utah) *(SC '12)*. IEEE Computer Society Press, Los Alamitos, CA, USA, Article 40, 11 pages. http://dl.acm.org/citation.cfm?id=2388996.2389051

[7] Uday Bondhugula. 2008. PLUTO - An automatic parallelizer and locality optimizer for affine loop nests. http://pluto-compiler.sourceforge.net/.

[8] Uday Bondhugula. 2008. PLUTO Compiler Repository - Examples. https://github.com/bondhugula/pluto/tree/master/examples.

[9] Uday Bondhugula, Muthu Baskaran, Sriram Krishnamoorthy, J. Ramanujam, Atanas Rountev, and P. Sadayappan. 2008. Automatic Transformations for Communication-Minimized Parallelization and Locality Optimization in the Polyhedral Model. In *Proceedings of the Joint European Conferences on Theory and Practice of Software 17th International Conference on Compiler Construction* (Budapest, Hungary) *(CC'08/ETAPS'08)*. Springer-Verlag, Berlin, Heidelberg, 132–146.

[10] Uday Bondhugula, Albert Hartono, J. Ramanujam, and P. Sadayappan. 2008. A Practical Automatic Polyhedral Parallelizer and Locality Optimizer. In *Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Tucson, AZ, USA) *(PLDI '08)*. Association for Computing Machinery, New York, NY, USA, 101–113. https://doi.org/10.1145/1375581.1375595

[11] Bing-Yang Cao and Ruo-Yu Dong. 2012. Nonequilibrium molecular dynamics simulation of shear viscosity by a uniform momentum source-and-sink scheme. *J. Comput. Phys.* 231, 16 (2012), 5306–5316.

[12] Riccardo Cattaneo, Giuseppe Natale, Carlo Sicignano, Donatella Sciuto, and Marco Domenico Santambrogio. 2016. On how to accelerate iterative stencil loops: a scalable streaming-based approach. *ACM Transactions on Architecture and Code Optimization (TACO)* 12, 4 (2016), 53.

[13] Adrian Caulfield, Eric Chung, Andrew Putnam, Hari Angepat, Jeremy Fowers, Michael Haselman, Stephen Heil, Matt Humphrey, Puneet Kaur, Joo-Young Kim, Daniel Lo, Todd Massengill, Kalin Ovtcharov, Michael Papamichael, Lisa Woods, Sitaram Lanka, Derek Chiou, and Doug Burger. 2016. A Cloud-Scale Acceleration Architecture. https://www.microsoft.com/en-us/research/publication/configurable-cloud-acceleration/

[14] Yuze Chi, Jason Cong, Peng Wei, and Peipei Zhou. 2018. SODA: stencil with optimized dataflow architecture. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–8.

[15] M. Christen, O. Schenk, and H. Burkhart. 2011. PATUS: A Code Generation and Autotuning Framework for Parallel Iterative Stencil Computations on Modern Microarchitectures. In *2011 IEEE International Parallel Distributed Processing Symposium*. 676–687. https://doi.org/10.1109/IPDPS.2011.70

[16] J. Cong, P. Li, B. Xiao, and P. Zhang. 2014. An Optimal Microarchitecture for Stencil Computation Acceleration Based on Non-Uniform Partitioning of Data Reuse Buffer. In *Proceedings of the 51st Annual Design Automation Conference* (San Francisco, CA, USA) *(DAC '14)*. ACM, New York, NY, USA, Article 77, 6 pages. https://doi.org/10.1145/2593069.2593090

[17] J. Cong, P. Li, B. Xiao, and P. Zhang. 2016. An Optimal Microarchitecture for Stencil Computation Acceleration Based on Nonuniform Partitioning of Data Reuse Buffers. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35, 3 (2016), 407–418.

[18] Hang Ding and Chang Shu. 2006. A stencil adaptive algorithm for finite difference solution of incompressible viscous flows. *J. Comput. Phys.* 214, 1 (2006), 397–420.

[19] Clément Farabet, Cyril Poulet, Jefferson Y. Han, and Yann Lecun. 2009. Cnp: An fpga-based processor for convolutional networks. In *in International Conference on Field Programmable Logic and Applications*.

[20] Paul Feautrier and Christian Lengauer. 2011. Polyhedron Model. 1581–1592. https://doi.org/10.1007/978-0-387-09766-4_502

[21] Matteo Frigo and Volker Strumpen. 2007. The memory behavior of cache oblivious stencil computations. *The Journal of Supercomputing* 39, 2 (2007), 93–112.

[22] V. Gokhale, A. Zaidy, A. X. M. Chang, and E. Culurciello. 2017. Snowflake: An efficient hardware accelerator for convolutional neural networks. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. 1–4.

[23] S. Grauer-Gray, L. Xu, R. Searles, S. Ayalasomayajula, and J. Cavazos. 2012. Auto-tuning a high-level language targeted to GPU codes. In *2012 Innovative Parallel Computing (InPar)*. 1–10. https://doi.org/10.1109/InPar.2012.6339595

[24] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang. 2018. Angel-Eye: A Complete Design Flow for Mapping CNN Onto Embedded FPGA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 1 (2018), 35–47.

[25] Kaiyuan Guo, Shulin Zeng, Jincheng Yu, Yu Wang, and Huazhong Yang. 2019. [DL] A survey of FPGA-based neural network inference accelerators. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 12, 1 (2019), 1–26.

[26] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[27] Robert M Haralick and Linda G Shapiro. 1992. *Computer and robot vision*. Vol. 1. Addison-wesley Reading.

[28] Justin Holewinski, Louis-Noël Pouchet, and P. Sadayappan. 2012. High-performance Code Generation for Stencil Computations on GPU Architectures. In *Proceedings of the 26th ACM International Conference on Supercomputing* (San Servolo Island, Venice, Italy) *(ICS '12)*. ACM, New York, NY, USA, 311–320. https://doi.org/10.1145/2304576.2304619

[29] Amazon Inc. 2018. EC2 F1 Instances. https://aws.amazon.com/it/ec2/instance-types/f1/. Accessed: 2018-02-12.

[30] Microsoft Inc. 2018. Project Brainwave. https://www.microsoft.com/en-us/research/blog/microsoft-unveils-project-brainwave/. Accessed: 2018-03-07.

[31] Xilinx Inc. 2018. Aurora 64B/66B link-layer protocol.g. https://www.xilinx.com/support/documentation/ip_documentation/aurora_64b66b_protocol_spec_sp011.pdf. Accessed: 2018-02-12.

[32] Kazufumi Ito and Jari Toivanen. 2009. Lagrange Multiplier Approach with Optimized Finite Difference Stencils for Pricing American Options under Stochastic Volatility. *SIAM J. Scientific Computing* 31 (2009), 2646–2664.

[33] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional Architecture for Fast Feature Embedding. In *Proceedings of the 22Nd ACM International Conference on Multimedia* (Orlando, Florida, USA) *(MM '14)*. ACM, New York, NY, USA, 675–678. https://doi.org/10.1145/2647868.2654889

[34] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, and Gaurav Agrawal Et al. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. https://arxiv.org/pdf/1704.04760.pdf

[35] Tomoyoshi Kobori and Tsutomu Maruyama. 2003. A high speed computation system for 3D FCHC lattice gas model with FPGA. In *International Conference on Field Programmable Logic and Applications*. Springer, 755–765.

[36] Raghuraman Krishnamoorthi. 2018. Quantizing deep convolutional networks for efficient inference: A whitepaper. *CoRR* abs/1806.08342 (2018).

[37] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 1097–1105. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[38] Andrew Lavin. 2015. Fast Algorithms for Convolutional Neural Networks. *CoRR* abs/1509.09308 (2015). arXiv:1509.09308 http://arxiv.org/abs/1509.09308

[39] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (Nov 1998), 2278–2324. https://doi.org/10.1109/5.726791

[40] Huimin Li, Xitian Fan, Li Jiao, Wei Cao, Xuegong Zhou, and Lingli Wang. 2016. 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, 1–9.

[41] Y. Liang, L. Lu, Q. Xiao, and S. Yan. 2020. Evaluating Fast Algorithms for Convolutional Neural Networks on FPGAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 4 (2020), 857–870. https://doi.org/10.1109/TCAD.2019.2897701

[42] Zhiqiang Liu, Yong Dou, Jingfei Jiang, and Jinwei Xu. 2016. Automatic code generation of convolutional neural networks in FPGA implementation. In *2016 International Conference on Field-Programmable Technology (FPT)*. IEEE, 61–68.

[43] Yufei Ma, Yu Cao, Sarma Vrudhula, and Jae-sun Seo. 2017. Optimizing Loop Operation and Dataflow in FPGA Acceleration of Deep Convolutional Neural Networks. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Monterey, California, USA) *(FPGA '17)*. Association for Computing Machinery, New York, NY, USA, 45–54. https://doi.org/10.1145/3020078.3021736

[44] Yufei Ma, Naveen Suda, Yu Cao, Sarma Vrudhula, and Jae sun Seo. 2018. ALAMO: FPGA acceleration of deep learning algorithms with a modularized RTL compiler. *Integration* 62 (2018), 14 – 23. https://doi.org/10.1016/j.vlsi.2017.12.009

[45] A Theodore Markettos, Paul J Fox, Simon W Moore, and Andrew W Moore. 2014. Interconnect for commodity FPGA clusters: Standardized or customized?. In *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 1–8.

[46] John Marshall, Alistair Adcroft, Chris Hill, Lev Perelman, and Curt Heisey. 1997. A finite-volume, incompressible Navier Stokes model for studies of the ocean on parallel computers. *Journal of Geophysical Research: Oceans* 102, C3 (1997), 5753–5766.

[47] Jiayuan Meng and Kevin Skadron. 2009. Performance Modeling and Automatic Ghost Zone Optimization for Iterative Stencil Loops on GPUs. In *Proceedings of the 23rd International Conference on Supercomputing* (Yorktown Heights, NY, USA) *(ICS '09)*. ACM, New York, NY, USA, 256–265. https://doi.org/10.1145/1542275.1542313

[48] A. Mondigo, K. Sano, and H. Takizawa. 2018. Performance Estimation of Deeply Pipelined Fluid Simulation on Multiple FPGAs with High-speed Communication Subsystem. In *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. 1–4.

[49] A. A. Nacci, V. Rana, F. Bruschi, D. Sciuto, P. di Milano, I. Beretta, and D. Atienza. 2013. A high-level synthesis flow for the implementation of iterative stencil loop algorithms on FPGA devices. In *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1145/2463209.2488797

[50] Aiichiro Nakano, Rajiv K Kalia, and Priya Vashishta. 1994. Multiresolution molecular dynamics algorithm for realistic materials modeling on parallel computers. *Computer Physics Communications* 83, 2-3 (1994), 197–214.

[51] Giuseppe Natale, Giulio Stramondo, Pietro Bressana, Riccardo Cattaneo, Donatella Sciuto, and Marco D Santambrogio. 2016. A polyhedral model-based framework for dataflow implementation on FPGA devices of iterative stencil loops. In *Proceedings of the 35th International Conference on Computer-Aided Design*. ACM, 77.

[52] NVIDIA. 2018. TensorRT. https://developer.nvidia.com/tensorrt. Accessed: 2018-09-07.

[53] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. (2017).

[54] A. Podili, C. Zhang, and V. Prasanna. 2017. Fast and efficient implementation of Convolutional Neural Networks on FPGA. In *2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. 11–18.

[55] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, Yu Wang, and Huazhong Yang. 2016. Going Deeper with Embedded FPGA Platform for Convolutional Neural Network. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Monterey, California, USA) *(FPGA '16)*. ACM, New York, NY, USA, 26–35. https://doi.org/10.1145/2847263.2847265

[56] N. Raspa, G. Natale, M. Bacis, and M. D. Santambrogio. 2018. A Framework with Cloud Integration for CNN Acceleration on FPGA Devices. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 170–177. https://doi.org/10.1109/IPDPSW.2018.00033

[57] Enrico Reggiani, Giuseppe Natale, Carlo Moroni, and Marco D Santambrogio. 2018. An FPGA-Based Acceleration Methodology and Performance Model for Iterative Stencils. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 115–122.

[58] Enrico Reggiani, Marco Rabozzi, Anna Maria Nestorov, Alberto Scolari, Luca Stornaiuolo, and Marco Santambrogio. 2019. Pareto optimal design space exploration for accelerated CNN on FPGA. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 107–114.

[59] Franz Richter, Michael Schmidt, and Dietmar Fey. 2012. A Configurable VHDL Template for Parallelization of 3 D Stencil Codes on FPGAs.

[60] M. Sankaradas, V. Jakkula, S. Cadambi, S. Chakradhar, I. Durdanovic, E. Cosatto, and H. P. Graf. 2009. A Massively Parallel Coprocessor for Convolutional Neural Networks. In *2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors*. 53–60. https://doi.org/10.1109/ASAP.2009.25

[61] K. Sano, Y. Hatsuda, and S. Yamamoto. 2014. Multi-FPGA Accelerator for Scalable Stencil Computation with Constant Memory Bandwidth. *IEEE Transactions on Parallel and Distributed Systems* 25, 3 (March 2014), 695–705. https://doi.org/10.1109/TPDS.2013.51

[62] K. Sano and S. Yamamoto. 2017. FPGA-Based Scalable and Power-Efficient Fluid Simulation using Floating-Point DSP Blocks. *IEEE Transactions on Parallel and Distributed Systems* 28, 10 (2017), 2823–2837.

[63] L Shapiro and G Stockman. 2001. Computer vision prentice hall. *Inc., New Jersey* (2001).

[64] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Mishra, and H. Esmaeilzadeh. 2016. From high-level deep neural models to FPGAs. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–12.

[65] Junzhong Shen, You Huang, Zelong Wang, Yuran Qiao, Mei Wen, and Chunyuan Zhang. 2018. Towards a Uniform Template-Based Architecture for Accelerating 2D and 3D CNNs on FPGA. In *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Monterey, CALIFORNIA, USA) *(FPGA '18)*. Association for Computing Machinery, New York, NY, USA, 97–106. https://doi.org/10.1145/3174243.3174257

[66] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. (09 2014).

[67] Gerard Sleijpen and Henk Van der Vorst. 2000. A Jacobi-Davidson Iteration Method for Linear Eigenvalue Problems. 17 (12 2000).

[68] John A. Stratton, Christopher Rodrigues, I-Jui Sung, Nady Obeid, Li-Wen Chang, Nasser Anssari, Geng Liu, and Wen mei W. Hwu. 2012. Parboil : A Revised Benchmark Suite for Scientific and Commercial Throughput Computing.

[69] D. Strigl, K. Kofler, and S. Podlipnig. 2010. Performance and Scalability of GPU-Based Convolutional Neural Networks. In *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*. 317–324. https://doi.org/10.1109/PDP.2010.43

[70] Naveen Suda, Vikas Chandra, Ganesh Dasika, Abinash Mohanty, Yufei Ma, Sarma Vrudhula, Jae-sun Seo, and Yu Cao. 2016. Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 16–25.

[71] Yuan Tang, Rezaul Alam Chowdhury, Bradley C. Kuszmaul, Chi-Keung Luk, and Charles E. Leiserson. 2011. The Pochoir Stencil Compiler. In *Proceedings of the Twenty-third Annual ACM Symposium on Parallelism in Algorithms and Architectures* (San Jose, California, USA) *(SPAA '11)*. ACM, New York, NY, USA, 117–128. https://doi.org/10.1145/1989493.1989508

[72] Maxeler Technologies. 2015. MPC-X Series. https://www.maxeler.com/products/mpc-xseries/.

[73] T. Tian, X. Jin, L. Zhao, X. Wang, J. Wang, and W. Wu. 2020. Exploration of Memory Access Optimization for FPGA-based 3D CNN Accelerator. In *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*. 1650–1655.

[74] S. I. Venieris and C. Bouganis. 2017. Latency-driven design for FPGA-based convolutional neural networks. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. 1–8.

[75] Stylianos I Venieris, Alexandros Kouris, and Christos-Savvas Bouganis. 2018. Toolflows for Mapping Convolutional Neural Networks on FPGAs: A Survey and Future Directions. *ACM Computing Surveys (CSUR)* 51, 3 (2018), 1–39.

[76] Hasitha Muthumala Waidyasooriya and Masanori Hariyama. 2019. Multi-FPGA Accelerator Architecture for Stencil Computation Exploiting Spacial and Temporal Scalability. *IEEE Access* 7 (2019), 53188–53201.

[77] Hasitha Muthumala Waidyasooriya, Yasuhiro Takei, Shunsuke Tatsumi, and Masanori Hariyama. 2016. OpenCL-based FPGA-platform for stencil computation and its optimization methodology. *IEEE Transactions on Parallel and Distributed Systems* 28, 5 (2016), 1390–1402.

[78] S. Wang and Y. Liang. 2017. A comprehensive framework for synthesizing stencil algorithms on FPGAs using OpenCL model. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1145/3061639.3062185

[79] Y. Wang, J. Xu, Y. Han, H. Li, and X. Li. 2016. DeepBurning: Automatic generation of FPGA-based learning accelerators for the Neural Network family. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6.

[80] Xuechao Wei, Cody Hao Yu, Peng Zhang, Youxiang Chen, Yuxin Wang, Han Hu, Yun Liang, and J. Cong. 2017. Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1145/3061639.3062207

[81] Stephen Wolfram. 1984. Computation theory of cellular automata. *Comm. Math. Phys.* 96, 1 (1984), 15–57. https://projecteuclid.org:443/euclid.cmp/1103941718

[82] Omry Yadan, Keith Adams, Yaniv Taigman, and Facebook Ai Group. [n.d.]. Multi-GPU training of convnets. *CoRR* ([n. d.]), 2013.

[83] Kai Yu. 2013. Large-scale Deep Learning at Baidu. In *Proceedings of the 22Nd ACM International Conference on Information & Knowledge Management* (San Francisco, California, USA) *(CIKM '13)*. ACM, New York, NY, USA, 2211–2212. https://doi.org/10.1145/2505515.2514699

[84] Hanqing Zeng, Ren Chen, Chi Zhang, and Viktor Prasanna. 2018. A Framework for Generating High Throughput CNN Implementations on FPGAs. In *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Monterey, CALIFORNIA, USA) *(FPGA '18)*. Association for Computing Machinery, New York, NY, USA, 117–126. https://doi.org/10.1145/3174243.3174265

[85] C. Zhang, Zhenman Fang, Peipei Zhou, Peichen Pan, and Jason Cong. 2016. Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks. In *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–8. https://doi.org/10.1145/2966986.2967011

[86] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. 2015. Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Monterey, California, USA) *(FPGA '15)*. ACM, New York, NY, USA, 161–170. https://doi.org/10.1145/2684746.2689060

[87] Chi Zhang and Viktor Prasanna. 2017. Frequency domain acceleration of convolutional neural networks on CPU-FPGA shared memory system. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 35–44.

[88] Chen Zhang, Di Wu, Jiayu Sun, Guangyu Sun, Guojie Luo, and Jason Cong. 2016. Energy-Efficient CNN Implementation on a Deeply Pipelined FPGA Cluster. In *Proceedings of the 2016 International Symposium on Low Power Electronics and Design* (San Francisco Airport, CA, USA) *(ISLPED '16)*. ACM, New York, NY, USA, 326–331. https://doi.org/10.1145/2934583.2934644

[89] Jialiang Zhang and Jing Li. 2017. Improving the Performance of OpenCL-based FPGA Accelerator for Convolutional Neural Network. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Monterey, California, USA) *(FPGA '17)*. ACM, New York, NY, USA, 25–34. https://doi.org/10.1145/3020078.3021698

[90] Xiaofan Zhang, Junsong Wang, Chao Zhu, Yonghua Lin, Jinjun Xiong, Wen-mei Hwu, and Deming Chen. 2018. 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, 1–8.