Vectorization and Optimization of Gradient Boost Libraries for EUPilot VEC Chiplet

Ali Serdar Atalay[1], Orkun Hasekioğlu[2], Muhammed Enis Şen[1] and Şener Özönder[3]

[1] AI4SEC OÜ, Tallinn, Estonia
[2] TUBITAK Fundamental Sciences Research Institute, Turkey
[3] Institute for Data Science Artificial Intelligence, Boğaziçi University

Introduction

The increasing demand for efficient and accurate machine learning algorithms is driving the exploration of specialized hardware architectures to accelerate computationally intensive workloads. Gradient Boosting Algorithms (GBA) [1], such as XGBoost, have demonstrated remarkable predictive power in this domain, but their performance can be significantly improved by leveraging hardware-specific optimizations. This work targets the RISC-V VEC chiplet, a vector accelerator integrated into the EUPilot's platform [2], and focuses on optimizing GBAs for its long vector length and custom intrinsics. We investigate manual vectorization using intrinsics and compiler-assisted automatic vectorization to accelerate the most intensive loops. Our implementation is benchmarked on publicly available datasets, where inference speed, throughput, and energy efficiency are all critical.

To fully exploit the VEC chiplet's projected capabilities, we analyze memory access patterns, data layout, and vector instruction utilization. The resulting implementation demonstrates notable gains in both runtime and performance-per-watt, offering insights into effective co-design strategies for machine learning workloads on specialized RISC-V systems.

Methodology

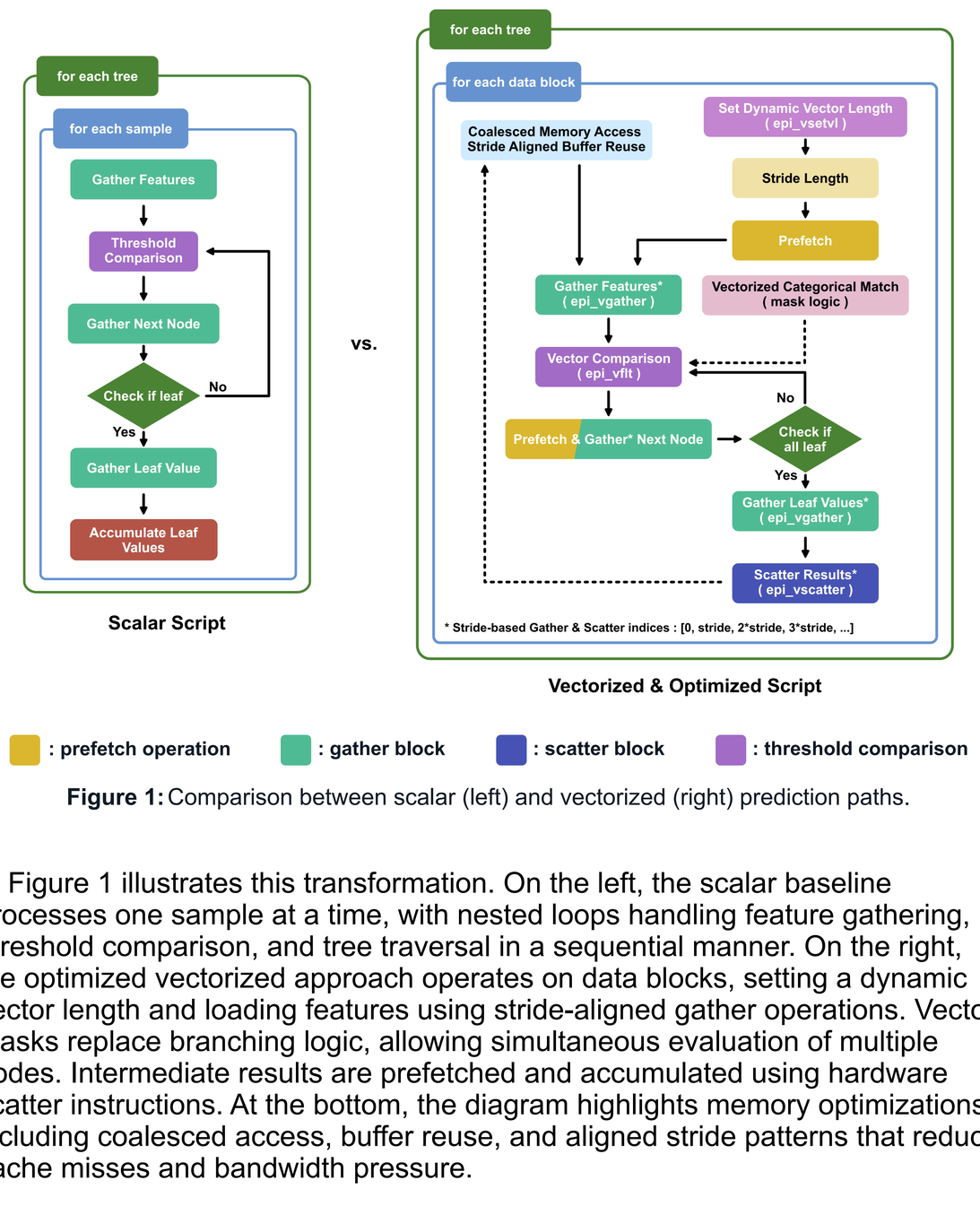

Figure 1: Comparison between scalar (left) and vectorized (right) prediction paths.

Figure 1 illustrates this transformation. On the left, the scalar baseline processes one sample at a time, with nested loops handling feature gathering, threshold comparison, and tree traversal in a sequential manner. On the right, the optimized vectorized approach operates on data blocks, setting a dynamic vector length and loading features using stride-aligned gather operations. Vector masks replace branching logic, allowing simultaneous evaluation of multiple nodes. Intermediate results are prefetched and accumulated using hardware scatter instructions. At the bottom, the diagram highlights memory optimizations including coalesced access, buffer reuse, and aligned stride patterns that reduce cache misses and bandwidth pressure.

Algorithm 1 Vectorized Prediction with Stride-based Tree Traversal

```
Set vector length based on sample size
Compute stride indices for vectorized access
for each decision tree in the ensemble do
    Prefetch thresholds, child indices, and leaf values for the tree
    for each block of samples do
        Prefetch upcoming input features
        Gather feature values using stride-based access
        Initialize current nodes to root for all samples
        while any node is not a leaf do
            for each feature block do
                Prefetch thresholds and child indices for next level
                Compare feature values with thresholds
                Use comparison result to choose left or right child
                Update current nodes
            end for
        end while
        Gather leaf values based on final nodes
        Write predictions back using stride-based scatter
    end for
end for
```

Results

Table 1 : Performance comparison of scalar vs. vectorized implementations across different compute configurations.

Configuration	Metric	Scalar	Vectorized
Single Core	Branch Mispredict Rate	15%	12%
	Cache Misses (per 1K Instr.)	50	35
	Memory Latency (cycles)	200	150
	Throughput (pred/s)	800	3,200
64 Core	Throughput (pred/s)	51,200	204,800
	Bandwidth Utilization	40 GB/s (40%)	120 GB/s (60%)
	L3 Cache Hit Rate	55%	70%
	Core Utilization	70%	75%
	Power Efficiency (pred/W)	2.5	4.0
4 Node	Total Throughput	204,800	655,360
	Scaling Efficiency	85%	80%

The final implementation was evaluated on the Google Universal Image Embedding dataset [3], using the Milk-V Pioneer platform [4], which features a 64-core RISC-V CPU powered by the SOPHON SG2042. Performance was assessed using metrics such as inference speed, vector utilization, cache hit rate, memory bandwidth, and energy per operation. Iterative refinements in memory layout, instruction placement, and prefetch tuning led to significant improvements in throughput and efficiency.

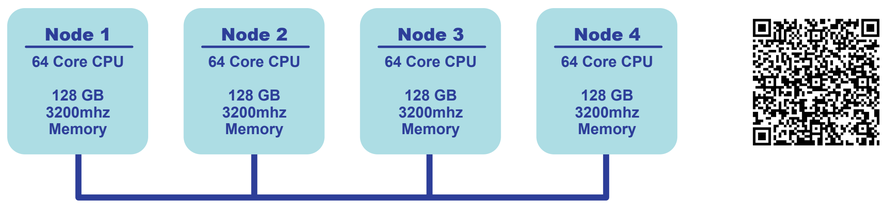

Figure 2: 4 Node Milk-V Pioneer Cluster in BSC

Discussion & Conclusion

Table 2 : Comparison of Scalar vs. Vectorized Prediction Paths in GBA

Features	Scalar Approach	Vectorized Approach
Execution Style	Per-sample, serial evaluation	SIMD lanes process multiple samples in parallel
Feature Gathering	Manual access using looped FVec	`epi_vload`, gathered via vector index
Comparison	Scalar `if (val < threshold)`	`epi_vfcmp`, parallel element-wise mask
Child Node Selection	Branch-based (if-else) per sample	`epi_vmerge`, mask-select for both paths
Leaf Check	Per-sample test on node flags	Vectorized mask logic + loop condition for all lanes
Memory Access Pattern	Irregular, cache-unaware	Stride-aligned buffers, coalesced gather/scatter
Prefetching	None	Multi-level lookahead prefetching using `epi_prefetch`
Batch Size Handling	Fixed-size, one-at-a-time	Dynamic VL (e.g., `epi_vsetvl`) adapts to batch
Categorical Feature Processing	Scalar loop or set search	Vectorized category check (e.g., broadcasting & masked eq)
Performance Adaptation	Static logic	Runtime metrics guide adaptive tuning
Compute Utilization	Low – branch-heavy, pipeline stalls	High – branchless logic via mask ops
Scatter to Output	Direct assignment: `out_pred[i] = leaf_val`	`epi_vscatter` for aligned SIMD write
Scalability (batch size, core count)	Poor	Excellent – vector units, large batches, parallel threads

This work highlights the potential of low-level vectorization techniques to enhance the performance of GBAs on VEC chiplets. By examining the VEC chiplet's vector capabilities and combining manual intrinsics with automated compiler optimizations, we significantly enhanced the performance of gradient boosting algorithms without sacrificing accuracy or scalability. The results highlight the importance of hardware-aware code transformation, particularly in memory-bound workloads where stride-based access and predicated vector operations can substantially reduce bottlenecks.

References

[1] Evgeny Kozinov et al. "Vectorization of Gradient Boosting of Decision Trees Prediction in the CatBoost Library for RISC-V Processors". In: arXiv preprint arXiv:2405.11062 (2024).
[2] EUPilot, The European PILOT project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 101034126. url: https://eupilot.eu/.
[3] Google Research and Kaggle. Google Universal Image Embedding. https://www.kaggle.com/competitions/googleuniversal-image-embedding. 2022.
[4] Milk-V Pioneer. https://milkv.io/pioneer. 2024.