ArKANe: Accelerating Kolmogorov-Arnold Networks on Reconfigurable Spatial Architectures

Yanze Wu, *Member, IEEE,* Md Tanvir Arafin, *Member, IEEE*

***Abstract*—Kolmogorov-Arnold networks (KAN) are promising for explainable machine learning on embedded devices. Although KANs require fewer model parameters and are smaller than traditional deep neural networks, their applicability is limited by the complexity of the B-spline functions used at every neuron. Hence, this work presents acceleration techniques for KANs on dataflow-processing-based accelerators implemented in AMD-Xilinx Versal ASoCs. Our experiments on the VCK-190 devices demonstrate that for large batch sizes, the average B-spline computation on systolic and wavefront architectures outperforms standard CPU implementation by 11x and 60x. On the other hand, wavefront accelerators demonstrate more than 3x improvement over GPUs (RTX 3090) in terms of energy consumption. Thus, this work opens new opportunities for spatial accelerators in non-conventional machine learning algorithms on embedded systems. This work's code and experimental artifacts are available at https://github.com/SPIRE-GMU/SPIRE-ARKANE.**

***Index Terms*—Kolmogorov-Arnold network, systolic architecture, wavefront array design, reconfigurable spatial accelerators.**

I. Introduction

Recent advances in the *deep* Kolmogorov-Arnold Network (KAN) have poised it as a strong contender to replace traditional deep neural networks (DNN) for explainable machine learning (ML) tasks in symbolic regression, solving partial differential equations, and continual learning [1]. KAN models are also significantly smaller and can provide better accuracy, efficient training, and fast inference. Thus, KAN-based learning solutions can usher in a new paradigm of explainable machine intelligence on edge devices.

Unfortunately, from an architectural perspective, KANs differ significantly from DNNs and thus suffer from slow training on CPUs and GPUs. Although the KAN model size is small, each neuron in the KAN architecture requires recursive basis-spline (B-spline) computation, which can become untractable for large networks. In addition, traditional GPU implementation of KAN is also resource-intensive and, therefore, unsuitable for embedded systems. Hence, realizing KAN on embedded devices remains an open problem.

Interestingly, tremendous progress has been made in the computation capabilities of embedded devices in recent years. The current generation of spatial computing architectures, *e.g.*, Tenstorrent's GreySkull processors [2], and AMD's adaptive system-on-chip (ASoC) [3] contain multitudes of processing solutions on a single chip. Such SoCs feature robust processing cores, programmable logic resources, and processing elements (PEs) connected via high bandwidth channels. These embedded architectures create an opportunity to investigate the novel acceleration techniques for KAN on embedded SoCs. This will, in turn, enable explainable machine learning, where the underlying physical relationship between model and data can be represented in a closed-form functional approximation at the edge devices [1].

However, the *compute-bound* B-spline computation presents a critical bottleneck for realizing KAN on such ASoCs. Efficient B-spline computations unlock other promising applications in edge IoT platforms. For example, [4] demonstrates the application of B-spline for path smoothing, [5] uses B-spline for inertial fusion, and [6] uses them for safe navigation in mobile robots. Thus, fundamental acceleration improvement of B-spline-based computation has a broader applicability on edge AI and IoT solutions. Hence, we present novel systolic and wavefront designs for accelerating B-spline-based computation developed on a spatial computing architecture. Our contributions are:

1. We implement the standard KAN architecture for AMD-Xilinx Versal ASoCs and investigate the acceleration bottlenecks of the KAN implementation.
2. We design novel systolic and wavefront architectures realizable on Versal ASoCs to accelerate the B-spline computation and integrate it with the overall KAN architecture. We implement our design on the Versal VCK-190 platform and compare them with equivalent CPU and GPU implementations.
3. We publish the code and computational artifacts at https://github.com/SPIRE-GMU/SPIRE-ARKANE.

II. Preliminaries

A. Kolmogorov-Arnold Network

According to the universal approximation theorem, any function M can be approximated within distance ϵ with a sufficiently deep neural network [7]. The Kolmogorov-Arnold representation theorem [8] presents a simpler case where any continuous multivariate function F can be represented with an accumulation of univariate functions Φ. This has inspired the *recent* design of the deep Kolmogorov-Arnold network where any multivariate function F is decomposed into a deep network of univariate functions as [1]:

$$F(x) = \sum_{m_{L-1}=1}^{n_{L-1}} \Phi_{L-1,m_L,m_{L-1}}\Big(\sum_{m_{L-2}=1}^{n_{L-2}} \cdots \Big(\sum_{m_0=1}^{n_0} \Phi_{0,m_1,m_0}(x_{m_0})\Big)\Big) \tag{1}$$

where n_m is the number of nodes in m^{th} layer, x_{m_0} is the input, and $\Phi(x)$ is the learnable univariate function given by,

$$\Phi(x) = wb(x) + \sum_{i=0}^{k+G-1} c_i B_i^k(x) \tag{2}$$

$$b(x) = silu(x) = x/(1+e^{-x}) \tag{3}$$

The authors are with the Cyber Security Engineering Department, George Mason University, Fairfax, VA 22030.

This material is based upon work partially supported by the National Science Foundation under grant #2245156 and Commonwealth Cyber Initiative grant N-3Q24-003. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Manuscript received January 1, 2025; revised April 11, 2025.

where w and c_i are learnable parameters, $b()$ is the activation function, and B is the B-spline function, *i.e.*,

$$B_i^k(x) = \frac{x - x_i}{x_{i+k-1} - x_i} B_i^{k-1}(x) + \frac{x_{i+k} - x}{x_{i+k} - x_{i+1}} B_{i+1}^{k-1}(x) \quad (4)$$

$$\text{where, } B_i^0(x) = \begin{cases} 1, & x_i < x < x_{i+1} \\ 0, & otherwise \end{cases} \quad (5)$$

Subscripts i and k denote the grid points and degree of the B-spline function. Fig. 1 depicts a 2-layer KAN architecture that takes a 2-dimensional input, uses $2n$ learnable functions ($\phi_{1,0}$, functional $\ldots$, $\phi_{n,0}$, $\phi_{n,1}$) in the first layer, n learnable functions in the second layer, and produces an n dimensional output.

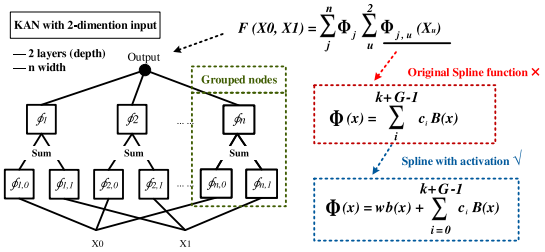

Fig. 1. Overview of Kolmogorov-Arnold Network. The B-spline functions are grouped with green boxes. The learnable univariate function contains an activation function $b(x)$. w and c_i are learnable parameters.

From the Eqns. 1 – 4, it is evident that the computation for KAN is dominated by evaluating the B-spline functions at different grid points. However, B-spline functions are compute-bound, *i.e.*, the number of compute operations is larger than the number of input/output operations. A standard method to speed up such compute-bound tasks is to decompose them into sub-operations and distribute them to different processing elements. To realize such a design, we investigate the current generation of reconfigurable spatial accelerators.

B. Reconfigurable Spatial/Dataflow Architecture

There has been a paradigm shift in the accelerator design for machine learning applications. Although GPUs can handle large ML models' training and inference processes, they suffer from data-movement bottlenecks, inefficient power consumption, and issues originating from the fixed SIMD architecture. Embedded accelerators such as Tenstorrent's GraySkull processors [9] and AMD's Versal AI Engines [3] are emerging as promising low-power reconfigurable platforms that can enable novel ML accelerators on embedded SoCs. Interestingly, most of these emerging devices share a typical spatial architecture pattern where a large number of simple processing elements are connected in a two-dimensional grid.

This work is developed on AMD-Xilinx's new spatial computing architecture, the adaptive intelligent engine (AIE). An outline of AIE-supported Versal devices is shown in Fig.2. The platform comprises processing systems (PS), AI engines, and programmable logic (PL). The PS side organizes various tasks, prepares data, and drives the AIEs and other peripheral devices. The AI engine is an array of up to 400 AIE tiles connected with AXI buses. Each AIE tile can be considered as an independent processing element (PE) with two AXI channels, ensuring it communicates simultaneously with the other two AIE tiles. Each AIE tile is a vector processor supporting

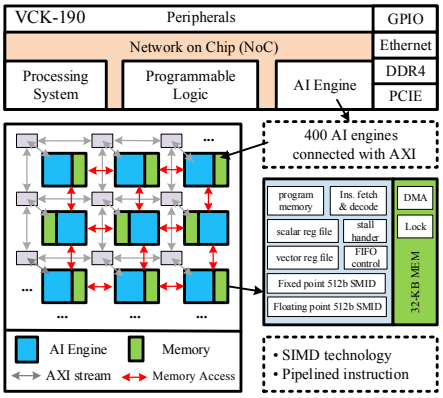

Fig. 2. Overview of Versal VCK-190 platform. The platform comprises processing systems (PS), AI engine tiles or processing elements (PEs), and programmable logic (PL) elements. The AI engine has up to 400 RISC processors connected via AXI streams.

SIMD parallelism. In addition, each AIE tile can operate independently, thus offering tile-level spatial parallelism for complex algorithms. As discussed in the following sections, such temporal and spatial parallelism can be leveraged to accelerate KAN architectures.

III. Accelerating B-spline Computation on a Spatial Architecture

Since the B-spline calculations dominate the KAN computation, we first analyze the B-spline dependence graph (DG) to understand the dataflow constraints. Then, we derive the signal-flow graph (SFG) to orchestrate this calculation onto systolic and wavefront architectures.

A. Dependence Graph Generation

A dependence graph is a single assignment representation that illustrates the dependence of the computations in temporal and spatial domains [10]. For example, we can express the k and i indices of Eqn. 4 as time and space index, respectively. Then, we can rewrite Eqn. 4 as:

$$B_i^k(x) = a_{i,0}^{k-1} B_i^{k-1}(x) + a_{i,1}^{k-1} B_{i+1}^{k-1}(x) \quad (6)$$

where $a_{i,0}^{k-1} = \frac{x - x_i}{x_{i+k-1} - x_i}$ and $a_{i,1}^{k-1} = \frac{x_{i+k} - x}{x_{i+k} - x_{i+1}}$. From this equation, we see the dependencies of these coefficients on the time index k. We unroll the B-spline function using this temporal dependency into a dependence graph, as shown in Figure 3(a). This graph depicts the computation dependency in the temporal domain, *i.e.,* over k. Note that the nodes with the same spatial index i at a given time-step k are independent, demonstrating the acceleration opportunities of B-spline functions over spatial dataflow architectures.

B. Systolic Architecture Design

One straightforward design method for determining a spatial architecture is to designate one PE for each node in the DG. However, this will require a large number of PEs and lead to inefficient utilization since each PE can be active only for a small fraction of the computation time. To improve PE utilization, it is desirable to map the nodes of DG into a smaller number of PEs.

Hence, we map the DG to an intermediate expression, *i.e.*, a signal-flow graph using linear projection. A linear mapping in

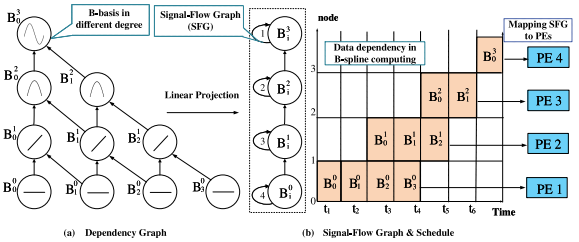

Fig. 3. (a) Dependence graph of a cubic B-spline function, its (b) linear projection and scheduling scheme. DG indicates relationship between basis with various degrees, *e.g.*, B^0, B^1, B^2, and B^3 denote piecewise constant, piecewise linear function, quadratic function, and cubic function respectively. The DG projects nodes along i-index and obtains a chain where each node calculates the basis with the same degree of k. The number inside the loop represents the computations for each basis.

i-axis direction converts the 2-D index space of the B-spline function into a 1-D process space, *i.e.*, the chain $B_i^3 \leftarrow B_i^2 \leftarrow B_i^1 \leftarrow B_i^0$. This significantly improves the PE utilization in a pipelined spatial design, as shown in Fig.3(b).

To design a systolic architecture on a spatial accelerator, we map each node of the projected SFG to a PE. This leads to a design depicted in Fig. 4. The grid points and input values are propagated horizontally, and the PEs implement the B-basis functions vertically. The PEs In and Out work as I/O boundaries when communicating with the memory. Since the systolic array synchronizes data with synchronization beats, it does not have deadlock or memory overflow issues.

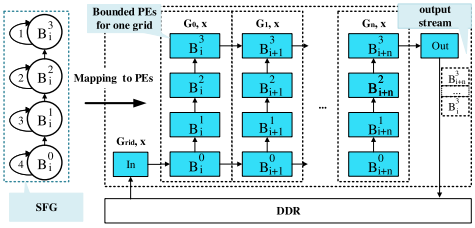

Fig. 4. Mapping SFG to design a systolic architecture. The rectangles are PEs on the chip. The processor array receives data from the DDR memory and sends results back to memory after computation.

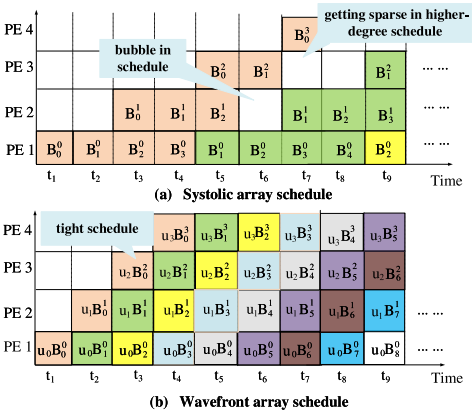

Fig. 5. Task scheduling of systolic and wavefront array for B-spline computation. Different colors represent the evaluation of different B-spline functions in the same grid.

C. Wavefront Array Design

The systolic array features highly pipelined multiprocessing and a continuous data flow between PEs. However, we find that the PEs that process higher degree B-basis have a lower throughput, *e.g.*, the last PE output only B_0^3 as shown in Figure 5 (a). To obtain a design with high throughput, we reevaluate the SFG by projecting the nodes along the k-index as shown in Fig. 6. This perspective retains the spatial index but eliminates the temporal index. Thus, the first term in the RHS of Eqn 6, can be expanded as,

$$B_i^{k-1}(x) = a_{i,0}^{k-2} B_i^{k-2}(x) + a_{i,1}^{k-2} B_{i+1}^{k-2}(x) \tag{7}$$

The second term can also be written in this chained format. Using this simplification, cubic B-spline computation can be written as a linear combination of 0-order B-basis, *i.e.*,

$$B_i^3(x) = u_0 B_i^0(x) + u_1 B_{i+1}^0(x) + u_2 B_{i+2}^0(x) + u_3 B_{i+3}^0(x) \tag{8}$$

where coefficients u_i can be pre-obtained when unrolling the items, such as $u_0 = a_{0,0}^2 \cdot a_{0,0}^1 \cdot a_{0,0}^0$. This expansion leads to a data-driven wavefront architecture where each PE continuously processes the corresponding items and propagates the result to the neighboring nodes. Each PE relaxes strict timing requirements and acts as a secondary data source by storing results in local memory. In essence, the PE in the wavefront architecture is activated immediately when the data is available from the front PE, as shown in Fig. 5(b). Direct memory access FIFO is set between the processors to avoid deadlock or overflow.

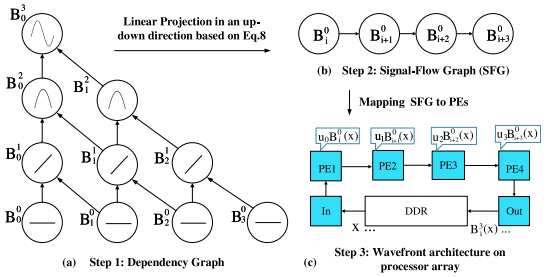

Fig. 6. Eq. 8 is projected onto PEs in a spatial perspective for designing a wavefront array. Each PE processes part of the B-spline and accumulates the result at the end.

IV. Experimental Results & Discussions

A. Experimental Setup

The experiments for this work are conducted on an x86 CPU, a GPU, and a Versal ASoC. Table II lists the configuration of the experimental setup.

B. Results & Discussions

First, we implement a three-layer KAN architecture using C in the ARM processor in VCK-190. Without any optimization, it takes 1155.9 seconds and 11.9 Watts to train this network on the Versal VCK-190 platform. From our experiments, it is evident that the B-spline function is one of the critical bottlenecks for KAN training and inference applications. Therefore, next, we focus on accelerating the B-spline function. We compare the standard implementation of the cubic B-spline function on CPU and Versal ASoC. Table III lists the compute time (with $i = 5$ and $k = 3$ for Equation 4) for the recursive De Boor algorithm, its iterative counterpart, and the loop-parallelized iterative implementation.

From Table III, it is clear that loop-parallelized iterative implementation of B-splines on AI engines (i.e., PEs of VCK-190) provides more than 10× speedup over its x86 version. These results demonstrate the applicability of spatial computing for accelerating B-spline-like functions. These

TABLE I
RUNTIME OF CUBIC B-SPLINE. ALL DATA IN SYSTOLIC PROCESSORS ARE PROCESSED IN THE FORM OF STREAMS.

x_{dim}	x86			Systolic Array				Wavefront Array			
	Cycles	Computing Time(μs)	Speedup	Cycles	Computing Time(μs)	Tiles	Speedup	Cycles	Computing Time(μs)	Tiles	Speedup
20	2559783	805	1.00	79320	64.5	30	12.48	41310	34.5	58	23.33
200	20378476	6396	1.00	429900	345.0	30	18.54	104584	83.7	58	76.42
2000	124103294	35929	1.00	4030661	3225.6	30	11.14	747016	597.6	58	60.12

TABLE II
CONFIGURATIONS OF VERSAL ASOC, HOST, AND GPU.

Versal ASoC	XCVC1902-VSVA2179-2MP-ES Dual-core ARM Cortex-A72 8 GB DIMM DDR4 , 400 AIEs @1.25GHz
CPU	Model:Intel(R) Core(TM) i9-14900K 32 cores @800-6000MHz (Min-Max) 2.2 MiB L1-I/D, 32 MiB L2, 36MiB L3 2× 32GB DIMM DDR5 @4800MHz Ubuntu 22.04.1 x86_64, GNU bash 5.1.16(1)
GPU	GPU: NVIDIA GeForce RTX 3090 Driver Version: 550.144.03 CUDA Version: 12.4, CUDA Cores: 10496 Memory: 24BG, Memory interface: 384-bit

TABLE III
RUNTIME OF CUBIC B-SPLINE FUNCTION.

Algorithm	Number of Cycles		Speedup	#Tiles
	x86	AIE		
De Boor	1745367	338867	5.15	1
Unrolled	946484	175230	5.40	1
Parallel	945643	92150	10.26	3

computations are compute-bound; therefore, realizing them on a single AIE tile is inefficient. So, as discussed in Section III, we illustrate the further improvement in B-spline computation using the systolic and wavefront architecture solutions.

Table I lists the performance of cubic B-spline calculation in systolic and wavefront arrays. Our results show that the compute time increases linearly with the input batch size. Systolic arrays deploy a highly pipelined architecture to address this scalability issue with increasing batch sizes, which start from the degree-0 basis function and calculate upwards. Thus, for systolic arrays with large batch size, *i.e.*, $x_{dim} = 2000$, the compute time improves by 11.14× compared to the CPU implementation. Similar performance gain is also visible for smaller batch sizes. The best performance is gained through the wavefront array architecture. A speedup of 60.12× is achieved for the large batch size, which is 5.39× faster than the systolic array. This performance improvement is due to the optimized dataflow design, which shifts away from the beat-driven nature of the systolic array to a relaxed operand-driven flow. However, as noted in Table IV, waveform design requires more power than their systolic counterpart.

TABLE IV
PERFORMANCE OF CUBIC B-SPLINE FUNCTION ON DIFFERENT PLATFORMS WITH $x_{dim} = 2000$.

Platform	Ave. Run Time (μs)	Power (W)	Energy (Power × Time)
CPU	35929	76.82	2.76 (J)
ASoC (Systolic)	3225.6	**7.30**	23.54 (mJ)
ASoC (Wavefront)	597.6	9.07	**5.42** (mJ)
GPU	**94.8**	189	17.92 (mJ)

We also accelerate the B-spline with GPU by distributing multi-dimension input into thousands of threads and exploring three architectures' energy efficiency. Table IV provides a broader comparison between CPU, GPU, and ASoC. Power consumption is adopted from the specific tool, *i.e.*, powerstat, nvidia-smi, and power design management (PDM), respectively. GPU achieves the best performance on average run time due to its multithreading, representing the highest throughput over three implementations. However, a higher parallelism also leads to a large power consumption of hundreds of watts. CPU requires around 77 Watts as it processes other tasks in the backend. The ASoC design works on a low-power domain that consumes less than 10 watts during the computation. We also introduce the energy by multiplying the power and the run time to evaluate the efficiency better. It shows that the wavefront design requires 5 mJ, which is the most efficient solution. GPU takes more energy as a result of its high power consumption. Compared to these two devices, the CPU is inefficient for such computing-intensive tasks.

V. CONCLUSION

This paper presents novel energy-efficient acceleration techniques to implement the KANs on edge-deployable Xilinx Versal ASoCs. Given the restrictive power budget at the edge, such designs should catalyze further innovation in realizing complex domain-specific machine learning and compute-intensive numerical algorithms for resource-constraint devices. In addition, we suggest exploring similar data-flow-based spatial designs to discover new architecture and computing primitives for emerging ML hardware, such as NVIDIA Tensor Cores and Tenstorrent's GraySkull processors.

REFERENCES

[1] Z. Liu, Y. Wang, S. Vaidya, F. Ruehle, J. Halverson, M. Soljačić, T. Y. Hou, and M. Tegmark, "KAN: Kolmogorov-Arnold Networks," 2024. [Online]. Available: https://arxiv.org/abs/2404.19756
[2] M. Thüning, "Attention in SRAM on Tenstorrent Grayskull," 2024. [Online]. Available: https://arxiv.org/abs/2407.13885
[3] AMD, "Versal: The First Adaptive Compute Acceleration Platform," Sep. 2020. [Online]. Available: https://docs.amd.com/v/u/en-US/wp505-versal-acap
[4] M. Elbanhawi, M. Simic, and R. N. Jazar, "Continuous Path Smoothing for Car-like Robots using B-spline Curves," *Journal of Intelligent & Robotic Systems*, vol. 80, pp. 23–56, 2015.
[5] K. Li, Z. Cao, and U. D. Hanebeck, "Continuous-Time Ultra-Wideband-Inertial Fusion," *IEEE Robotics and Automation Letters*, vol. 8, no. 7, pp. 4338–4345, 2023.
[6] N. T. Nguyen *et al.*, "B-spline Path planner for Safe Navigation of Mobile Robots," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2021, pp. 339–345.
[7] T. Chen and H. Chen, "Universal Approximation to Nonlinear Operators by Neural Networks with Arbitrary Activation Functions and its Application to Dynamical Systems," *IEEE Transactions on Neural Networks*, vol. 6, no. 4, pp. 911–917, Jul. 1995.
[8] L. Chierchia, "Kolmogorov–Arnold–Moser (KAM) Theory," in *Encyclopedia of Complexity and Systems Science*, R. A. Meyers, Ed. New York, NY: Springer New York, 2009, pp. 5064–5091.
[9] J. Vasiljevic *et al.*, "Compute Substrate for Software 2.0," *IEEE micro*, vol. 41, no. 2, pp. 50–55, 2021.
[10] S. Kung, "VLSI Array Processors," *IEEE ASSP Magazine*, vol. 2, no. 3, pp. 4–22, 1985.