TAPA-CS: Enabling Scalable Accelerator Design on Distributed HBM-FPGAs

Neha Prakriya, Yuze Chi, Suhail Basalama, Linghao Song, Jason Cong

University of California, Los Angeles

nehaprakriya, chiyuze, basalama, linghaosong, cong@cs.ucla.edu

Abstract

Despite the increasing adoption of Field-Programmable Gate Arrays (FPGAs) in compute clouds, there remains a significant gap in programming tools and abstractions which can leverage network-connected, cloud-scale, multi-die FPGAs to generate accelerators with high frequency and throughput. To this end, we propose TAPA-CS, a task-parallel dataflow programming framework which automatically partitions and compiles a large design across a cluster of FPGAs with no additional user effort while achieving high frequency and throughput. TAPA-CS has three main contributions. First, it is an open-source framework which allows users to leverage virtually "unlimited" accelerator fabric, high-bandwidth memory (HBM), and on-chip memory, by abstracting away the underlying hardware. This reduces the user's programming burden to a logical one, enabling software developers and researchers with limited FPGA domain knowledge to deploy larger designs than possible earlier. Second, given as input a large design, TAPA-CS automatically partitions the design to map to multiple FPGAs, while ensuring congestion control, resource balancing, and overlapping of communication and computation. Third, TAPA-CS couples coarse-grained floorplanning with automated interconnect pipelining at the inter- and intra-FPGA levels to ensure high frequency. We have tested TAPA-CS on our multi-FPGA testbed where the FPGAs communicate through a high-speed 100GBps Ethernet infrastructure. We have evaluated the performance and scalability of our tool on designs, including systolic-array based convolutional neural networks (CNNs), graph processing workloads such as page rank, stencil applications like the Dilate kernel, and K-nearest neighbors (KNN). TAPA-CS has the potential to accelerate development of increasingly complex and large designs on the low power and reconfigurable FPGAs. In fact, 64% of our evaluated designs fail placement or routing on a single FPGA through traditional CAD tools but are successfully routed by TAPA-CS on 2-4 FPGAs with an average design frequency of 300MHz. We have evaluated TAPA-CS for multiple test input configurations, achieving an average throughput improvement of 1.45x and a frequency improvement between 18-116% compared with single FPGA designs routed by traditional CAD toolflows.

1. Introduction

In the big data era, there has been an exponential rise in the demand for scalable, cheap, and high performance acceleration. FPGAs have emerged as a promising solution to counter
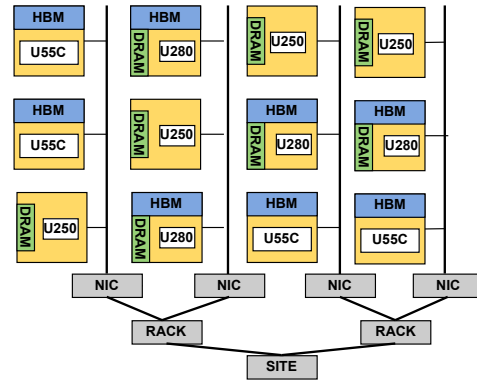

Figure 1: Network-Connected FPGAs

the breakdown of Dennard's scaling [31, 20] due to their reconfigurability and low power consumption. One of the greatest successful demonstrations is Microsoft's Catapult project which sped up the Bing Search engine using Stratix V FPGAs, achieving a 95% increase in throughput with a minimal power consumption increase of only 10% [49]. Microsoft also displayed the use of FPGAs for accelerating DNN inference and data compression in their servers [46, 33, 23, 34, 28]. Today, other major players such as Amazon [5, 4], Alibaba [1], Baidu [7], and Huawei [9] also use FPGAs to accelerate their workloads, and offer them as a service in their cloud. Most of these are achieved through manual RTL coding.

High-level synthesis (HLS) tools like Vitis HLS [13] and Intel HLS [10] raise the abstraction level for programming individual FPGAs in the cloud from RTL to C++/OpenCL. This allows the programmer to have little to no knowledge of the underlying hardware and cycle accuracy. While these tools deliver great results, they are limited to programming a single FPGA. At the same time, with the gradual end of Moore's law, accelerator designs are becoming larger than ever before, and require more programmable logic and memory than that available on a single FPGA device [45]. Our goal is to support the network-connected devices shown in Figure 1. Utilizing such multi-FPGA setups requires careful consideration for high-speed communication support and efficient workload distribution to ammortize the cost of inter-FPGA communication.

The design of modern FPGAs also adds a new layer of complexity. Modern FPGA architectures have varying interconnection support (PCIe and Ethernet-based QSFP28 ports), different types of memory bandwidth and capacity (on-chip BRAM, off-chip DRAM, and HBM), programmable logic

Method	HLS	Ethernet	Floorplanning	Interconnect Pipelining	Topology-Aware	Automatic Partitioning	Hardware Execution	Generalizable	Fmax (MHz)
FPGA'12[32]	×	×	×	×	×	×	×	✓	85
Simulation-based [40, 42, 53]	×	×	×	×	×	✓	×	✓	-
Virtualization-based [60, 61, 62, 26]	✓	✓	×	×	×	✓	✓	✓	100-300
CNN/DNN [15, 63, 17, 18, 64, 55, 39]	✓	✓	×	×	×	✓	✓	×	240
TAPA-CS (Ours)	✓	✓	✓	✓	✓	✓	✓	✓	300

Table 1: Comparison of TAPA-CS and existing methods providing scale-out acceleration across multiple FPGAs.

units organized into multiple dies (i.e., chiplets), and degrees of data transfer cost (on-die, cross-die, cross-chip). Consider the example of the Xilinx/AMD Alveo U55C cards. This card supports 2 Ethernet-compatible QSFP28 ports for networking, offering 100GBps bandwidth per port. It also features an HBM with 16GB capacity exposing a bandwidth of 460GBps. The on-chip memory provides a high bandwidth of 35TBps but has a small capacity of 43MB [2]. Furthermore, these FPGAs are divided into multiple dies joined by silicon interposers, with a high inter-die crossing delay. We introduce details of modern FPGA architecture in Section 2.

An expert designer will consider all these factors when designing and partitioning their kernel code across chips. However, manual workload partitioning is inefficient as the design gets larger, and, in-turn, raises the barrier for using FPGAs. Therefore, despite the advances in CAD tools which allow the user to program a single *FPGA in the cloud*, there is a significant lack of programming tools which can target *multiple FPGAs* potentially at the cloud scale. Such a framework should take as input large workloads from the user and automatically partition it across multiple devices efficiently. We identify the following three main challenges to address when developing such frameworks for cloud-scale FPGAs:

1. Need a lightweight inter-FPGA communication infrastructure which enables reliable and high speed data transfers.
2. Need to partition and map application code efficiently keeping in mind factors like compute-load balancing, network topology, the varying cost of on- and off-chip communication, and keeping resource utilization in each die under a specified threshold.
3. Need to ensure high design frequency by hiding inter-FPGA communication latency, and sufficiently pipelining the interconnect.

Several prior works have attempted to leverage the networking capabilities available in modern FPGAs [51, 54, 30, 37, 50, 38, 3]. These prior works differ in the achievable data transfer throughput (10-90GBps), orchestration of data transfers (host/FPGA), and the resource overheads. We compare these methods in detail in Section 6.

There are some initial efforts addressing Challenges 2 and 3 which we compare in Table 1. Simulation-based tools [40, 42, 53] enable rapid prototyping but are not substitutes for real hardware execution. Prior work such as [15, 63, 17, 18, 64, 55, 39] proposed CNN/DNN partitioning across FPGAs, but are not generalizable to different workloads. Other works such as [32] leverage latency-insensitivity (discussed in Section 4.3) to partition the design across FPGAs but expect the user to provide module-to-FPGA mappings in RTL,

and perform simulation-based experiments. If HLS front-ends are combined with automatically partitioned designs, users from varied backgrounds are more likely to adopt FPGAs. Recent virtualization-based work [60, 61, 62] also leverages latency-insensitive design to partition the workload, but virtualizes the FPGA by creating pre-placed and pre-routed static regions to which user logic is mapped. This increases the area overheads and degrades the customizable nature of FPGAs. Most prior works take advantage of the high-bandwidth networking capabilities available in modern FPGAs in the form of Ethernet-compatible ports, but do not formulate their design partitioners in a way that minimizes and hides this latency. We find that none of these prior works consider coupling intelligent floorplanning of the compute modules and interconnect pipelining with HLS compilation. This step is crucial in achieving designs with high frequency as we discuss in Section 2. Also, none of the prior works consider the topology of the networked-FPGA infrastructure. This might result in the suboptimal mapping of compute modules to devices, and is a major hurdle in the scalability of the tool beyond two FPGAs. We discuss more details of prior works in Section 6.

To this end, we propose TAPA-CS which takes as input any large-scale dataflow workload expressed in C/C++, and automatically partitions and maps it to a cluster of modern FPGAs during HLS compilation. TAPA-CS couples the process of intra- and inter-FPGA floorplanning with interconnect pipelining to ensure high frequency. TAPA-CS is built upon the latest progress in dataflow-based FPGA HLS design tools [24, 35, 36]. Our main contributions are as follows:

1. Integrate two layers of floorplanning (inter- and intra-FPGA) and interconnect pipelining with HLS compilation using an Integer Linear Programming (ILP)-based resource allocator which takes into account network topology, internal FPGA chip layout, and the resource requirements of the input workload. This ensures generalizability to different workloads and network topologies, low resource congestion on-board, and high frequency designs.
2. Utilize the latency-insensitive nature of the dataflow design to partition it across devices, allowing us flexibility in implementing the inter-FPGA communication infrastructure.
3. Raise the abstraction level of programming cloud-scale FPGAs by hiding the cluster complexity and virtualizing multiple devices as one from the user perspective.
4. We test TAPA-CS on different applications ranging from systolic-array based CNNs, stencil designs, Page Rank, and KNN. Out of the tested designs, 64% fail placement or routing stages of traditional CAD tools, but can be successfully routed with TAPA-CS achieving an average frequency of
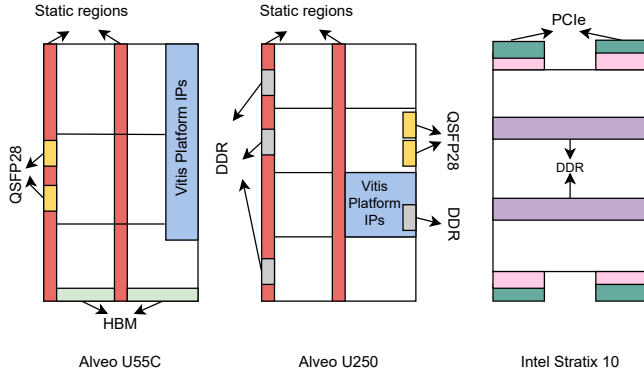
Figure 2: Architecture examples of modern FPGAs.

Figure 3: (A) Typical FPGA compilation flow, (B) Additions by TAPA-CS are highlighted in blue.

300 MHz. Across the tested designs we achieve an average throughput increase of 1.45x and frequency increase between 18-116% compared with traditional CAD tools which route the design on a single FPGA.

2. Background

FPGAs consist of programmable logic organized in the form of a 2-D grid. This programmable region consists of look-up tables (LUTs) which can implement the truth table for any 6-input function. In recent years, FPGAs have also come to include several hard IPs such as PCIe IPs, DDR/HBM controllers, and other platform-specific IPs between the programmable logic regions. While these IPs improve design performance, they have a fixed location on-board and consume a significant amount of logic around them. AMD/Xilinx UltraScale+ FPGAs [6] are also organized into multiple dies separated by silicon interposers. Crossing these die boundaries results in a much higher delay than on-die interconnect. Similar trends are also observed in case of Intel FPGAs [11]. Both AMD and Intel FPGA boards expose physical interfaces in the form of PCIe ports and networking interfaces in the form of Ethernet-compatible QSFP28 ports. Figure 2 describes the chip layout of the Alveo U55C, U250 cards and the Intel Stratix 10 cards.

Accelerators for such FPGAs can be designed using commercial tools like Vitis [13] and Intel HLS [10]. Figure 3 (A) depicts the key steps involved in such toolflows. First, the untimed C/C++ input is converted into a timed RTL using HLS. Next, the timed RTL is passed for synthesis, placement and routing where the logic is mapped to the physical hardware. While these tools simplify the design process, they often suffer from poor quality of results compared with an expert-tuned RTL. The key reason behind this is that HLS tools cannot correctly estimate the final placement of compute modules on the board, and insert insufficient number of clock boundaries (registers) between them while converting the untimed input into a timed output. Due to this, several connections remain underpipelined, degrading the final frequency. Therefore, it is important to provide the tool a global view of the chip lay-
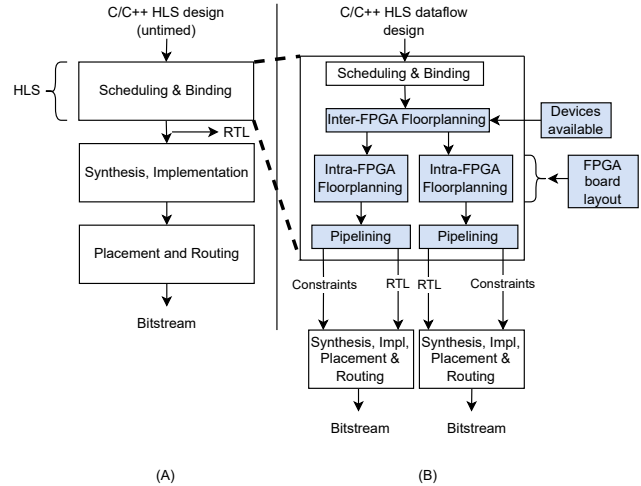
out and the placement of the compute modules *during* HLS compilation.

Prior work such as Autobridge [36] integrates a coarse-grained floorplanning step with interconnect pipelining in HLS compilation for optimization on a single FPGA device. Other approaches such as [65, 59] also attempt to provide HLS with a layout of the device, but either suffer from high runtime or evaluate their method only on small applications. TAPA [25] extends HLS to feature fast compilation, and an expressive programming model for task-parallel programs. It exposes user-friendly APIs which decouple communication and computation, allowing the user great flexibility in implementing the inter-module communication patterns.

3. Motivating Example of TAPA-CS

We demonstrate the importance of using TAPA-CS through the KNN application. Through this example, we aim to dispel the common misconception that scale-out acceleration is only useful when the design cannot be routed on a single FPGA. We find that even when designs can be successfully routed on a single device, span-out acceleration across multiple devices allows the design to efficiently utilize on-chip memory and HBM.

We use the KNN algorithm presented in [44]. There are two major phases in this algorithm. The first phase calculates the distance of an input query data point with every other datapoint in the dataset. Given that the dataset contains N data points, each represented as a D-dimensional feature vector, this phase has a computational and memory access complexity of $O(N * D)$ for a single query. Since the KNN application is commonly used on very large datasets with large features, the computational and memory access cost quickly scales up. The second phase sorts the N distances calculated in Phase 1 and returns the top K nearest neighbors. Since K is usually small and we only need to sort for the K smallest distances,

the complexity of this phase is $O(N * K)$. The topology of the application is as illustrated in Figure 4 (A).

The design generated by traditional CAD tools results in a low design frequency of 165MHz and high latency. There are three main reasons for this. First, this design utilizes a port width and buffer size of 256 bits and 32KB which only saturates about 51.2% of the per-bank HBM bandwidth. Prior work has also found that in case of memory-bound applications when multiple processing elements (PEs) access the HBM channels, the achievable HBM bandwidth can drop to as low as 9.4GBps [27]. We find that the optimal port width and buffer size which allows us to saturate the per-bank bandwidth is 512 bits and 128KB respectively. This configuration however, results in very high resource utilization in the lower die, leading to a failure in the routing phase of traditional CAD tools. Second, since a smaller buffer size is used, there is a higher number of HBM accesses. Given that HBM accesses are about 76x slower than on-chip memory access, an efficient design should use on-chip memory as much as possible. Third, this design does not feature any intelligent floorplanning or interconnect pipelining. Even with intelligent floorplanning and interconnect pipelining on a single FPGA, the design frequency increases to 198MHz and still incurs a high latency since the HBM bandwidth cannot be saturated.

TAPA-CS includes all these considerations to generate an optimized KNN implementation automatically partitioned across two FPGAs as shown in Figure 4 (B). This provides sufficient resources in the lower die to route the optimal port width and buffer size configuration. Also, since the input data is divided between the two FPGAs, the compute load is balanced and neither FPGA is idle. The designs generated by TAPA-CS result in a design frequency of 300MHz and are 2.0x faster than designs on a single FPGA. Therefore, it is often a misconception that a multi-FPGA design is worse than a single FPGA design if it could be routed successfully on a single FPGA. Using multiple FPGAs can expose higher HBM bandwidth per compute module (aiding the performance of memory-bound applications), and enable the successful routing of larger designs (aiding the design of compute-bound applications). We discuss more details of this design in Section 5.

4. TAPA-CS Design

4.1. Problem Formulation

TAPA-CS takes as input a C/C++ dataflow program currently written in the TAPA-format [25] in which each function compiles into an RTL module and communicates with other functions using FIFOs. We also take the network topology and number of FPGAs present in the user cluster as input.

We model the input program as a graph $G(V, E)$, where each vertex ($v_i \in V$) is one of the functions, and the edges ($e_i \in E$) correspond to the FIFOs connecting them as shown in Figure 5 (A). Our goal is to map each vertex v_i to an FPGA
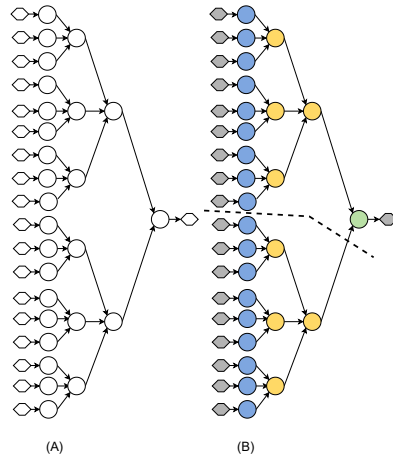

Figure 4: (A) Topology of the KNN application as found by the graph extraction step of TAPA-CS. Here, circles are compute modules and hexagons indicate HBM access. (B) Partition found by TAPA-CS is indicated by the dashed line.

F_i in the cluster such that the inter-FPGA communication cost is minimized while ensuring the compute-load between the multiple FPGAs is balanced. We explain how we model this cost in Section 4.3. We also apply several chip-level optimizations to ensure high frequency as discussed in Section 4.5.

4.2. Key Steps

There are seven major steps in TAPA-CS:
1. **Task graph construction**: We model the input workload as a graph $G(V, E)$ where nodes are compute modules and edges are the FIFOs connecting them as shown in Figure 5(A).
2. **Task extraction and parallel synthesis**: We extract and synthesize each compute module in parallel providing HLS an accurate resource utilization profile as shown in Figure 5 (B).
3. **Inter-FPGA floorplanning**: We use the resource utilization profile to intelligently floorplan this design across multiple FPGAs connected to each other through any topology (daisy-chained, ring, bus, star, mesh, hypercube, etc.) and data transfer protocol (PCIe, Ethernet, etc.). This step allows us to address the key limitation of traditional CAD toolflows (discussed in Section 2) by providing the scheduling and binding stage an accurate view of the topology and available programmable resources. Our goal is to assign each compute module to an FPGA as shown in Figure 5 (C). We explain the details of this step in Section 4.3.
4. **Inter-FPGA communication logic insertion**: After mapping the design to multiple devices, we add the inter-FPGA communication logic as shown in Figure 5(D). We discuss details of the communication logic in Section 4.4.
5. **Intra-FPGA floorplanning**: We intelligently floorplan the design across each FPGA chip by providing the scheduling and binding stage information about the locations of the
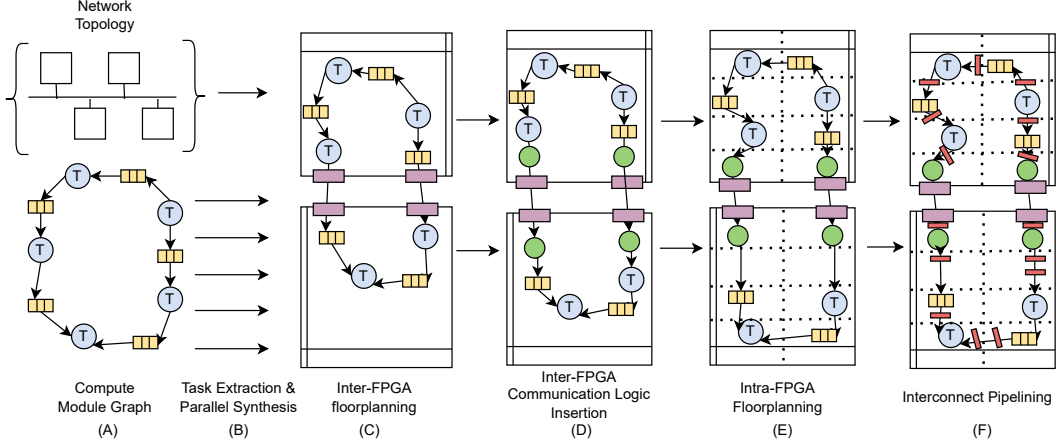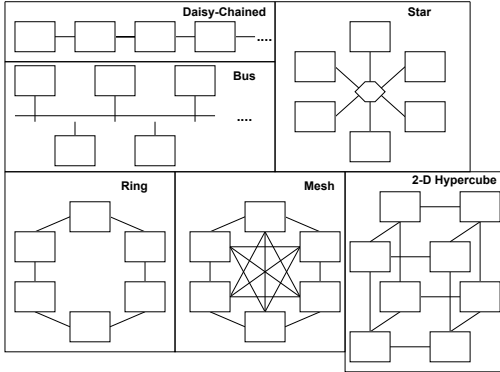
4

Figure 5: Key Steps in TAPA-CS

Figure 6: Network Topologies

hard IPs, and I/O ports, and the internal chip layout as shown in Figure 5 (E). Section 4.5 details how we formalize this information and divide the FPGA into multiple slots.

6. **Interconnect Pipelining**: We add pipeline registers to the interconnect at the slot crossings to ensure high frequency designs as shown in Figure 5 (F). We also ensure correctness and that the final design execution cycles are not compromised by this interconnect pipelining step as discussed in Section 4.6.

7. **Bitstream generation**: Finally, the optimized designs and floorplanning constraints found by TAPA-CS are passed back into the traditional CAD stack to produce the bitstreams.

TAPA-CS is completely integrated with existing FPGA CAD toolflows. While the method proposed in TAPA-CS can be applied to any multi-FPGA setup, for the scope of this paper, we test TAPA-CS on Xilinx/AMD Alveo boards. In the following Sections, we discuss each of these steps in detail and display the features through which our tool achieves high throughput and frequency accelerators.

4.3. Inter-FPGA Module Mapping

In this step, we provide the traditional CAD toolflows with a view of the available FPGA devices, their topologies, and the hierarchy of the user application, so that we can automatically find the optimal module-to-FPGA mapping. Design partitioning is enabled by the latency-insensitive nature of dataflow designs. Latency-insensitive design [22, 21] decouples the design of the interconnect from that of the compute modules. This allows the designer to add interconnect pipeline registers, and connections over different networks with arbitrarily long latency between compute modules, without affecting the functional correctness of the design.

In this mapping step, our goal is to minimize the high inter-FPGA communication cost while ensuring there is no resource congestion. If we are using two FPGAs, we consider the total available region to be divided into two grids as shown in Figure 5 (C).

Now, the placement task reduces to assigning each task to one of the two grids. For this, we use an ILP-based formulation to obtain exact partitioning solutions. While heuristic solvers are faster, ILP allows an accurate solution. We also show in Section 5 that both our intra- and inter- FPGA partitioning algorithms only add between 2.8-49.7 seconds overhead to the overall compilation flow for number of compute modules ranging from 30 to 493, making the method scalable.

Let the binary variable denoting whether vertex v_i is placed on device F_i or not be v_d. Let the task $v_i \in V$ have a resource utilization profile of v_{area}. If r_v is the resources used by the set of tasks already placed in device F_i, then, before placing a new task v_i in this device, we need to ensure that there are enough resources. That is, for each type of on-chip resource,

$$\sum_{v \in r_v} v_d \times v_{area} < T, \tag{1}$$

where T is the threshold of utilization for each resource. Next, our ILP solver ensures that the cost of inter-FPGA communication is minimized. Therefore, we consider the placement of

5

all the neighbors of task v_i in the cost function as follows:

$$\sum_{e_{ij}\in E} e_{ij}.width \times dist(F_i,F_j) \times \lambda \quad (2)$$

where $e_{ij}.width$ is the bitwidth of the FIFO channel connecting the two vertices v_i and v_j, function $dist(F_i,F_j)$ is a metric of the cost of communication between tasks v_i and v_j placed on the same or different FPGAs, and λ is a scaling factor to adjust the cost for different data transfer protocols like Ethernet and PCIe.

The communication cost function $dist(F_i,F_j)$ depends on the topology of the network-connected FPGAs. Consider the network topologies shown in Figure 6. In case the FPGAs are daisy-chained,

$$dist(F_i,F_j) = |F_i.device\_num - F_j.device\_num| \quad (3)$$

where $F_i.device\_num$ and $F_j.device\_num$ are the device IDs associated with the FPGAs. Similarly, in the case of a bidirectional ring topology, the distance metric changes to:

$$dist(F_i,F_j) = \min(|F_i.device\_num - F_j.device\_num|,$$
$$(total\_num - |F_i.device\_num - F_j.device\_num|))$$

where $total\_num$ is the total number of FPGAs in the ring.

The scaling factor λ is used to adjust the cost in a system with multiple interconnection media. We use Ethernet-based connections offering 100GBps bandwidths as the baseline and scale the cost for other media accordingly. For example, if the interconnection used is PCIe Gen3x16, then the cost is scaled by a factor of 12.5 compared with the cost of using Ethernet-based connections.

Note that our partitioner does not always recommend the min-cut. For the placement of each module, we consider the resource and communication cost added by its placement on- and off-chip. In case the module can be accommodated on the same chip as its neighbors, we would pay a high price by moving the compute module off-chip than placing it on-chip. However, if the placement of this module on-chip results in congestion (which in turn lowers design frequency), it would be more beneficial to place the module off-chip even at the cost of increased inter-FPGA connections. This trade-off ensures that we can achieve high frequency designs on both FPGAs.

4.4. Inter-FPGA Communication

Despite the excellent opportunity to leverage networking capabilities in modern FPGAs, existing CAD tools do not explicitly support networking. TAPA-CS supports a library of inter-FPGA communication protocols, such as Ethernet-based RoCE v2, and PCIe-based P2P DMA [14]. However, for the scope of this paper, we limit our discussions and evaluations to using the QSFP28 Ethernet ports. Here, we use AlveoLink [3] described in Figure 7 to illustrate how to add networking support to the existing toolflows. AlveoLink ensures reliable, lossless, and in-order data transfer with a low

resource overhead of $\sim$5% on the Alveo U55C cards. It offers a low round-trip data transfer latency of 1 μs between two FPGAs. AlveoLink is 12.5x faster compared with PCIe Gen3x16 -based connections. AlveoLink's main components are as follows:

1. HiveNet IP: This is a Vitis-compatible implementation of the RoCE v2 protocol which directly connects to user kernels.
2. CMAC kernel: This is a board-specific interface between the signals detected at the QSFP28 ports and the signals detected at the commodity network.
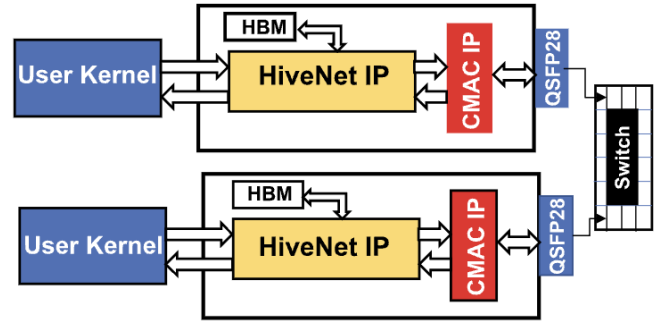

Figure 7: AlveoLink features as described in [3]

4.5. Intra-FPGA Module Mapping

After assigning each task to the set of devices in our cluster and adding the inter-FPGA communication interfaces, the next step is to apply a similar top-down partitioning approach to each FPGA (Figure 5(E)). To formalize the device-specific information and present it to the scheduling and binding stage, we view each FPGA as a grid divided into slots by the hard IPs and static regions. For example, the Alveo U55C card shown in Figure 2 is presented to TAPA-CS as a grid with 6 slots divided into two columns and 3 rows. Our goal is to place each vertex in one of these slots based on the resource utilization ratios per slot and the cost of connecting this module to all its neighbors. In this step, our goal is to minimize the cost of inter-die communication. Therefore, the new cost function is:

$$\sum_{e_{ij}\in E} e_{ij}.width \times (|v_i.row - v_j.row| + |v_i.col - v_j.col|) \quad (4)$$

where $v_i.row$, $v_j.row$ represent the rows in which tasks v_i and v_j are placed respectively, and the same for the columns. We continue such a two-way ILP-based partitioning scheme until we divide each FPGA into eight grids.

One of the key considerations in this step is the optimal usage of the HBM channels. Consider the example of the U55C cards. While the HBM offers a high aggregate bandwidth of 460GBps, it is still 76x slower than on-chip data accesses.

Resource Type	Available
LUT	1146240
FF	2292480
BRAM	1776
DSP	8376
URAM	960

Table 2: Resource availability on the Alveo U55C cards.

Therefore, it is important to utilize the HBM channels exposed to the user kernel optimally. As shown earlier in Figure 2, all the HBM channels on-board the U55C are exposed in the bottom-most die. Suboptimal HBM channel binding can result in large routing delays and increase congestion in this die, leading to routing failure. Therefore, TAPA-CS supports an automatic HBM channel binding exploration where we find the optimal mappings based on the workload characteristics.

4.6. Interconnect Pipelining

Following the intra-FPGA optimizations, we also conservatively pipeline the interconnect at all slot-crossings to prevent long delays from degrading the final clock frequency. In contrast to prior work like [41, 22], we conservatively pipeline all slot-crossing wires because each of our compute modules compiles into an RTL controlled by a finite state machine (FSM). Therefore, it is difficult to estimate the latency added by the pipelining step. Next, to ensure that the design throughput is not negatively affected by the additional pipeline registers, we also balance the latency of parallel paths based on cut-set pipelining [48] as shown in [36]. In this step, the latency added by reconvergent paths is balanced to ensure that final correctness is not impacted. The pipeline FIFOs are indicated in red in Figure 5(F).

5. Evaluation

We implement TAPA-CS in Python and integrate it with Vitis 2022.1. Our ILP formulations described in Sections 4.3 and 4.5 can be solved through either MIP [12] or the Gurobi solver [8] (free for academia). We test TAPA-CS on a server equipped with four Xilinx Alveo U55C cards connected through their QSFP28 ports using active cables. The total programmable resources available per card are shown in Table 2. This board can achieve a maximum design frequency of 300MHz. The server features a 128 core AMD EPYC 7V13 CPU operating at 2.45GHz.

5.1. Benchmarks and Baselines

We evaluate TAPA-CS over multiple variations of the following benchmarks:

1. Stencil Dilate: This is a 2-D 13-point stencil kernel from the Rodinia HLS benchmark [29, 56] which is used in biomedical research to track leukocytes in blood vessels. We test this kernel over multiple iterations ranging from 64 to 512 across 2-4 FPGAs.

2. Page Rank created by [25]: This kernel features eight PEs and one central controller with dependency cycles between the compute modules. It implements the algorithm described in [47]. We test this design over multiple graph sizes to measure the design throughput.

3. KNN created by [44]: This kernel features 17 compute modules implementing an optimized accelerator for calculating each data point's distance to its neighbor, and sorting the distances to obtain the K-nearest neighbors. We test this design across varying input sizes and feature dimensions.

4. Systolic-array CNN accelerators created by AutoSA [58]. This systolic array accelerator consists of multiple PEs arranged in a grid format, with a total of 493 compute modules. The CNN we choose is an implementation of the third layer of the VGG model [52]. We test TAPA-CS on multiple grid dimensions ranging from 13 x 4 to 13 x 20.

The topology of each of the benchmarks is shown in Figure 8. We compare the performance of TAPA-CS across the tested designs in terms of frequency and latency with a single FPGA implementation generated by Vitis HLS as a baseline. To better understand the effects of scaling up to multiple FPGAs, we also compare with a single FPGA version which features the intra-FPGA floorplanning and interconnect pipelining discussed in Section 4.5. We refer to this enhanced baseline as Vitis+F&P. Comparisons with Vitis+F&P allows us to quantify the benefits obtained through floorplanning and interconnect pipelining and those obtained through span-out acceleration across multiple FPGAs. These baselines allow us to demonstrate that our tool can successfully route large designs which fail placement and routing on a single FPGA, across multiple FPGAs. On the designs which can be successfully routed on a single FPGA, TAPA-CS generates better optimized designs across 2 FPGAs, leading to higher throughput and frequency. We also discuss the scalability of the tool to 4 FPGAs in Section 5.6.

We report the floorplanning overheads added to the overall compile time by the inter- and intra-FPGA floorplaning steps for the Stencil (smallest benchmark in terms of number of compute modules) and CNN (largest benchmarks in terms of number of compute modules) benchmarks to demonstrate the low overheads added by TAPA-CS.

5.2. Stencil

Stencil kernels apply a sliding window (or stencil) of computation over an input array to produce an output array. Stencil kernels can either be memory-bound or compute-bound based on the input size and the number of iterations. Prior work [56] found that for a fixed input size, stencil designs with smaller number of iterations are memory-bound while designs with larger number of iterations are compute-bound. We chose the Dilate kernel from the Rodinia HLS benchmark to test TAPA-CS. It is a 2D 13-point kernel which we test over an input size of 4096x4096 with iterations ranging from 64 to 256. We also demonstrate a 4-FPGA design for 512 iterations
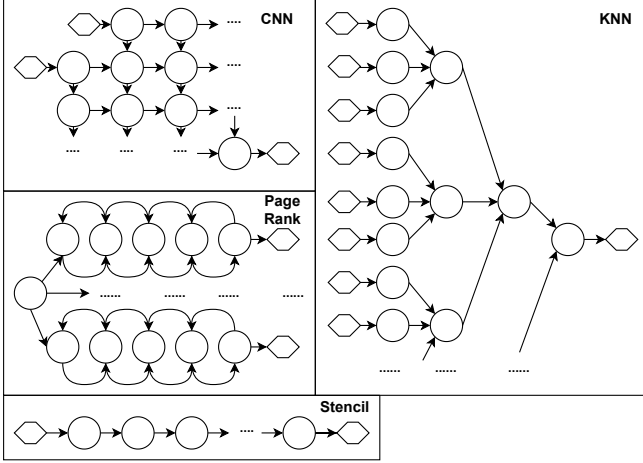
Figure 8: Topology of benchmarks. Here, circles represent compute modules while hexagons represent HBM access.

in Section 5.6. The topology of this design is shown in Figure 8.

We compare the design frequency obtained by TAPA-CS and our baselines in Table 3. Consider the design with 64 iterations. Here, Vitis HLS generates a single-FPGA design with a frequency of 165 MHz. Next, we introduce intelligent floorplanning and pipelining to this single-FPGA design achieving a final design frequency of 250MHz. Finally, we find that the 2-FPGA version generated by TAPA-CS achieves a frequency of 300 MHz on each FPGA. We also compare the latency of these designs in Figure 9. As can be observed, despite this design configuration being successfully routed on a single FPGA, the two-FPGA version obtains a 1.9x speed-up gain and 82% frequency improvement compared with Vitis HLS. Also, compared with a single-FPGA design which features the intra-FPGA floorplanning and pipelining, TAPA-CS achieves a 1.4x speed-up and 20% frequency improvement. The key reason behind this is that span-out acceleration provides a better opportunity to exploit the HBM bandwidth and on-chip memory. In the single-FPGA case, there are a total of 16.5k transfers between the HBM and the on-chip compute modules. When using TAPA-CS, the total number of transfers between HBM and on-chip modules reduces by a factor of 2x, greatly reducing the overall design latency. This results in the processing elements spending lesser idle time. Therefore, even though this design configuration can be routed on a single FPGA, using two FPGAs allows better performance.

Note that out of the 3 iteration variations (64, 128, and 256) tested for this input configuration, the design with 128 and 256 iterations fail in either the placement or routing phases of traditional CAD tools as shown in Table 3. However, TAPA-CS can successfully route them across 2 FPGAs achieving an average design frequency of 300MHz.

The floorplanning overheads added by our tool using the Gurobi solver are shown in Table 4. The stencil design has 30 compute modules connected through FIFO channels. As

Iterations	Vitis (MHz)	Vitis+F&P (MHz)	TAPA-CS (F1, F2) (MHz)
64	165	250	300, 300
128	-	-	300, 300
256	-	-	300, 300

Table 3: Stencil: Frequency comparison between TAPA-CS and Vitis HLS with and without floorplanning and pipelining. Here, "-" implies that the design failed to complete placement or routing, and "F1, F2" refers to the 2 FPGAs used by TAPA-CS.
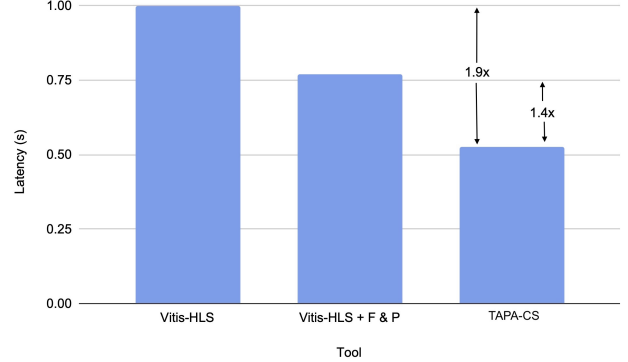

Figure 9: Stencil: Latency comparison between TAPA-CS and single-FPGA versions generated using Vitis HLS with and without floorplanning and pipelining.

can be observed from the values, the time taken to solve the ILP formulation for inter- and intra-FPGA floorplanning adds a low overhead of < 1.24 seconds, making our tool scalable.

Iterations	Inter-FPGA (s)	Intra-FPGA (F1, F2) (s)
64	1.22	0.74, 0.65
128	1.22	0.81, 0.72
256	1.24	0.79, 0.80

Table 4: Stencil: Additional floorplanning time added to overall HLS compile time. Here, "F1, F2" refers to the 2 FPGAs used by TAPA-CS.

5.3. Page Rank

The topology of the Page Rank application is as shown in Figure 8. This accelerator implements the citation ranking algorithm described in [47, 25]. First the input graph is pre-processed on the host and loaded onto the device HBM. Then, the edges are streamed to each PE on-chip, which calculate and propagate weighted rankings from source vertex to destination vertex. These updates are stored back into the HBM before they are accumulated over each vertex to calculate the final ranking.

Table 5 shows the frequency and resource utilization comparison with a single FPGA accelerator designed through Vitis with and without the floorplanning and pipelining introduced in Section 4. For this 8-PE design configuration, we obtain a frequency improvement of 116.26% increase compared with Vitis and 40% increase compared with a single FPGA design
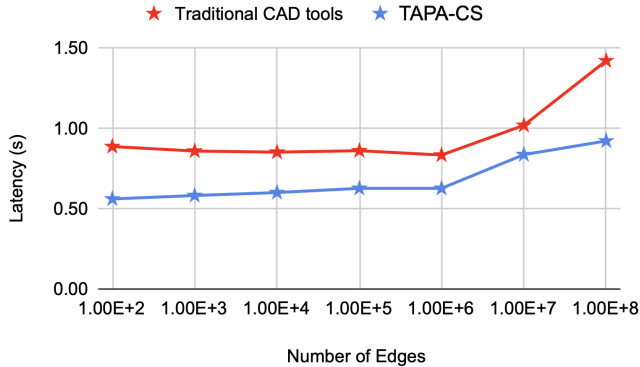
Figure 10: Page Rank: Latency comparison of TAPA-CS and Vitis HLS over increasing number of edges in the input graph. The performance improvement obtained by TAPA-CS increases as the input graph becomes larger.
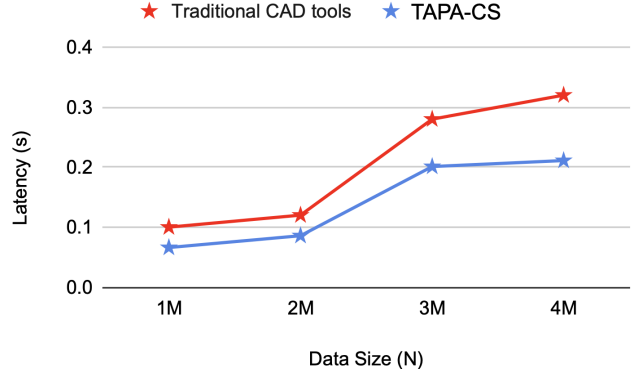

Figure 11: KNN: Latency comparison between TAPA-CS and Vitis HLS on varying input dataset sizes. Here, we fix the dimension size M as 2 and K as 10.

5.4. KNN

We use the KNN accelerator designed by [44] to test TAPA-CS. As discussed in Section 3, this design consists of a total of 27 compute modules of three different types. The topology of this design is shown in Figure 8. Table 7 shows the frequency and resource utilization comparison of TAPA-CS and Vitis and Vitis+F&P. For this design configuration we obtain a frequency improvement of 87.5% compared with Vitis and 31.6% increase compared with Vitis+F&P while using a fraction of the resources per FPGA card.

As discussed in Section 3, TAPA-CS can route an optimized version of the KNN application which fails placement and routing through traditional CAD tools. In this version, we use a port width of 512 bits and a data access size of 128KB. This configuration can best saturate the per-bank bandwidth exposed by the HBM. We study the latency comparison between this version generated by TAPA-CS, and the single-FPGA unoptimized version generated by Vitis+F&P. First, we compare the latency obtained by the baseline and TAPA-CS over increasing dataset sizes in Figure 11. While the performance for smaller dataset sizes is comparable, TAPA-CS achieves a 1.5x speed-up for larger dataset sizes. Similarly, we measure the latency obtained when varying feature dimension in Figure 12. In this case, TAPA-CS achieves a 1.6x speed-up compared with the baseline for larger dimension size. Therefore, TAPA-CS uses the available HBM banks in a more optimized manner than traditional CAD tools, resulting in higher speed-up as the computational and memory complexity scales up.

with floorplanning and pipelining, while using a fraction of the resources on each FPGA.

	Vitis	Vitis+F&P	TAPA-CS (F1, F2) (MHz)
Fmax (MHz)	123	190	266, 266
LUT %	83.19	67.83	10.26, 10.26
FF %	14.71	14.71	6.21, 6.21
BRAM %	4.73	4.95	2.93, 2.93
DSP %	15.47	15.39	7.74, 7.74
URAM %	53.33	53.33	26.67, 26.67

Table 5: Page Rank: Frequency, Resource Usage Comparison. Here, F1, F2 refer to the 2 FPGAs used by TAPA-CS.

We also study the latency obtained across graphs of increasing sizes in Figure 10. These graphs are synthetically generated for the purpose of this analysis. Across the tested graph sizes, TAPA-CS achieves a speed-up of 1.42x.

To analyze the scalability of TAPA-CS to real-world graphs, we use the Berkley-Stanford web graph [43] consisting of about 7.6M edges and 700k nodes. The latency results and the number of edges processed per second for this graph are shown in Table 6.

Reducing the number of HBM accesses per FPGA can significantly reduce the idle time of PEs and the overall latency of the design. We test the single-FPGA and TAPA-CS-based two-FPGA Page Rank accelerators on a toy graph with 10 edges. In case of the single FPGA design, there are a total of 188 HBM accesses while the same design across two FPGAs observes 2x lesser HBM accesses. This reduces the idle time of the PEs and ensures a high bandwidth utilization.

Tool	Latency (s)	Edges/Second
Vitis + F& P	1.19	6347158.20
TAPA-CS	0.84	9079240.03

Table 6: Page Rank: Latency comparison between TAPA-CS and Vitis HLS for the Berkley-Stanford web graph [43].

5.5. CNN

The CNN accelerator we chose is a systolic-array based implementation of the third layer of the VGG model [52] generated by AutoSA [58]. This accelerator consists of a grid of PEs performing identical computations. The grid size can be adjusted to meet throughput/resource constraints.

We test TAPA-CS over varying grid sizes between 13x4 to 13x20. Out of the five grid topologies, four fail placement or
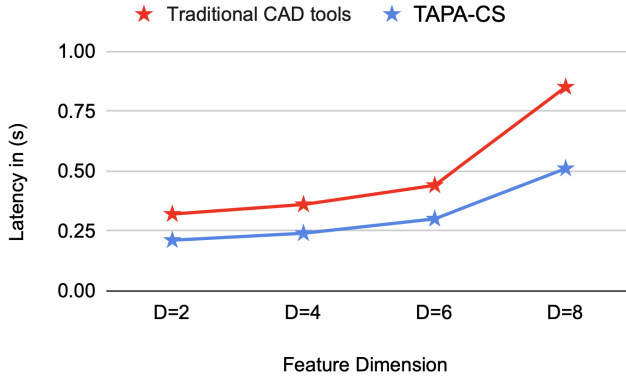
Figure 12: KNN: Latency comparison between TAPA-CS and Vitis HLS on varying feature dimension sizes. Here, we fix the dataset size (N) as 4M and K as 10.

	Vitis	Vitis+F&P	TAPA-CS (F1, F2)(MHz)
Fmax (MHz)	165	198	300, 300
LUT %	68	67.9	39.2, 39.0
FF %	31.3	32.7	19.5, 19.0
BRAM %	66.1	65.8	35.6, 35.6
DSP %	11	11.9	1.5, 1.5
URAM %	15	15	5.6, 5.6

Table 7: KNN: Frequency, Resource Usage Comparison. Here, "F1, F2" are the 2 FPGAs used by TAPA-CS.

routing stages using traditional Vitis tools and two fail using Vitis+F&P. TAPA-CS can successfully route them achieving an average design frequency of 300 MHz. Table 9 and 10 describe the resource utilizations and frequency comparisons respectively. Overall, we find that TAPA-CS can successfully route all the configurations with a final design frequency of 300MHz per board using a fraction of the resources compared with traditional CAD flows. While the throughput achieved by TAPA-CS remains the same as those achieved by Vitis+F&P for the 13x4 and 13x8 configurations, we observe a 1.3x speedup compared with the 13x12 implementation generated by Vitis+F&P. Therefore, TAPA-CS enables better optimization of larger designs across multiple chips.

Since the CNN application has the largest number of compute modules (493), we measure the floorplanning overheads added by this application in TAPA-CS. The floorplanning overheads are mentioned in Table 8. We find that the floorplanning steps add at most 49.7 seconds to the overall compilation time, making our tool scalable even for large designs.

5.6. Scalability

One of the major shortcomings of prior work discussed in Table 1 and Section 6 is the inability to scale designs beyond 2 FPGAs. TAPA-CS uses the network topology information to accurately model the inter-FPGA communications costs and partition the design efficiently. To display the scalability of TAPA-CS beyond two FPGAs, we use the stencil design presented in Section 5.2 as an example. The single FPGA

Configuration	Inter-FPGA (s)	Intra-FPGA (s)
13x4	0.27	0.13, 0.10
13x8	4.72	2.7, 2.16
13x12	14.69	7.11, 7.10
13x16	19.53	8.98, 9.32
13x20	24.61	12.25, 12.85

Table 8: CNN: Floorplanning overheads added to the overall HLS compile time.

design could successfully route an 8 PE stencil design with 64, 128, and 256 iterations. To scale the design, we consider a 32 PE design with 512 iterations. Such a design has 4x more HBM accesses than the 8 PE design, thus lowering the achievable HBM bandwidth to <10GBps. Also, the increased computational intensity causes the design to fail routing on a single FPGA. Similarly, this design cannot be routed even on 2 FPGAs due to the high requirement of both memory and programmable resources. We distribute 8 PEs across each FPGA to enable a 4-FPGA acceleration of this design. Each of the partitioned designs achieve a frequency of 300MHz and has a resource utilization as shown in Table 11.

6. Related Works

6.1. Prior Work on Inter-FPGA Communication

Prior attempts at leveraging the networking capabilities exposed by modern FPGAs can be classified into two categories. The first category of work uses host-orchestrated data transfers [51, 50, 38]. In these works the host coordinates the inter-accelerator communication by exposing programmer-friendly MPI-like primitives. Using host-orchestration avoids re-programming the FPGA bitstream whenever a different communication pattern is to be followed. However, considering that several dataflow FPGA workloads suit streaming, where data is produced and consumed every cycle, host-orchestration adds significant overheads. The second category of works uses device-side initiation of inter-FPGA communication [54, 30, 37]. Such papers benefit from the streaming nature of designs but suffer from frequent regeneration of the bitstream.

We compare prior work and AlveoLink (described in Section 4.4 in Table 12. Compared with EasyNet [37] which also achieves a similar data transfer throughput of 90GBps, AlveoLink requires about half of the on-board resources. This allows larger designs to be mapped to the FPGA and utilize networking capabilities than in case of EasyNet.

6.2. Prior Work on Partitioning, Mapping, & High Design Frequency

There are some initial research efforts which address partitioning and mapping across multiple FPGAs which provide a good starting point for TAPA-CS, but they suffer from several shortcomings. Prior work such as Elastic-DF [15] propose

Configuration	LUT %		FF %		BRAM %		DSP %		URAM %	
	Orig	TAPA-CS	Orig	TAPA-CS	Orig	TAPA-CS	Orig	TAPA-CS	Orig	TAPA-CS
13x4	20.1/20.4	5.9, 6.1	12.1/12.3	4.5, 4.3	14.2/14.4	17.2, 15.9	25.2/24.5	7.6, 8.2	0/0	0, 0
13x8	38/38.3	11.1, 11.5	23/23.5	8.3, 9.2	23.3/23.7	26.5, 25.1	48.4/49	15.3, 14.1	0/0	0, 0
13x12	55.8/56.1	27.3, 28.8	34.3/34.6	17, 16.5	32.7/33.1	18.1, 18.6	73.1/73.4	22.6, 23.1	0/0	0, 0
13x16	73.3/74	35.2, 36.8	45.2/45.7	22.1, 23.5	42.3/42.5	19.3, 18.7	97.6/97.9	30.1, 29.9	0/0	3.3, 3.1
13x20	91.8/91.9	44.7, 45.9	56.9/57.2	28.4, 28.2	51.8/52.1	25.1, 25.0	122.4/123.7	41.2, 41.5	0/0	5.0, 6.1

Table 9: CNN: Resource utilization comparison between TAPA-CS and Vitis/Vitis+F&P for different systolic array configurations. Here, the column "Orig" refers to the resource utilization of Vitis and Vitis+F&P seperated by a "/". The resource utilization of TAPA-CS is shown for both FPGAs seperated by ",".

Configuration	Vitis (MHz)	Vitis+F&P (MHz)	TAPA-CS (F1, F2) (MHz)
13x4	300	300	300, 300
13x8	-	300	300, 300
13x12	-	250	300, 300
13x16	-	-	300, 300
13x20	-	-	300, 300

Table 10: CNN: Frequency comparison between TAPA-CS and Vitis / Vitis+F&P across different configurations. Here, "-" implies that the corresponding design failed to complete placement or routing, and "F1, F2" refers to the 2 FPGAs used by TAPA-CS.

	Vitis	TAPA-CS (F1, F2, F3, F4)
Fmax (MHz)	-	300, 300, 300, 300
LUT %	70.7	26.4, 23.0, 24.1, 25
FF %	76.0	16.0, 16.6, 15.9, 16.8
BRAM %	54.8	13.7, 13.0, 13.2, 13.6
DSP %	0.0	0.0, 0.0, 0.0, 0.0
URAM %	0.0	0.0, 0.0, 0.0, 0.0

Table 11: Stencil: Span-out acceleration of a 32 PE stencil design across 4 FPGAs. Here, "-" implies that the design fails to complete placement or routing. "F1, F2, F3, F4" are the 4 FPGAs used by TAPA-CS.

an ILP-based partitioner similar to ours, but suffer from poor frequency (190-240MHz) as they do not couple floorplanning and interconnect pipelining with HLS compilation. Also, Elastic-DF is integrated with the DNN inference compiler FINN [57, 19], leading to poor generalizability to different workloads.

A different approach to design partitioning leverages latency-insensitivity [22, 21] to break the design at the latency-insensitive endpoints [32, 60, 61]. Such works view the design as consisting of multiple modules connected by FI-

Project	Orchestration	Resource Overhead (%)	Performance (GBps)
TMD-MPI[51]	Host	26	10
Galapagos[54]	Device	11.5	10
SMI[30]	Device	2	40
EasyNet[37]	Device	10	90
ZRLMPI[50]	Host	-	10
ACCL[38]	Host	16	80
AlveoLink[3]	Device	5	90

Table 12: Comparison of prior work addressing Challenge 1 in terms of data transfer throughput and FPGA resource area overhead. Here, "-" implies that the project does not discuss the area overhead.

FOs, allowing the authors freedom in implementing the inter-module communication. [32] leverage this technique, but expect the user to provide a static module-to-FPGA mapping, use Verilog/VHDL for specifying the modules, and perform simulation-based experiments. In case of TAPA-CS, we partition the design at latency-insensitive endpoints and automatically find the optimal module-to-FPGA mapping, expose an easy-to-use C++ interface to the user, and perform experiments on real FPGAs achieving high frequency. Virtualization-based works such as [60, 61, 62] assign modules to pre-placed and pre-routed "soft blocks" which does not scale well to real-world large-scale designs where each function is compiled into an RTL controlled by a finite state machine (FSM) and has varying resource requirements.

SMAPPIC [26] introduces a multi-node emulation system where each node can be a single die of an FPGA or the whole FPGA. It uses the computational cores shipped with BYOC [16] assign cores to the nodes. However, SMAPPIC uses Gen3x16 PCIe-based connections between FPGAs which provides a slow round-trip latency of 1250ns, and does not provide the tool with a hardware layout of the FPGAs. This leads the designs to have a low final frequency of 100MHz. In contrast, TAPA-CS has an inter-FPGA round trip latency of 1 μs (12.5x faster than SMAPPIC), provides the partitioning tool with a global view of the chip layout allowing us to achieve a high frequency of between 266-300MHz. We also do not fix the granularity of resource allocation for a computational core/module to a node, allowing greater flexibility in the target applications. For example, in TAPA-CS a single die can contain any number of modules, and modules spanning across multiple dies are pipelined sufficiently to maintain the final frequency.

7. Conclusion and Outlook

This paper presents TAPA-CS - a task-parallel dataflow programming framework which automatically partitions, generates, and compiles a large design across a cluster of FPGAs achieving high throughput and frequency. TAPA-CS uses an ILP-based partitioning framework which takes into account resource utilization profiles of compute modules before mapping them to different FPGAs. It also couples a coarse-grained floorplanning step with pipelining at the inter- and intra-FPGA levels to ensure high accuracy. We test the design across

multiple benchmarks of varying compute and memory requirements to validate the scalability and generalizability of the tool. Across all the tested designs, TAPA-CS achieves an average throughput improvement of 1.45x and frequency improvement between 18-116% compared with existing CAD tools. We plan to open-source TAPA-CS once the paper is accepted, which will allow us or the community to add support for Intel FPGAs in the near future.

References

[1] Alibaba FPGAs in the Cloud. https://www.alibabacloud.com/help/en/fpga-based-ecs-instance.

[2] Alveo U55C High Performance Compute Card. https://www.xilinx.com/products/boards-and-kits/alveo/u55c.html#specifications.

[3] AlveoLink. https://github.com/Xilinx/AlveoLink.

[4] Amazon AQUA. https://aws.amazon.com/redshift/features/.

[5] Amazon EC2 F1 Instances. https://aws.amazon.com/ec2/instance-types/f1/.

[6] AMD/Xilinx UltraScale+ Devices Overview. https://docs.xilinx.com/r/en-US/ug1120-alveo-platforms/Overview.

[7] Baidu FPGAs in the Cloud. https://intl.cloud.baidu.com/product/bcc.html.

[8] Gurobi Solver. https://www.gurobi.com/downloads/gurobi-optimizer-eula/.

[9] Huawei FPGAs in the Cloud.

[10] Intel HLS. https://www.intel.com/content/dam/www/central-libraries/us/en/documents/hls-production-brief.pdf.

[11] Intel Stratix 10.

[12] Python MIP. https://www.python-mip.com/.

[13] Vitis HLS 2022.2. https://docs.xilinx.com/r/en-US/ug1399-vitis-hls.

[14] Xilinx PCIe-Based P2P. https://xilinx.github.io/XRT/master/html/p2p.html.

[15] Tobias Alonso, Lucian Petrica, Mario Ruiz, Jakoba Petri-Koenig, Yaman Umuroglu, Ioannis Stamelos, Elias Koromilas, Michaela Blott, and Kees Vissers. Elastic-DF: Scaling Performance of DNN Inference in FPGA Clouds through Automatic Partitioning. *ACM Trans. Reconfigurable Technol. Syst.*, 15(2), dec 2021.

[16] Jonathan Balkind, Katie Lim, Michael Schaffner, Fei Gao, Grigory Chirkov, Ang Li, Alexey Lavrov, Tri M. Nguyen, Yaosheng Fu, Florian Zaruba, Kunal Gulati, Luca Benini, and David Wentzlaff. BYOC: A "Bring Your Own Core" Framework for Heterogeneous-ISA Research. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, page 699–714, New York, NY, USA, 2020. Association for Computing Machinery.

[17] Chaim Baskin, Natan Liss, Evgenii Zheltonozhskii, Alex M. Bronstein, and Avi Mendelson. Streaming Architecture for Large-Scale Quantized Neural Networks on an FPGA-Based Dataflow Platform. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, may 2018.

[18] Chaim Baskin, Natan Liss, Evgenii Zheltonozhskii, Alex M. Bronstein, and Avi Mendelson. Streaming Architecture for Large-Scale Quantized Neural Networks on an FPGA-Based Dataflow Platform. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 162–169, 2018.

[19] Michaela Blott, Thomas Preusser, Nicholas Fraser, Giulio Gambardella, Kenneth O'Brien, and Yaman Umuroglu. FINN-R: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks, 2018.

[20] Shekhar Borkar and Andrew A. Chien. The Future of Microprocessors. *Commun. ACM*, 54(5):67–77, may 2011.

[21] L.P. Carloni, K.L. McMillan, A. Saldanha, and A.L. Sangiovanni-Vincentelli. A methodology for correct-by-construction latency insensitive design. In *1999 IEEE/ACM International Conference on Computer-Aided Design. Digest of Technical Papers (Cat. No.99CH37051)*, pages 309–315, 1999.

[22] L.P. Carloni, K.L. McMillan, and A.L. Sangiovanni-Vincentelli. Theory of latency-insensitive design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(9):1059–1076, 2001.

[23] 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. A Cloud-Scale Acceleration Architecture. In *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, October 2016.

[24] Yuze Chi, Licheng Guo, Jason Lau, Young-kyu Choi, Jie Wang, and Jason Cong. Extending High-Level Synthesis for Task-Parallel Programs. In *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 204–213, 2021.

[25] Yuze Chi, Licheng Guo, Jason Lau, Young kyu Choi, Jie Wang, and Jason Cong. Extending High-Level Synthesis for Task-Parallel Programs, 2021.

[26] Grigory Chirkov and David Wentzlaff. SMAPPIC: Scalable Multi-FPGA Architecture Prototype Platform in the Cloud. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS 2023, page 733–746, New York, NY, USA, 2023. Association for Computing Machinery.

[27] Young-kyu Choi, Yuze Chi, Weikang Qiao, Nikola Samardzic, and Jason Cong. HBM Connect: High-Performance HLS Interconnect for FPGA HBM. In *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '21, page 116–126, New York, NY, USA, 2021. Association for Computing Machinery.

[28] Eric Chung, Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Adrian Caulfield, Todd Massengill, Ming Liu, Mahdi Ghandi, Daniel Lo, Steve Reinhardt, Shlomi Alkalay, Hari Angepat, Derek Chiou, Alessandro Forin, Doug Burger, Lisa Woods, Gabriel Weisz, Michael Haselman, and Dan Zhang. Serving DNNs in Real Time at Datacenter Scale with Project Brainwave. *IEEE Micro*, 38:8–20, March 2018.

[29] Jason Cong, Zhenman Fang, Michael Lo, Hanrui Wang, Jingxian Xu, and Shaochong Zhang. Understanding Performance Differences of FPGAs and GPUs. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 93–96, 2018.

[30] Tiziano De Matteis, Johannes de Fine Licht, Jakub Beránek, and Torsten Hoefler. Streaming Message Interface: High-Performance Distributed Memory Programming on Reconfigurable Hardware. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '19, New York, NY, USA, 2019. Association for Computing Machinery.

[31] R.H. Dennard, F.H. Gaensslen, Hwa-Nien Yu, V.L. Rideout, E. Bassous, and A.R. LeBlanc. Design of ion-implanted MOSFET's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, 1974.

[32] Kermin Elliott Fleming, Michael Adler, Michael Pellauer, Angshuman Parashar, Arvind Mithal, and Joel Emer. Leveraging Latency-Insensitivity to Ease Multiple FPGA Design. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '12, page 175–184, New York, NY, USA, 2012. Association for Computing Machinery.

[33] Jeremy Fowers, Joo-Young Kim, Doug Burger, and Scott Hauck. A Scalable High-Bandwidth Architecture for Lossless Compression on FPGAs. In *The 23rd IEEE International Symposium on Field-Programmable Custom Computing Machines*. IEEE – Institute of Electrical and Electronics Engineers, May 2015.

[34] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Logan Adams, Mahdi Ghandi, Stephen Heil, Prerak Patel, Adam Sapek, Gabriel Weisz, Lisa Woods, Sitaram Lanka, Steve Reinhardt, Adrian Caulfield, Eric Chung, and Doug Burger. A Configurable Cloud-Scale DNN Processor for Real-Time AI. In *Proceedings of the 45th International Symposium on Computer Architecture, 2018*. ACM, June 2018.

[35] Licheng Guo, Yuze Chi, Jason Lau, Linghao Song, Xingyu Tian, Moazin Khatti, Weikang Qiao, Jie Wang, Ecenur Ustun, Zhenman Fang, Zhiru Zhang, and Jason Cong. TAPA: A Scalable Task-Parallel Dataflow Programming Framework for Modern FPGAs with Co-Optimization of HLS and Physical Design, 2022.

[36] Licheng Guo, Yuze Chi, Jie Wang, Jason Lau, Weikang Qiao, Ecenur Ustun, Zhiru Zhang, and Jason Cong. AutoBridge: Coupling Coarse-Grained Floorplanning and Pipelining for High-Frequency HLS Design on Multi-Die FPGAs. In *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '21, page 81–92, New York, NY, USA, 2021. Association for Computing Machinery.

[37] Zhenhao He, Dario Korolija, and Gustavo Alonso. EasyNet: 100 Gbps Network for HLS. pages 197–203, 08 2021.

[38] Zhenhao He, Daniele Parravicini, Lucian Petrica, Kenneth O'Brien, Gustavo Alonso, and Michaela Blott. ACCL: FPGA-Accelerated Collectives over 100 Gbps TCP-IP. In *2021 IEEE/ACM International Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC)*, pages 33–43, 2021.

[39] Weiwen Jiang, Edwin H.-M. Sha, Xinyi Zhang, Lei Yang, Qingfeng Zhuge, Yiyu Shi, and Jingtong Hu. Achieving Super-Linear Speedup across Multi-FPGA for Real-Time DNN Inference. *ACM Trans. Embed. Comput. Syst.*, 18(5s), oct 2019.

[40] Sagar Karandikar, Howard Mao, Donggyu Kim, David Biancolin, Alon Amid, Dayeol Lee, Nathan Pemberton, Emmanuel Amaro, Colin Schmidt, Aditya Chopra, Qijing Huang, Kyle Kovacs, Borivoje Nikolic, Randy Katz, Jonathan Bachrach, and Krste Asanovic. FireSim: FPGA-Accelerated Cycle-Exact Scale-Out System Simulation in the Public Cloud. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 29–42, 2018.

[41] E.A. Lee and D.G. Messerschmitt. Synchronous Data Flow. *Proceedings of the IEEE*, 75(9):1235–1245, 1987.

[42] Michel Lemaire, Daniel Massicotte, and Jean Bélanger. Multi-FPGA Communication Interface for Electric Circuit Co-Simulation. In *2020 IEEE Electric Power and Energy Conference (EPEC)*, pages 1–6, 2020.

[43] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters, 2008.

[44] Alec Lu, Zhenman Fang, Nazanin Farahpour, and Lesley Shannon. CHIP-KNN: A Configurable and High-Performance K-Nearest Neighbors Accelerator on Cloud FPGAs. In *2020 International Conference on Field-Programmable Technology (ICFPT)*, pages 139–147, 2020.

[45] Samuel Naffziger, Noah Beck, Thomas Burd, Kevin Lepak, Gabriel H. Loh, Mahesh Subramony, and Sean White. Pioneering Chiplet Technology and Design for the AMD EPYC™ and Ryzen™ Processor Families : Industrial Product. In *2021 ACM/IEEE 48th International Symposium on Computer Architecture (ISCA)*, pages 57–70, 2021.

[46] Kalin Ovtcharov, Olatunji Ruwase, Joo-Young Kim, Jeremy Fowers, Karin Strauss, and Eric Chung. Accelerating Deep Convolutional Neural Networks Using Specialized Hardware, February 2015.

[47] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank Citation Ranking : Bringing Order to the Web. In *The Web Conference*, 1999.

[48] Keshab K Parhi. VLSI digital signal processing systems: design and implementation. In *John Wiley & Sons*, 2007.

[49] Andrew Putnam, Adrian M. Caulfield, Eric S. Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, Michael Haselman, Scott Hauck, Stephen Heil, Amir Hormati, Joo-Young Kim, Sitaram Lanka, James Larus, Eric Peterson, Simon Pope, Aaron Smith, Jason Thong, Phillip Yi Xiao, and Doug Burger.

A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services. *IEEE Micro*, 35(3):10–22, 2015.

[50] Burkhard Ringlein, Francois Abel, Alexander Ditter, Beat Weiss, Christoph Hagleitner, and Dietmar Fey. ZRLMPI: A Unified Programming Model for Reconfigurable Heterogeneous Computing Clusters. In *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 220–220, 2020.

[51] Manuel Saldana and Paul Chow. TMD-MPI: An MPI Implementation for Multiple Processors Across Multiple FPGAs. In *2006 International Conference on Field Programmable Logic and Applications*, pages 1–6, 2006.

[52] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition, 2015.

[53] Zhangxi Tan, Zhenghao Qian, Xi Chen, Krste Asanovic, and David Patterson. DIABLO: A Warehouse-Scale Computer Network Simulator Using FPGAs. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '15, page 207–221, New York, NY, USA, 2015. Association for Computing Machinery.

[54] Naif Tarafdar, Nariman Eskandari, Varun Sharma, Charles Lo, and Paul Chow. Galapagos: A Full Stack Approach to FPGA Integration in the Cloud. *IEEE Micro*, 38(6):18–24, 2018.

[55] Naif Tarafdar, Giuseppe Di Guglielmo, Philip C Harris, Jeffrey D Krupa, Vladimir Loncar, Dylan S Rankin, Nhan Tran, Zhenbin Wu, Qianfeng Shen, and Paul Chow. AIgean: An Open Framework for Machine Learning on Heterogeneous Clusters. In *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 239–239, 2020.

[56] Xingyu Tian, Zhifan Ye, Alec Lu, Licheng Guo, Yuze Chi, and Zhenman Fang. SASA: A Scalable and Automatic Stencil Acceleration Framework for Optimized Hybrid Spatial and Temporal Parallelism on HBM-Based FPGAs. *ACM Trans. Reconfigurable Technol. Syst.*, 16(2), apr 2023.

[57] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. FINN. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, feb 2017.

[58] Jie Wang, Licheng Guo, and Jason Cong. AutoSA: A Polyhedral Compiler for High-Performance Systolic Arrays on FPGA. In *The 2021 ACM/SIGDA International*

Symposium on Field-Programmable Gate Arrays, FPGA '21, page 93–104, New York, NY, USA, 2021. Association for Computing Machinery.

[59] Min Xu and Fadi J. Kurdahi. Layout-Driven RTL Binding Techniques for High-Level Synthesis Using Accurate Estimators. *ACM Trans. Des. Autom. Electron. Syst.*, 2(4):312–343, oct 1997.

[60] Yue Zha and Jing Li. Virtualizing FPGAs in the Cloud. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, page 845–858, New York, NY, USA, 2020. Association for Computing Machinery.

[61] Yue Zha and Jing Li. Hetero-ViTAL: A Virtualization Stack for Heterogeneous FPGA Clusters. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 470–483, 2021.

[62] Yue Zha and Jing Li. When Application-Specific ISA Meets FPGAs: A Multi-Layer Virtualization Framework for Heterogeneous Cloud FPGAs. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '21, page 123–134, New York, NY, USA, 2021. Association for Computing Machinery.

[63] Chen Zhang, Di Wu, Jiayu Sun, Guangyu Sun, Guojie Luo, and Jason Cong. Energy-Efficient CNN Implementation on a Deeply Pipelined FPGA Cluster. In *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, ISLPED '16, page 326–331, New York, NY, USA, 2016. Association for Computing Machinery.

[64] Wentai Zhang, Jiaxi Zhang, Minghua Shen, Guojie Luo, and Nong Xiao. An Efficient Mapping Approach to Large-Scale DNNs on Multi-FPGA Architectures. In *2019 Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 1241–1244, 2019.

[65] Hongbin Zheng, Swathi T. Gurumani, Kyle Rupnow, and Deming Chen. Fast and Effective Placement and Routing Directed High-Level Synthesis for FPGAs. In *Proceedings of the 2014 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '14, page 1–10, New York, NY, USA, 2014. Association for Computing Machinery.