52.5 TOPS/W 1.7GHz Reconfigurable XGBoost Inference Accelerator based on Modular-Unit-Tree with Dynamic Data and Compute Gating

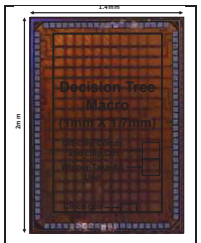
Chang Eun Song[1], Yidong Li[1], Amardeep Ramnani[1], Pulkit Agrawal[1], Purvi Agrawal[1], Sung-Joon Jang[2], Sang-Seol Lee[2], Tajana Rosing[1], Mingu Kang[1]

[1]University of California San Diego, USA, [2]KETI, Korea

The XGBoost has emerged as a powerful AI algorithm achieving high accuracy winning multiple Kaggle competitions in many tasks including medical diagnosis, recommendation system, and autonomous driving [1]. It provides a great potential for edge devices due to a binary-tree-based simple computing kernel compared to deep learning [2]. Despite such a potential from the kernel-level simplicity, the efficient end-to-end realization is hindered by multiple design challenges due to 1) the highly irregular tree shape, 2) low hardware utilization, 3) delay from the sequential processing of each tree node, and 4) a large data movement to all nodes [3-5]. We propose low-power and high performance XGBoost accelerator by employing modular unit trees and reconfigurable interconnects along with a selective data movement and execution. The proposed accelerator achieves 52.5 TOPS/W and 0.41 TOPS/mm^2, which are the best among the reported CNN and tree-based classifiers [2-5].

Fig. 1 (left top) describes the inference of XGBoost which generates the final decision by combining the local decisions from multiple (H) trees. Each tree corresponds to a specific class. Multiple trees are dedicated for the specific class, e.g., the # of trees dedicated for r-th class is H_r, where r = 1,…, R (H_1+...+H_R = H). At every tree node, the feature ID decides which feature out of multiple input features will be used to be compared with a threshold (Th) to establish a path between left and right children. When a node can no longer be compared (i.e., it is a leaf node), such node outputs gain (G), which is unique for each leaf node. The output gain ($G_{r,i}$) obtained from the i-th tree for r-th class is multiplied by a tree-specific weight ($W_{r,i}$). Summing the output gains of trees for the same (r-th) class yields the class gain ($G_r = \sum_{i=1}^{Hr} W_{r,i} \cdot G_{r,i}$). The final decision is obtained by selecting the class with the maximum gain. The XGBoost operation described above incurs multiple implementation challenges. The shape of trees has high irregularities in its width and depth across different trees. Brute force implementation with a full tree result in a large number of nodes as all potential node positions have to be annotated, e.g., 4095 nodes per tree with a depth of 12. As the depth grows, many nodes are not utilized, leading to low node utilizations, e.g., 0.7% in case2 (right top in Fig. 1). The operation also suffers from the inherent sequential nature (left bottom in Fig. 1) by running level by level sequentially toward leaf node leading to the large delay of >12. Running all nodes in parallel can enhance performance but at high energy costs, computing even unselected paths. Our solution uses a Modular Unit Tree (MUT) and reconfigurable interconnects to construct trees, conserving hardware resources by making unnecessary to prepare for all potential node positions. We also implement path-based dynamic gating (PDG) to activate data movement and execution only for selected paths, and a hybrid parallel-serial execution for balanced energy and delay. We achieve further energy reductions based on shape-based tree disabling to strictly gate the unused MUTs after building trees.

Fig. 2 shows the proposed XGBoost tree's architecture with depth D, organized into three MUT group stages with depths of D/3. The first stage has one MUT, the second N, and the third M MUTs (1<N<M). MUTs are linked by reconfigurable child node interconnects (RCNI) to form tree shapes before inference. MUTs are pre-loaded with feature IDs, thresholds, child tree IDs for bottom-most nodes, and leaf node output gains. The bottom-most nodes in the MUT can be routed to any MUT in the next stage increasing the hardware utilization based on the enhanced flexibility. The child tree ID at the bottom-most node indicate which MUT in the next stage should be connected to the bottom node. If a certain bottom-most node is a leaf (no further connection is needed), the corresponding gain (output) is stored instead of child tree ID. After building the shape of tree, the unused MUTs are fully disabled by shutting the clock for power savings, so-called shape-based MUT disabling. Such a reconfiguration for the tree shape building is required only once before the inference for multiple streamed inputs. The total count of nodes is much lower (e.g., 3.6×) than full shape-based trees even considering some of MUTs are left unused. The full-tree based conventional approach has a low node utilization of 0.78% (Fig. 2). In contrast, the proposed MUT-based approach achieves 3.6× improvement in the node utilization. The Winner-take-all unit generates the final decision by choosing the class with the maximum accumulated gain from many trees.



Die micrograph.

Fig. 3 depicts the parallel-serial hybrid operation of tree where all the nodes inside an MUT are computed in parallel while the MUTs across different stages are processed sequentially, e.g., 1st → 2nd → 3rd stage, which guarantees reaching a leaf node within 3 cycles whereas full serial processing takes >12 cycles. The MUT-based operation also guarantees executing only 45 nodes at maximum from all stages in total whereas full tree executes 4095 nodes. During the execution, the MUT receives input features. Each node's feature ID selects an input feature for threshold comparison. The results then go to a Lookup Table (LUT) to guide MUT-level decisions, yielding an output gain if it's a leaf or directing to the next MUT via the RCNI if not. The RCNI is pre-configured during the tree-building phase to choose a path during execution based on the input feature. We propose path-based dynamic gating (PDG) which enables the only MUTs on the selective path. All the other MUTs on the unselected path and sometimes the entire stage is completely gated preventing data movement on registers and computation to minimize power consumption leading to ~60% power savings compared to the case without such selective MUT enabling.

Fig. 4 depicts how input features and output gains move across stages with their timings. After the tree is built, stages execute in a pipeline, and if a leaf is reached at the 1st or 2nd stage, the process stops, passing the gain forward without further stage execution. The leaf-based data movement (LDM) method only transfers data to the next stage if the leaf isn't reached, reducing unnecessary data movement.

Fig. 5 analyzes the proposed architecture on four realistic datasets 1) traffic sign, 2) stroke, 3) heart disease, and 4) diabetes detections (8-classes for traffic sign, 2-classes for others). The depth (D) is 12 and the tree number (H) is 28 as the accuracy saturates at the number. The 8-bit fixed-point is used for all the data, which results negligible accuracy drop (<0.3%) compared to floating-point (Traffic sign: 90.0%, Stroke: 97.2%, Heart disease: 90.2%, Diabetes: 95.9%). The prototype supports up to 20 features, 10 classes, N = 14 and M = 62, which are sufficient for all the datasets, and a single 12-depth tree. It is scalable as the Winner-Take-All unit can collect the decisions from up to 150 trees through the serial off-chip interface. With real datasets, about half the inputs reach leaf nodes by the 2nd stage, highlighting the efficiency of our data and compute gating methods. Trees ending at the 1st stage use up to 29% less power than those ending at the 3rd. On average, our approach saves 12% power across datasets and trees, thanks to our PDG and LDM. Setting a D=4 for each MUT optimizes the node count and boosts node utilization to 32%, resulting in a 53× increase over traditional full tree designs.

Fig. 6 compares the prototype with prior tree accelerators [2-5]. The proposed work achieves >42× higher throughput than other tree-based works although this work supports the largest tree depth and more nodes. It also achieves more than 253× higher TOPS/W, and 404× higher TOPS/mm^2 (25°C) compared to others [3-5] due to its reconfigurability and selective processing. Shmoo plot shows the operation at the maximum 1.7GHz with 1.0V. A significant (72%) reduction in area is also achieved compared to the full tree. We also compare the energy efficiency with an 8-b CNN accelerator [2]. Even assuming tiny CNN model such as LeNET-5 [2, 6], the proposed work achieves 320× lower energy per decision even with 900 trees due to the simple computing kernel.

Acknowledgements:

This work was supported by IITP grant funded by MSIT (No.2022-0-00394, Development of AI semiconductor for level 4 self-driving based on functional safety), and in part by PRISM, center in JUMP 2.0, an SRC program sponsored by DARPA.

References:

[1] Tianqi Chen, et al., XGBoost: A Scalable Tree Boosting System. KDD, 2016.
[2] Y. Khodke, et al., CICC, 2023.
[3] M. Kang, et al., JSSC, 2018.
[4] K. J. Lee, et al., JSSC, 2015.
[5] T.-W. Chen, et al., VLSI, 2012.
[6] Yan LeCun, et al., Proceedings of the IEEE, 1998.

2024 IEEE Custom Integrated Circuits Conference (CICC) | 979-8-3503-9406-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/CICC60959.2024.10529017

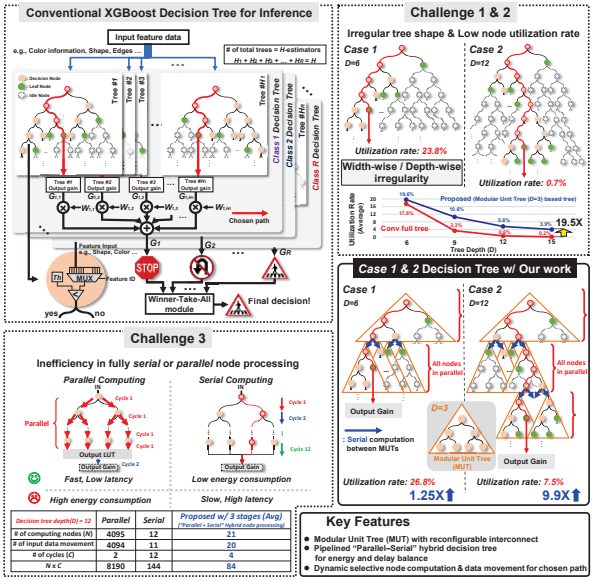


Fig. 1.XGBoost algorithm, implementation challenges and solutions.

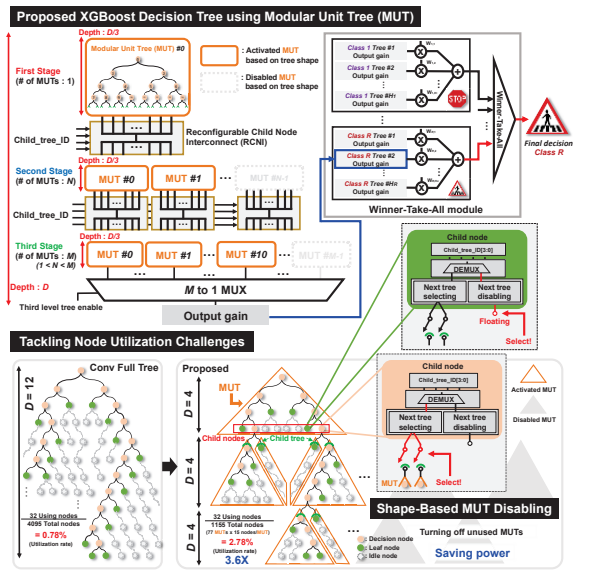


Fig. 2. The overall architecture with reconfigurability based on Modular Unit Tree (MUT) and shape-based MUT disabling.

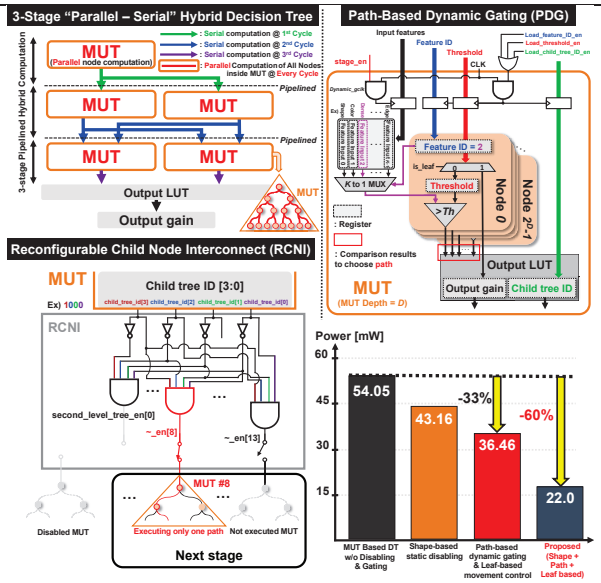


Fig. 3. Parallel-serial hybrid tree execution and path-based dynamic gating (PDG) for enhanced energy efficiency.

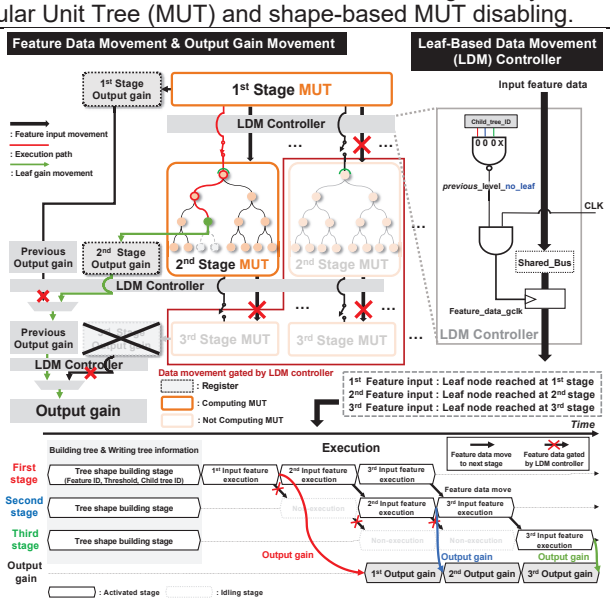


Fig. 4. The dataflow of input feature and output gain controlled by Leaf-based data movement (LDM) and timing diagram.

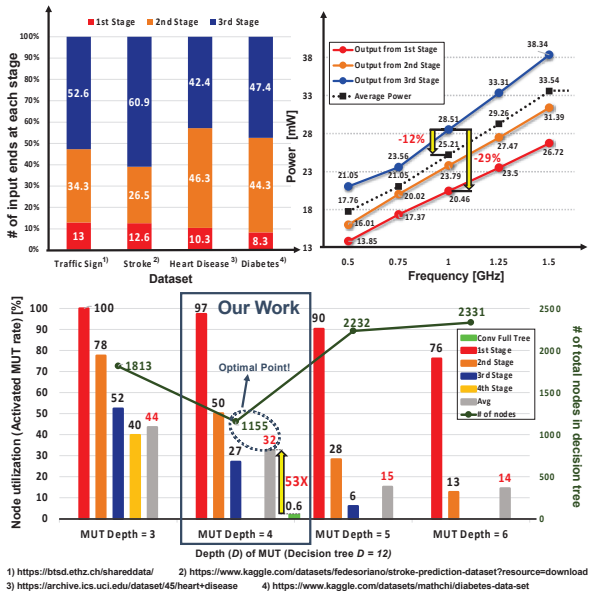
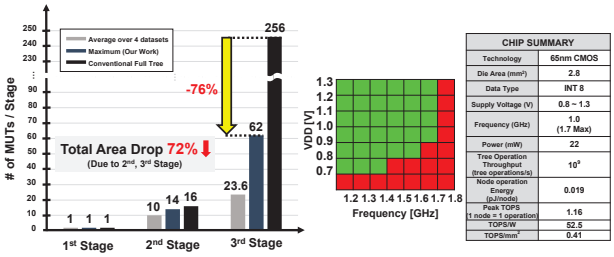


Fig. 5. Statistics with four realistic datasets, estimated benefits, and decision optimizations.



CHIP SUMMARY	
Technology	65nm CMOS
Die Area (mm²)	2.8
Data Type	INT 8
Supply Voltage (V)	0.8 ~ 1.3
Frequency (GHz)	1.0 (1.7 Max)
Power (mW)	22
Tree Operation Throughput (tree operations/s)	10^9
Node operation Energy (pJ/node)	0.019
Peak TOPS (1 node = 1 operation)	1.16
TOPS/W	52.5
TOPS/mm²	0.41

	Our Work	[3] JSSC 2018	[4] JSSC 2015	[5] VLSI 12	[2] CICC 2023
Process	**65 nm CMOS**	65 nm CMOS	65 nm CMOS	65 nm CMOS	65 nm CMOS
Algorithm	**XGBoost**	Random Forest	Vocabulary Forest	Vocabulary Tree	CNN, FC, attention
Frequency (GHz)	**1.0 (1.7 Max)**	1	0.25	0.2	0.5
Power (mW)	**22**	7.1	27.6	5.6	400
Die Area (mm²)	**2.8**	1.44	2.3	6.3	1.87
Tree operation Throughput (tree operations/s)	**10^9**	23.3 X 10^6	7680	3840	-
Tree Depth	**12**	6	8	8	-
Total nodes/Tree	**1155**	63	255	255	-
Node operation Energy (pJ/node)	**0.019**	4.83	14093.13	5718.95	1.67 pJ/op
Peak TOPS (1 node = 1 operation)	**1.16**	0.0014	1.95 X 10^{-6}	0.97 X 10^{-6}	1.024
TOPS/W	**52.5**	0.20	7.095 X 10^{-5}	17.4 X 10^{-5}	0.6
TOPS/mm²	**0.41**	0.001	8.51 X 10^{-7}	4.25 X 10^{-7}	0.16

Fig. 6. Chip summary, Measured results, shmoo plot, and comparison with the prior CNN and tree-based works.