TaPaSCo-AIE: An Open-Source Framework for Streaming-based Heterogeneous Acceleration using AMD AI Engines

Carsten Heinz, Torben Kalkhof, Yannick Lavan, Andreas Koch
Embedded Systems and Applications Group
TU Darmstadt
Darmstadt, Germany
{heinz, kalkhof, lavan, koch}@esa.tu-darmstadt.de

***Abstract*—AMD AI Engines (AIEs) extend the design space and open up new options for coarse-grained processing in re-configurable accelerators. Pure FPGA designs for machine learning often struggle to compete with the high clock frequencies of GPUs for data-intensive workloads with only limited control flow. Having AIEs available on-chip with an FPGA fabric allows for low-latency co-processing and permits parts of an application to be placed on the most suitable kind of processing unit.**

Many data-heavy workloads, particularly in the AI domain, benefit from data streaming. With TaPaSCo-AIE, we present a framework for heterogeneous systems centered around data streams. Our framework focuses on AMD Versal devices and incorporates AI Engines and 100G network. We demonstrate the efficient use of TaPaSCo-AIE in a real-world evaluation based on a neural network, and achieve significant performance improvements over CPUs, and even exceed the performance of an A100 GPU.

***Index Terms*—FPGA, Versal, AI Engine, Heterogeneous computing, Network**

I. Introduction

Designing hardware for an FPGA design is a tedious task. The required times for synthesis, place, and route usually take hours. In addition, implementing optimized hardware designs requires deep knowledge of hardware design languages (HDLs). FPGAs can achieve good performance for control-flow intensive tasks with wide data paths. However, FPGA designs usually operate at frequencies (far) below 1 GHz and cannot match the performance of GPUs for highly parallel workloads, such as AI/ML workloads. Moreover, programming GPUs does not require expertise in an HDL, and compilation times are shorter compared to FPGA synthesis. Thus, it is desirable to choose the best architecture balancing high performance with development effort.

Heterogeneous computing systems combine different types of computing units into a single system. For instance, System-on-chips like AMD Zynq integrate hardened ARM CPUs with FPGA fabric. This allows designers to use the fast CPU for general computing tasks and benefit from shorter development cycles. On performance-critical sections, designers can spend more time and effort to implement accelerators on the FPGA fabric.

This research has been partially funded by the German Federal Ministry for Education and Research (BMBF) with the funding IDs 01IS21007B (SCAMP) and 01IS18081C (PANDAS) and by the German Research Foundation (DFG) within the Collaborative Research Center (CRC) 1053 MAKI. The authors would like to thank AMD Inc. for supporting their work by donations of hard- and software.

With the AI Engines (AIEs) AMD introduced for Versal devices and for selected AMD Ryzen CPUs (Ryzen AI), a developer is offered an additional option for performance improvements. As AIEs consist of VLIW-based tiles with vector units implemented as hard-cores, they operate at higher clock frequency than the FPGA fabric, closer to that of GPUs. In addition, with an array of up to 400 AIE tiles in Versal AI Core devices, the AIEs provide a vast range of opportunities for implementing operations in parallel. Furthermore, the connections between FPGA Programmable Logic (PL) and AIEs provide an aggregated performance of up to 1.56 TB/s from PL to AIEs. Since each AIE not just supports wide SIMD-style operation, but also VLIW-style parallelism, it is more flexible than a pure vector/SIMD machine. AIEs have a software-like programming model and are independent of FPGA synthesis after PLIOs are placed.

In this paper, we present TaPaSCo-AIE, which extends the existing FPGA accelerator framework TaPaSCo [1] with support for AMD/Xilinx Versal FPGAs, its AIEs and DMA streaming infrastructure. Furthermore, we also combine several new architectural features of Versal, such as the Network-on-Chip (NoC) and 100 G Ethernet, to realize scalable distributed streaming processing architectures. To demonstrate the advanced capabilities of TaPaSCo-AIE, we implement a custom neural network inference application, which we also distribute across two devices.

II. Related Work

As the models for artificial intelligence are increasing in size, many novel computer architectures have been created. A common characteristic of these architectures is high computing power through parallelism and fast, distributed memory. The tile-based architecture in Graphcore [2] facilitates a highly scalable system by splitting computing and communication into separate phases. Another approach by Cerebras [3]

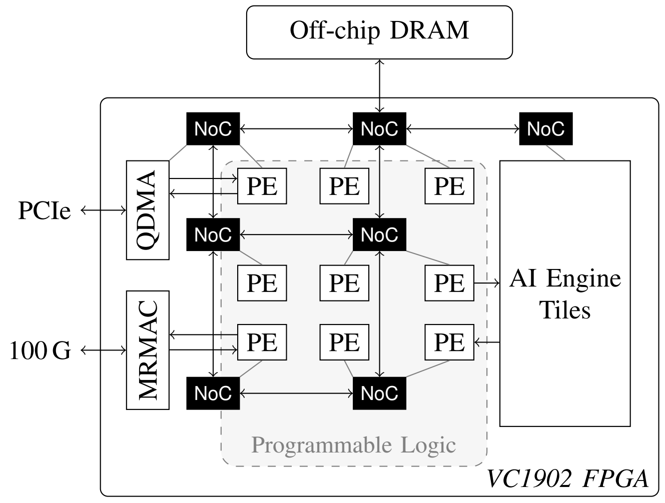


Fig. 1. TaPaSCo-AIE system overview on a Versal device.

enables scaling of compute power by improved manufacturing techniques which allow the operation on entire wafers. AMD Versal [4], which is used in this work, extends its existing FPGA fabric by incorporating highly optimized, hardened IP cores, such as AI engines. A similar approach can be observed in networking, where hardware acceleration is necessary for computations at line rate. To address this need, Intel and NVIDIA have both introduced specialized hardware units known as IPU and DPU, respectively. These units offload and accelerate network functions from a host, leading to improved performance [5], [6]. FPGAs are commonly used for network processing, e.g. in the NetFPGA framework [7] and as a network switch [8].

AIEs have already been utilized for different applications [9], [10], [11], [12], [13]. The existing work focuses on *specific* applications, while this work presents the integration of AIEs into a *general-purpose* framework and tries to reduce the effort for many future applications. In contrast to existing work, we target entire applications, including data transfers between host and device. To our knowledge, no prior work exists on using the network on the Versal architecture to scale AIE operations across distributed Versal cards.

III. TaPaSCo on Versal

TaPaSCo is a platform-independent framework to integrate FPGA accelerators into heterogeneous systems. The framework provides a tool-flow to generate the FPGA design based on a composition of Processing Elements (PEs). PEs may be provided as HLS kernels or directly written in an HDL. The required infrastructure around the PEs, such as interconnects, memory controllers, interrupt controller, or host connection, is auto-generated according to the specific platform.

In addition, a software runtime library and Linux kernel module facilitate the interaction with the accelerator. The easy-to-use C++ and Rust APIs allow passing arguments and data buffers, and launching the PE in just one function call. Once written, the application runs on all platforms. Supported platforms include various data center cards (AMD Virtex-7, UltraScale+), AWS cloud instances, and embedded Zynq-based devices (7000 and MPSoC). A plugin system allows for extensions, such as Shared Virtual Memory (SVM) [14] or 100 G Ethernet [1].

The Versal architecture introduces many new hardened units, which can be used in addition to the programmable logic. With this approach, commonly used functionality can be provided efficiently and does not need to be implemented on the PL, reducing the mapping time of design synthesis, leaving more logic available for custom applications, and potentially achieving higher clock frequencies.

An example of such a system is shown in Figure 1. This system consists of 9 PEs, of which the upper left one has a DMA streaming connection to PCIe, and the lower left one has a streaming connection via the MRMAC 100 G network module to Ethernet. The lower two PEs on the right provide data to and consume data from the AIEs, respectively. All PEs have their control interface and memory access connected via the NoC.

A. Network-on-Chip

As chip size grew in transistor count, Network-on-Chips (NoCs) became a common mechanism to connect the individual components on a chip. Even for FPGA designs, "soft" NoCs have been implemented on top of the programmable logic [15], [16]. As routing resources and clock frequencies are more limited than in custom ASIC designs, though, performance is often more limited.

In the Versal architecture, AMD provides an efficient hardened NoC as a central part of the chip. It connects the on-chip hard-cores, such as DMA engine, AIEs and memory controllers, and provides multiple ports to communicate with the programmable logic. The hard NoC provides a 128 bit wide data path at a frequency of 1 GHz [17], which enables a high throughput that would be difficult to achieve in many soft NoCs.

By using the various NoC ports, TaPaSCo enables efficient memory access to the entire address space for all PEs without additional resources or latency costs due to crossbars in the PL.

B. QDMA for PCIe Transfers

Memory transfers are among the most important aspects of heterogeneous acceleration systems. If the data transfers from the host to the PCIe-attached accelerator card are slow, even the fastest accelerator cannot achieve much system-level acceleration. Thus, high-performance DMA has always been an important aspect in TaPaSCo. The initial implementation was based on ffLink [18], and has since then been improved with a switch to *BlueDMA*, a fast DMA core written in BlueSpec System Verilog.

The original vendor-provided DMA interface, in the form of the XDMA IP block, is known to have somewhat limited performance [19]. The successor IP with better performance, namely the Queue-based Direct Memory Access (QDMA)

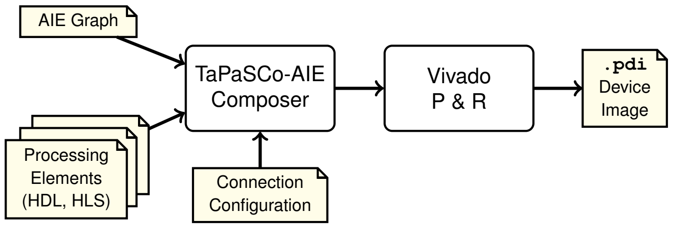


Fig. 2. TaPaSCo-AIE flow for flexible system design with AI Engines, integrated into the general-purpose Vivado platform.

block, was initially made available for the UltraScale+ generation of devices. As many cores on UltraScale+, it was realized on programmable “soft” logic.

The Versal implementation of QDMA has been refined further. First, it now is a hard-core, meaning that more programmable logic can be saved for use by the actual application, and synthesis times are reduced. In addition, the QDMA hard-core is directly connected to the hard NoC and can perform memory operations without interaction with the programmable logic.

Thus, we have integrated QDMA into TaPaSCo. In default operation mode, the QDMA expects host software to place descriptors for DMA transfers in host memory. However, we use the QDMA in bypass mode, meaning that a custom hardware module generates the descriptors on the FPGA and feeds them directly into the QDMA. This saves PCIe bandwidth, especially for larger transfers, since the host software writes the control registers of the descriptor generator only once per transfer, and the QDMA does not need to fetch any descriptor from host memory. Furthermore, we reduce latency since the descriptors are already on the FPGA, saving the PCIe turnaround time for fetching the first descriptor from host memory. The interface of the descriptor generator is compatible with the existing BlueDMA infrastructure, thus keeping changes to the runtime and kernel module at a minimum. Additionally, the TaPaSCo interrupt controller had to be adapted to the interrupt interface of the QDMA core, as interrupts are now also managed by the hard-core.

C. Streaming DMA

Moving data from the host to off-chip memory on the device, and vice versa, is often a major performance bottleneck in hardware-accelerated applications. Usually, all required data is moved from the host to off-chip memory on the PCIe card before the accelerator is launched and operates on the data. After it has finished, results are copied back to the host memory. However, a streaming accelerator (as most neural network accelerators are) does not access its input data randomly, and, thus, does not need to have all data available in off-chip memory. Hence, we use the QDMA streaming mode and enable direct streaming DMA operations in TaPaSCo. Instead of writing input data to device memory prior to launching the PE on the FPGA, we stream the data directly from the host memory using the QDMA into the PE during runtime. In the same way, we write the PE output stream directly back to the host memory, without involving device memory operations at any point. However, TaPaSCo does retain the option to involve off-chip-memory for other kinds of accelerators (e.g., for a database of graph operations) that do require random access to the input and/or output data.

Descriptors for streaming QDMA operations are generated directly in hardware, similar to the process for standard memory writes. We use a separate queue for streaming in order to enable both streaming and ordinary DMA operations in parallel. This allows a user to combine streaming and non-streaming-based PEs in one hardware design. The TaPaSCo runtime creates one thread per stream to supply the PE with data as required, thus avoiding blocking other concurrently running operations on the host side.

D. AI Engine

AIEs in the Versal architecture are a cluster of VLIW compute elements called *tiles*. Each tile provides a scalar and SIMD vector processor each. Two 32 bit streams to a streaming interconnect provide fast and high-bandwidth data transfers to all tiles. Moreover, every tile has a 384 bit cascade stream interface to one neighboring tile. In addition, neighboring tiles have access each other’s tile-local memories and exchange data using ping-pong buffers, providing higher bandwidth than streaming interconnections.

The entire AIE cluster can communicate over multiple PLIO connections with the PL FPGA fabric and access the off-chip-memory through the GMIO interface via the NoC.

The recommended flow for integrating ML applications with the AIEs is through the AMD Vitis AI development platform. However, this platform is only capable of implementing a deep learning processing unit (DPU) on the AIEs. In contrast, the Vitis platform and TaPaSCo-AIE allow developers to create custom AIE graphs, enabling a wide range of applications beyond typical AI workloads. The AIE graph is composed of kernels that run on the AIE tiles. These kernels are written in C++ using the custom AIE SDK. As shown in Figure 2, the TaPaSCo-AIE toolflow is solely based on *Vivado*. In addition to the AIE graph and PL PEs, the user may provide the mapping between PE streaming ports and PLIOs to the TaPaSCo-AIE composer.

E. 100 G Ethernet

For the Versal architecture, AMD includes a hard-core Multirate Ethernet MAC (MRMAC) for 100 G Ethernet [20]. As the name suggests, the core can be configured to operate at different speeds. Here, we only consider the mode for 100 G, as current data centers often employ 100 G (or even higher) network speeds.

The user interface of the core can be configured to two different data widths. A low-latency data path, which runs at the same frequency as the MRMAC core itself, provides 256 bit at 644.531 MHz. When the datapath can only operate at a lower frequency, a wider interface is provided to handle the bandwidth at the slower clock, namely having 384 bit at 390.625 MHz.

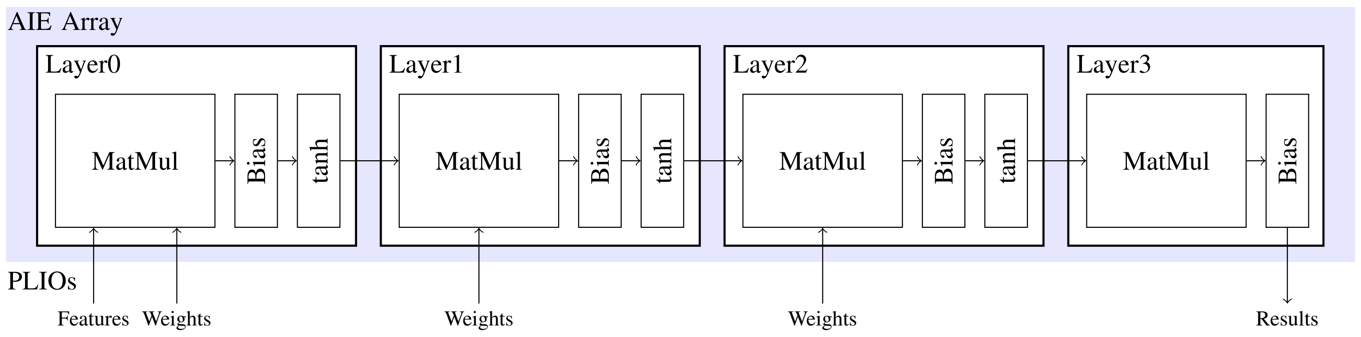


Fig. 3. AI engine graph. Input features and weights for Layers 0 through 2 are streamed into the matrix multiplication graph over PLIOs from the programmable logic. Weights in the final matrix multiplication are kept in AIE tile SRAM. Results are streamed out from Layer 3 over PLIO.

For small 192 B frames, we observed a latency with a loop-back 100 G DAC in low-latency 256 bit and low frequency 384 bit mode of 189 ns and 220 ns, respectively, compared to 298 ns on previous generation UltraScale+ (US+) devices. This increases to 295 ns and 328 ns, respectively, for Ethernet frames of 1500 B (406 ns on US+). Overall, the MRMAC on Versal has a significantly lower latency compared to US+.

In TaPaSCo-AIE, the networking feature connects AXI stream interfaces of PEs to the MRMAC, making it easy to implement accelerators with Ethernet connectivity.

IV. Case study: Neural Network

To demonstrate and evaluate TaPaSCo-AIE, we examine a simple feed-forward neural network for regression. We compare software-only approaches on CPU and GPU to an AIE-based accelerator, which we describe in the following. Our evaluation results are given in Section V. Note that we do *not* aim to realize the most optimal NN, we strictly focus on using it as a real-world benchmark for the different system architectures.

The input dataset is randomly generated and consists of samples with 64 features. Each input sample is mapped to a single result value. The network consists of three fully connected hidden layers with 128, 64, and 64 units, respectively, each applying a `tanh` activation function. The regression value is obtained after computing the output layer with one output neuron without any activation. The network structure is shown in Figure 3.

We leverage the AIEs for the compute-intense tasks, namely matrix multiplication, bias addition, and activation function calculation. Splitting these tasks into several kernels and subgraphs makes it possible to pipeline the entire computation. We use the PL for input and output data handling. One PL kernel is responsible for streaming the matrix weights for the first three layers into the AIEs over PLIOs. A second PL kernel handles the feature input and result output streams.

We designed the accelerator to perform inference with a batch size of 32 samples. Thus, the input feature matrix consists of 32 rows and 64 columns, resulting in the matrix multiplications in Table I. By instantiating the graph two times on the AIEs, we can always process two batches in parallel.

TABLE I
Per-layer matrix multiplication dimensions.

Layer	Feature Matrix Size	Weight Mat. Size	Output Mat. Size
0	32×64	64×128	32×128
1	32×128	128×64	32×64
2	32×64	64×64	32×64
3	32×64	64×1	32×1

The following sections elaborate on the AI engine graph's implementation details.

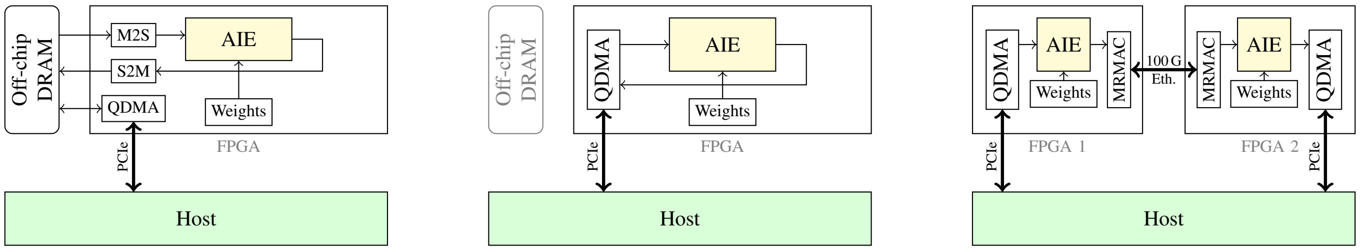
A. Weights and Bias

Except for the output layer, every layer is implemented by using the Vitis DSP Library matrix multiplication graph [21]. This graph allows us to parallelize parts of the matrix multiplication over several AIE tiles without significant effort. When not using automatic insertion of tiling kernels, it requires that the first input matrix arrives at the graph using a four-by-four tiling for floating-point input values. The second input matrix and the output matrix each cohere to a four-by-two tiling. These tiling constraints require us to add some data re-arrangement logic inside our hand-written bias addition kernels.

The computation's data parallel nature allows instantiating matrix multiplication, bias addition, and activation function several times per layer. Moreover, each matrix multiplication uses four cascaded AIE tiles, resulting in intra-layer pipelining and increasing throughput further. In addition to the input features, the Vitis DSP Library requires streaming the matrix multiplication weights through PLIOs into the AIE. Due to the limited number of available PLIOs, we can only instantiate the matrix multiplication eight, four, and four times for the first three layers, respectively. This results in using a total of 68 out of 78 input PLIOs.

The final layer stores its weights in SRAM local to the used AIE tiles and streams the output from its bias kernel to the PL through a further 128 bit PLIO.

For the first three layers, the output values of bias kernels are fed into a hand-written `tanh` implementation on the AI engines, which we explain in the next section.



(a) Memory-mapped (XRT / T-MM) (b) Streaming (T-ST) (c) Streaming and Network (T-ST-N)

Fig. 4. System architectures for the three different evaluation setups, (a) memory-mapped architecture of XRT and TaPaSCo, (b) streaming architecture of TaPaSCo-AIE, (c) streaming and network architecture of TaPaSCo-AIE with two FPGAs.

B. Tanh Activations

Since the AI engines do not have built-in support of the $\tanh$ function, but also do not support the exponential function natively, we decided to implement an approximation kernel manually on the AIE array. The corresponding function is shown in Equation (1).

$$\tanh(x) \approx \min\left(\max\left(x\left(\frac{n_0x^2+n_1}{x^2+d_0}+1\right),-1\right),1\right) \quad (1)$$

Due to the asymptotic behavior of $\tanh$, we simply cut values outside the range $[-1,1]$. A rational approximation is used for the function itself with the parameters computed based on the Remez algorithm. The approximation achieves an absolute error below 2.8×10^{-3} while requiring only seven operations [22]. All operations are vectorized to compute eight values in parallel, except the division, for which only a scalar floating point unit is available. More accurate, but slower approximation functions would be feasible as well if desired, since the profiling data of the AIE hardware emulation in Vitis shows that the $\tanh$ kernels are not time-critical in comparison to the matrix multiplications. With Vitis HLS, it is also easily possible to implement $\tanh$ as a kernel in the PL. Again, as this function is not time-critical, we do not expect any improvements in performance. On the contrary, the data transfers between AIE and PL add additional load on the interconnect between the AIE tiles and require additional PLIOs.

V. Evaluation

Our test system is based on an AMD Epyc 7443P 24-core CPU, combined with 256 GB memory, and running Rocky Linux 8. We evaluate our flow targeting the AMD VCK5000 accelerator card, which carries a VC1902 Versal AI Core device and provides two QSFP28 network ports. Two VCK5000 are attached by PCIe 4.0 x8 to our host CPU, which is the maximum supported by this FPGA generation, and connected via a direct 100 G Ethernet link to each other. GPU experiments are run on an NVIDIA A100.

As tools, we use Vitis 2022.2 for XRT, which is the latest supported version for the VCK5000 card, and Vivado 2023.1 for our TaPaSCo designs.

All our results are averaged over 1,000 runs, and *include* runtime of all necessary data transfers over PCIe. We vary the overall number of samples in our test points. The AIE graph processes data in batches of 32 samples with 64 single-precision float features each, equalling to 8 kB of input data.

A. Evaluation Setups

As a baseline, we use a NumPy-based implementation. We also evaluate our neural network compiled with Keras on both CPU and GPU. To evaluate our AIE-based implementations, we compare three different variants shown in Figure 4. All variants use the AIE graph described in detail in Section IV.

PCIe-based TaPaSCo designs act as an accelerator card for a host system, similar to the commonly used XRT. Figure 4a shows the setup of our neural network implementation on the VCK5000 accelerator card using TaPaSCo (denoted as T-MM in the following) or XRT, containing the FPGA with AIE and off-chip DRAM memory, connected via PCIe to a host.

In both frameworks, it is required to copy data from the host to the off-chip memory of the FPGA. From there, data streaming kernels in the PL (M2S and S2M) can convert between this memory-mapped off-chip memory and the streams required to interface with the AIE. The result data is copied back to the host memory afterward.

The new TaPaSCo streaming feature introduced in Section III-C, allows us to stream data directly from the DMA engine into the PL FPGA fabric and vice versa. As a result, we can avoid intermediate copies to the off-chip memory, as shown in Figure 4b. A PL kernel splits the incoming stream and forwards feature data directly to the AIEs. We denote this variant as T-ST.

In our third variant, we utilize the 100 G feature of TaPaSCo to distribute the AIE graph over two FPGAs, as shown in Figure 4c. Using a network connection as a fast link with low latency to transfer data between two FPGAs allows realizing large applications that would not fit into a *single* FPGA or require more AIE tiles than available on a single device. To demonstrate this capability, we partition the computation of our neural network between Layer1 and Layer2, assigning each FPGA with the calculation of two layers. We again use DMA streaming to transfer the input data and results via PCIe between the host and the respective FPGA. However, the intermediate results are streamed over 100 G Ethernet from the first to the second FPGA. This variant is denoted as T-ST-N in the following.

TABLE II
ABSOLUTE MEAN RUNTIMES OF THE NEURAL NETWORK BENCHMARK MEASURED ON THE HOST, DEPENDENT ON THE NUMBER OF INPUT SAMPLES

Samples	2^6	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}	2^{22}
NumPy (CPU)	0.137 ms	0.579 ms	3.100 ms	12.04 ms	42.91 ms	146.8 ms	455.2 ms	1.476 s	5.613 s
Keras (CPU)	0.240 ms	0.370 ms	0.606 ms	1.36 ms	2.51 ms	15.3 ms	38.2 ms	0.122 s	0.465 s
Keras (GPU)	0.405 ms	0.394 ms	0.404 ms	0.56 ms	0.83 ms	3.1 ms	27.2 ms	0.106 s	0.423 s
XRT (FPGA)	0.786 ms	0.888 ms	0.917 ms	1.29 ms	2.77 ms	11.0 ms	27.0 ms	0.497 s	1.952 s
T-MM (FPGA)	0.224 ms	0.237 ms	0.284 ms	0.60 ms	1.72 ms	6.2 ms	24.4 ms	0.096 s	0.384 s
T-ST (FPGA)	0.138 ms	0.154 ms	0.190 ms	0.36 ms	1.02 ms	3.6 ms	14.1 ms	0.056 s	0.223 s
T-ST-N (2 FPGAs)	-	0.156 ms	0.229 ms	0.30 ms	0.70 ms	2.4 ms	9.7 ms	0.039 s	0.156 s

Partitioning the neural network on two FPGAs allows us to instantiate our AIE graph four times in total and process always four batches in parallel, while the AIE tiles of a single FPGA can only accommodate two graph instances.

B. Results

Figure 5 shows the speedup of XRT, T-MM, T-ST, T-ST-N, and Keras (CPU and GPU), compared to the NumPy software baseline. The absolute mean runtimes are listed in Table II.

For the smallest number of samples (2^6), the NumPy baseline is the fastest implementation with 0.14 ms. However, as soon as we increase the number of samples, all other variants achieve speedups. All variants using TaPaSCo show the highest speedup for 2^{15} samples with 25x (T-MM), 43x (T-ST), and 67x (T-ST-N), respectively. Beyond this point, runtimes of these three variants grow proportionally to the number of samples. However, the NumPy baseline increasingly benefits from multi-threaded execution and scales *superlinearly*, leading to a *decrease* in the speedups achievable using other techniques. Nonetheless, T-ST and T-ST-N process 2^{22} samples (1 GB input data) in 0.22 s (25x) and 0.16 s (40x) compared to 0.46 s using Keras, and 5.61 s using NumPy. In general, T-ST is about 1.7x faster than T-MM, since data is now streamed directly from host memory into the PL and AIEs, without being written to off-chip memory first. Our distributed T-ST-N achieves additional 40–60 % speedup over T-ST. However, T-MM is still faster than Keras (CPU) for all numbers of samples.

XRT achieves its highest speedup over the NumPy baseline for 2^{18} samples with 17x. This is the only test point where XRT achieves comparable runtimes to T-MM. However, XRT's performance drops significantly for larger numbers of samples, with a slowdown of 2.5x - 3x compared to Keras on the CPU.

Running inference with Keras on the GPU achieves the best performance between 2^{13} - 2^{16} samples with up to 52x speedup. This is faster than both single-FPGA TaPaSCo variants on a single FPGA, but cannot beat our distributed T-ST-N implementation. For the remaining test points, GPU execution is slightly slower than T-MM. In addition, power consumption of T-ST is less than half of the GPU's consumption with 68 W (Xilinx Power Design Manager estimation) compared to 155 W on the GPU.

In real-time inference applications, it is important to have a low "jitter" of execution times. We thus examined the relative standard deviation of our execution time measurements. T-ST shows higher relative standard deviations for a small number

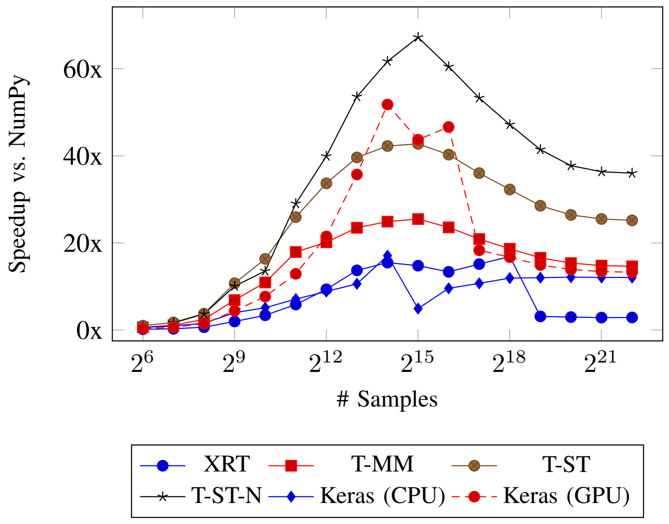


Fig. 5. Speedup compared to NumPy software execution baseline for the inference of the neural network, evaluated on different system architectures. Geomean across 1,000 executions.

of samples, with a maximum of 18 % for 2^7 samples. The higher variance in T-ST can be attributed to the spawning and rejoining of the DMA handling threads. However, T-ST achieves the lowest relative standard deviations of all variants for sample sizes beyond 2^{15}, with a minimum of 0.02 % for the largest number of samples. In T-ST, the variance is only influenced by the fill and drain phases of the streams, since the remaining data in between is streamed seamlessly into the PE. These phases only constitute a small fraction of the total execution time when processing large amounts of data, leading to highly deterministic runtimes. T-ST-N has slightly higher standard deviations in the measurements than T-ST.

VI. Conclusion

We have presented the TaPaSCo-AIE programming framework for Versal devices, which supports several novel device features, most notably the heterogeneous execution across AIEs and FPGA logic. As an array of software-programmable processing units, the AIEs can be programmed without needing expert knowledge in hardware description languages and computer architecture.

TaPaSCo-AIE enables the efficient use of the streaming-based architecture of the AIEs by actually performing streaming DMA, avoiding data transfers buffered in off-chip memory.

For our neural network case study, we achieve a speedup of up to 67x compared to a CPU implementation and up to 2.9x compared to GPUs while maintaining low variance. Due to its streaming architecture, TaPaSCo-AIE has lower latency and, as a result, reduced overhead for hardware accelerators compared to traditional memory-mapped transfers. In combination with 100 G network, TaPaSCo-AIE allows using the AIEs as in-network accelerators and scaling across multiple cards without the PCIe bus becoming the bottleneck.

TaPaSCo-AIE will be made publicly available as an extension to TaPaSCo.

References

[1] C. Heinz, J. Hofmann, J. Korinth, L. Sommer, L. Weber, and A. Koch, "The TaPaSCo Open-Source Toolflow," *Journal of Signal Processing Systems*, May 2021. [Online]. Available: https://doi.org/10.1007/s11265-021-01640-8

[2] S. Knowles, "Graphcore," in *2021 IEEE Hot Chips 33 Symposium (HCS)*, 2021, pp. 1–25.

[3] G. Lauterbach, "The Path to Successful Wafer-Scale Integration: The Cerebras Story," *IEEE Micro*, vol. 41, no. 6, pp. 52–57, 2021.

[4] B. Gaide, D. Gaitonde, C. Ravishankar, and T. Bauer, "Xilinx Adaptive Compute Acceleration Platform: Versal™ Architecture," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 84–93. [Online]. Available: https://doi.org/10.1145/3289602.3293906

[5] B. Burres, D. Daly, M. Debbage, E. Louzoun, C. Severns-Williams, N. Sundar, N. Turbovich, B. Wolford, and Y. Li, "Intel's Hyperscale-Ready Infrastructure Processing Unit (IPU)," in *2021 IEEE Hot Chips 33 Symposium (HCS)*, 2021, pp. 1–16.

[6] I. Burstein, "Nvidia Data Center Processing Unit (DPU) Architecture," in *2021 IEEE Hot Chips 33 Symposium (HCS)*, 2021, pp. 1–20.

[7] N. Zilberman, Y. Audzevich, G. A. Covington, and A. W. Moore, "NetFPGA SUME: Toward 100 Gbps as Research Commodity," *IEEE Micro*, vol. 34, no. 5, pp. 32–41, 2014.

[8] P. Papaphilippou, J. Meng, and W. Luk, "High-Performance FPGA Network Switch Architecture," in *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 76–85. [Online]. Available: https://doi.org/10.1145/3373087.3375299

[9] N. Brown, "Exploring the Versal AI Engines for Accelerating Stencil-Based Atmospheric Advection Simulation," in *Proceedings of the 2023 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 91–97. [Online]. Available: https://doi.org/10.1145/3543622.3573047

[10] X. Jia, Y. Zhang, G. Liu, X. Yang, T. Zhang, J. Zheng, D. Xu, H. Wang, R. Zheng, S. Pareek, L. Tian, D. Xie, H. Luo, and Y. Shan, "XVDPU: A High Performance CNN Accelerator on the Versal Platform Powered by the AI Engine," in *2022 32nd International Conference on Field-Programmable Logic and Applications (FPL)*, 2022, pp. 01–09.

[11] C. Zhang, T. Geng, A. Guo, J. Tian, M. Herbordt, A. Li, and D. Tao, "H-GCN: A Graph Convolutional Network Accelerator on Versal ACAP Architecture," in *2022 32nd International Conference on Field-Programmable Logic and Applications (FPL)*, 2022, pp. 200–208.

[12] J. Zhuang, J. Lau, H. Ye, Z. Yang, Y. Du, J. Lo, K. Denolf, S. Neuendorffer, A. Jones, J. Hu, D. Chen, J. Cong, and P. Zhou, "CHARM: Composing Heterogeneous AcceleRators for Matrix Multiply on Versal ACAP Architecture," in *Proceedings of the 2023 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 153–164. [Online]. Available: https://doi.org/10.1145/3543622.3573210

[13] E. Taka, A. Arora, K. C. Wu, and D. Marculescu, "MaxEVA: Maximizing the Efficiency of Matrix Multiplication on Versal AI Engine," in *22nd International Conference on Field Programmable Technology (FPT)*. IEEE, 2023.

[14] T. Kalkhof and A. Koch, "Efficient Physical Page Migrations in Shared Virtual Memory Reconfigurable Computing Systems," in *2021 International Conference on Field-Programmable Technology (ICFPT)*, 2021.

[15] N. Kapre and J. Gray, "Hoplite: Building austere overlay NoCs for FPGAs," in *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, 2015, pp. 1–8.

[16] J. Gray, "GRVI Phalanx: A Massively Parallel RISC-V FPGA Accelerator Accelerator," in *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2016, pp. 17–20.

[17] AMD, "Versal Adaptive SoC Programmable Network on Chip and Integrated Memory Controller 1.0 LogiCORE IP Product Guide (PG313)," 2023. [Online]. Available: https://docs.xilinx.com/r/en-US/pg313-network-on-chip

[18] D. de la Chevallerie, J. Korinth, and A. Koch, "ffLink: A Lightweight High-Performance Open-Source PCI Express Gen3 Interface for Reconfigurable Accelerators," *SIGARCH Comput. Archit. News*, vol. 43, no. 4, p. 34–39, apr 2016. [Online]. Available: https://doi.org/10.1145/2927964.2927971

[19] T. Yokono, Y. Yamabe, K. Tanaka, Y. Arikawa, and T. Ishizaki, "FPGA-based Accelerators System with Low Latency Autonomous DMA Engine," in *2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2022, pp. 1–1.

[20] AMD, "Versal Devices Integrated 100G Multirate Ethernet MAC Subsystem Product Guide (PG314)," 2023. [Online]. Available: https://docs.xilinx.com/r/en-US/pg314-versal-mrmac

[21] Xilinx Inc., "Vitis DSP Library," https://docs.xilinx.com/r/en-US/Vitis_Libraries/dsp/index.html.

[22] N. Juffa, "Best non-trigonometric floating point approximation of tanh(x) in 10 instructions or less," https://stackoverflow.com/a/73781591, 2022.