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

Abstract

The increasing demand for efficient and accurate machine learning algorithms in various fields, including drug discovery, has led to a growing interest in optimizing gradient boost libraries for specialized hardware architectures. Various vectorization techniques, including loop unrolling to reduce overhead, data parallelism for simultaneous processing of multiple data elements, and instruction-level parallelism to execute multiple instructions per clock cycle, along with SIMD and SIMT, can be employed to enhance performance. By leveraging these vectorization techniques, the XGBoost library is optimized to achieve significant performance gains on a RISC-V platform representative of future integration with the EUPilot VEC chiplet, a cutting-edge, low-power, and highly scalable vector processing unit designed to accelerate machine learning workloads, which is a key component of the EUPilot Horizon Europe project, aiming to develop a scalable and energy-efficient computing platform. The optimized libraries demonstrate significant improvements in computational efficiency, enabling faster and more accurate predictions. Specifically, manual vectorization using intrinsics as well as automated vectorization tools and techniques are employed to optimize the computationally intensive loops in Gradient Boosting Algorithms. Our work is compared with non-optimized versions, and the results are analyzed in terms of performance gain, accuracy, and computational efficiency.

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 VEC chiplets.

Methodology

The optimization process began with profiling the baseline implementation to identify performance bottlenecks. Over 70% of execution time was concentrated in tree traversal and prediction accumulation routines, which relied on scalar operations with sequential feature loading and node comparisons. These routines exhibited three properties ideal for vectorization: regular memory access, independent comparisons across nodes, and a high ratio of computations to memory loads.

Based on these insights, the core prediction logic was redesigned for vectorized execution. Feature access was reorganized to enable stride-based memory operations, and scalar comparisons were replaced with vectorized threshold checks using mask-based logic. Conditional branches were eliminated through vector predication, and prefetch buffers were introduced to reduce memory access latency. The architecture's dynamic vector length feature was leveraged to adapt processing width at runtime, improving overall utilization of vector lanes.

To fully exploit the available hardware capabilities, manual vectorization was applied using low-level vector intrinsics for gather, scatter, and masked operations. These were supported by automated vectorization enabled through compiler tuning and loop restructuring. Special attention was given to cache-aligned stride patterns, minimizing memory traffic, and maximizing arithmetic throughput.

Figure 1 illustrates this transformation. On the left, the scalar baseline processes one sample at a time, with

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%

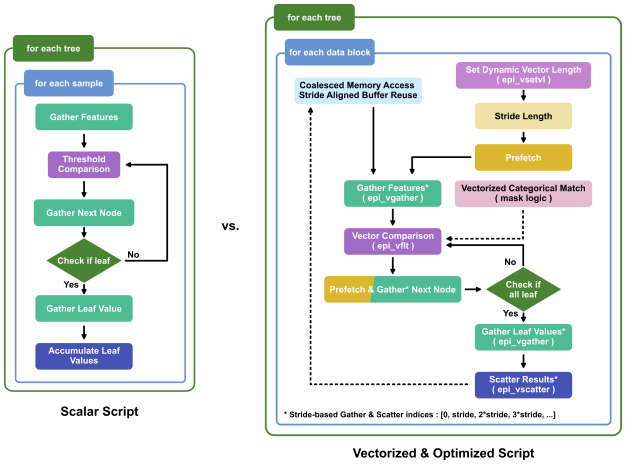

Figure 1: *Comparison between scalar (left) and vectorized (right) prediction paths.*

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.

Results

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.

As shown in Table 1, the vectorized implementation achieved a 4× increase in single-core throughput, reduced cache misses and memory latency by 30% and 25%, and improved IPC from 0.8 to 1.2 with 65% vector unit utilization. On a 64-core node, these gains scaled effectively, quadrupling throughput and increasing memory bandwidth utilization to 60%, while improving power efficiency from 2.5 to 4.0 predictions per watt. The 4-node setup sustained a 3.2× speedup, reaching over 655,000 predictions per second with strong scaling efficiency. These results demonstrate the effectiveness of vectorization in accelerating gradient boosting on such architectures.

Discussion & Conclusion

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.

Future work involves staged validation starting with commercial RISC-V nodes, FPGA emulation, and finally a full VEC-based HPC system under EUPilot.

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/google-universal-image-embedding`. 2022.

[4] *Milk-V Pioneer*. `https://milkv.io/pioneer`. 2024.