QUETZAL: Vector Acceleration Framework for Modern Genome Sequence Analysis Algorithms

Julian Pavon, Ivan Vargas Valdivieso, Osman Ünsal and Adrian Cristal
Department of Computing Science
Barcelona Supercomputing Center
Barcelona, Spain
Email: {julian.pavon, ivan.vargas, osman.unsal, adrian.cristal}@bsc.es

***Abstract*—Genome sequence analysis is fundamental to medical breakthroughs such as developing vaccines, enabling genome editing, and facilitating personalized medicine. The exponentially expanding sequencing datasets and complexity of sequencing algorithms necessitate performance enhancements. While the performance of software solutions is constrained by their underlying hardware platforms, the utility of fixed-function accelerators is restricted to only certain sequencing algorithms. This paper introduces `Quetzal`, a novel vector acceleration framework for genomics algorithms. It addresses limitations in conventional CPU vector datapaths, offering hardware-software co-design with novel vector instructions. `Quetzal` supports both short and long reads, minimizes memory latency, and achieves a 5.67× speedup over vectorized CPU baselines in sequencing algorithms.**

I. Motivation

A. Challenges in accelerating modern genome sequencing algorithms on vector architectures

Commodity high-performance ARM CPUs include support for vector hardware composed of a Vector Register File (VRF), where each vector register is an array of elements, and a Vector Processing Unit (VPU), which consists of multiple parallel execution units referred to as lanes [1].

Modern genome sequencing algorithms like Wavefront Align [2] employ scatter-gather memory instructions[1], which limit the performance of vector hardware. These instructions are split into multiple memory requests, extending their overall processing latency. Each request calculates an associated address independently, requiring multiple cycles. The load-store queue lacks memory coalescing for memory indexed instructions. For instance, in Intel and Fujitsu A64FX processors, scatter-gather instruction latency is at least 22 and 19 cycles, respectively, even with all data in the L1D cache. To better understand this bottleneck, Fig. 1 depicts the breakdown of the execution time for various vectorized genome sequence analysis benchmarks running on a HPC ARM machine with two levels of cache, using the methodology outlined in Section III. The figure shows how cache accesses represent a considerable amount of the overall execution time in all algorithms. This bottleneck worsens with input sequence size due to the expanding active working set, exceeding on-device memory capacity and leading to a memory-bound behavior.

[1]Scatter-gather memory instructions are also called the memory indexed instructions. This paper uses these two terms interchangeably.

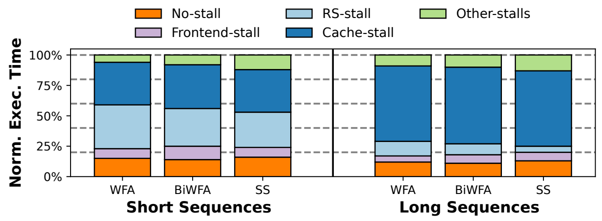

Fig. 1. Execution breakdown of vectorized genome sequence analysis benchmarks for short and long input sequences, broken down into: no-stall and stalls due to frontend, Reservation Station (RS), cache and others.

Additionally, scatter-gather instructions fragment into multiple memory requests, occupying processor pipeline structures like Load/Store queues or caches, thereby serializing execution of other memory operations. Thus, efficient hardware implementation of these instructions will notably enhance genome sequencing algorithm performance.

B. Rationale for flexible general-purpose accelerators

ASIC-based domain-specific accelerators (e.g., [3]–[6]) offer superior performance and energy efficiency over general-purpose CPUs, but require costly custom silicon tailored to specific algorithms. Meanwhile, genome analysis algorithms evolve rapidly, making fixed-function accelerators inflexible. For instance, Smith-Waterman (SW) has evolved from its original form [7] to banded [8] and adaptive banded variants [9]. Alser *et al.* [10] reviewed 107 tools from 1988–2020 and observed: (1) continual publication of new tools and algorithms [10], (2) varied sequencing technologies increasing algorithmic demands [10]–[12], and (3) a trend toward combining multiple algorithms at runtime [10]. These trends motivate the need for flexible, programmable hardware acceleration.

CPUs and GPUs offer programmability for genome analysis. SIMD/vector acceleration on CPUs has been widely explored (e.g., [13]–[16]), but non-unit stride memory patterns limit efficiency (see SectionI-A). GPUs exploit massive parallelism and achieve strong performance on short sequences [6], [17]–[20], yet scale poorly with long sequences due to memory pressure [21], [22]. Long-read sequencing [23] and growing genome datasets increase the demand for efficient long-sequence analysis [10], [12], [23]–[25], underscoring the need for versatile acceleration.

We target CPU-based acceleration of genome analysis for two reasons: (1) as shown in Section I-A, vectorized CPU performance is limited by memory instruction inefficiencies —optimizing the execution of these memory instructions can yield substantial performance improvements. (2) As shown in Section IV-C, GPUs underperform on long sequences due to memory constraints. Thus, hardware-software co-designed CPU-based solutions, such as our proposal (`Quetzal`), could outperform GPUs for long-read workloads.

II. Quetzal Overview

`Quetzal` is a vector acceleration framework consisting of two main components: a vector accelerator tightly coupled to the Vector Processing Unit (VPU) datapath and a set of novel vector instructions that expose the functionality of the accelerator to the programming model. `Quetzal` design is driven by three main goals: (1) accelerate memory indexed instructions in modern genome sequencing algorithms, (2) provide a flexible framework applicable to multiple algorithms, and (3) achieve a light-weight hardware implementation that reuses the available hardware in ARM SVE implementations.

`Quetzal` accelerator is composed of four main components, as shown in Fig. 2: (1) Two hardware `buffers` that are directly connected to the VPU to quickly forward data to the vector ALU without using the cache hierarchy (*e.g.,* the input genome sequences). (2) `data encoder` that applies a static bit-encoding to reduce the size of the DNA/RNA input sequences stored in the `buffers`. (3) `access ctrl` that process all the data accesses from the VPU to the `buffers` and works as the interface between the `buffers` and the core's VPU components, and (4) `count ALU` that counts the number of consecutive elements between two input values.

1) Accelerating memory indexed instructions: `Quetzal` incorporates two hardware `buffers` specifically designed to deliver sufficient bandwidth to the VPU for rapid execution of memory indexed instructions. These `buffers` store frequently used genome sequencing data, in particular those values accessed through memory indexed instructions. Then, the algorithm utilizes `Quetzal` instructions to access the values previously stored in the `buffers`. `Quetzal buffers` features three key characteristics enabling them to provide more efficient support for memory indexed operations. (1) They are direct-mapped. Then, instead of using memory addresses (requiring address translation), `Quetzal` uses indices to access the `buffers`, thus, requiring a simpler control compared to caches. (2) `Quetzal buffers` are highly multiported structures, allowing the VPU to access data in only two cycles, a significant improvement over the 22 or 19 cycles required in Intel and A64FX cores, respectively. (3) `Quetzal buffers` support bit-encoded values (less than 8 bits), reducing the overhead of accessing unaligned.

2) Accelerating counting consecutive matching elements: Counting consecutive matching elements is useful for different applications that calculate maximal exact matches (MEMs) [26] and maximal unique matches (MUMs) [27]

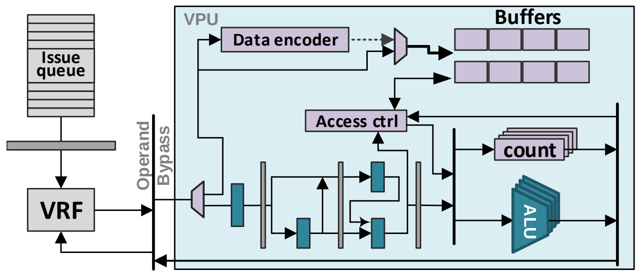

Fig. 2. Overview of `Quetzal` hardware (purple) integrated into the VPU datapath (dark blue).

including SneakySnake [28], protein multiple sequence alignment [29], read mapping [26], and sequence alignment [30]. `Quetzal` features a specialized `count ALU` capable of efficiently counting the number of consecutive matches between two input sequences. We employ this functional unit together with the `Quetzal buffers` to significantly reduce the instruction overhead of modern algorithms when counting consecutive matching elements.

III. Experimental Environment

Simulator: We evaluate `Quetzal` using the gem5 simulator [31], simulating a 16-core aarch64 full-system with Ubuntu 20.04 and Linux Kernel 4.18.0+. Our gem5 model is validated against a Fujitsu A64FX-like architecture [32], used in the Fugaku supercomputer. Each core includes a `Quetzal` module connected to its VPU. We extend gem5's Out-of-Order model to simulate `Quetzal`'s functionality and latency.

Benchmarks and datasets: We assess `Quetzal`'s performance with widely used sequence alignment algorithms. For the baseline, we select two modern sequence aligners: Wavefront Align (`WFA` [2]), Bidirectional Wavefront Align (`BiWFA` [33]), and one modern sequence filter: SneakySnake (`SS` [28]). For the baseline algorithms, we utilize compiler auto-vectorization. For each algorithm we implemented a hand-coded vectorized version using ARM SVE intrinsics (referred as VEC). Additionally, we evaluate two `Quetzal` versions: one only with hardware buffers (QUETZAL) and one including the `count ALU` as well (QUETZAL+C). We validate correctness by bitwise comparing `Quetzal` outputs with naive versions. We evaluate `Quetzal` using both short (100 - 300 base pairs) and long (1K - 30K base pairs) DNA/RNA sequences.

Comparison between Quetzal and GPU approaches. We compare the performance of the `Quetzal`-based implementation of `WFA` against WFA-GPU [21], a GPU-based approach using the same algorithm. In these experiments, we use a 16-core CPU featuring `Quetzal` and an NVIDIA A40 GPU. We use the open-source implementation available for WFA-GPU [34].

IV. Evaluation

A. Single-core performance analysis

Fig. 3 depicts the normalized performance results for all the evaluated algorithms.

Modern sequence aligners: For short reads, QUETZAL and QUETZAL+C achieve 1.5× and 2.1× better performance, respectively, compared to the VEC algorithm. For long reads, the improvement is 5.1× and 5.5×, respectively. These enhancements stem from (1) the `buffers` reducing memory indexed instruction latency to just 2 cycles and (2) the `count ALU` hardware accelerating the counting of consecutive matching elements in a single instruction.

When processing short reads, modern algorithms are dominated by both reservation station stalls and cache accesses (as shown in Section I-A). As such, QUETZAL+C provides significantly better performance by reducing the number of instructions executed. On the other hand, when long reads are processed, these algorithms are dominated by cache accesses. As such, `Quetzal` provides significant performance benefits even when using only the `buffers`.

Modern sequence filters: On average, a system with QUETZAL+C shows 2.1× and 5.2× better performance than the VEC algorithm for short and long reads respectively. `SS` features similar bottlenecks than `WFA` and `BiWFA`, thus, `Quetzal`'s hardware efficiently accelerates this algorithm.

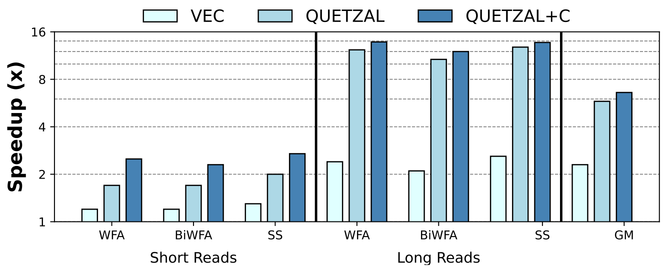

Fig. 3. QUETZAL performance results for all the evaluated algorithms using short reads and long reads input datasets. Results are normalized to the naive implementation of each algorithm.

B. Multicore scalability

Fig. 4.a depicts the multicore scalability evaluation of `Quetzal` over all the previously evaluated algorithms and datasets using the QUETZAL+C configuration. All `Quetzal`-based implementations demonstrate good performance scalability as thread count increases. However, performance does not increase linearly with the number of threads.

For small input sequences, the cache hierarchy can accommodate the entire set of DP matrices, enabling near-linear speedups. In contrast, for large input sequences, each DP matrix exceeds the capacity of the last-level cache (LLC), requiring frequent off-chip memory accesses to read and update matrix entries. As the number of threads increases, the aggregate number of off-chip memory requests rises accordingly, causing memory bandwidth to become the primary bottleneck that limits further scalability.

C. Comparison with GPU approaches

We evaluate the performance of `Quetzal` compared to the WFA-GPU implementation. In our experiments, we use the entire NVIDIA A40 GPU and a 16-core `Quetzal` capable

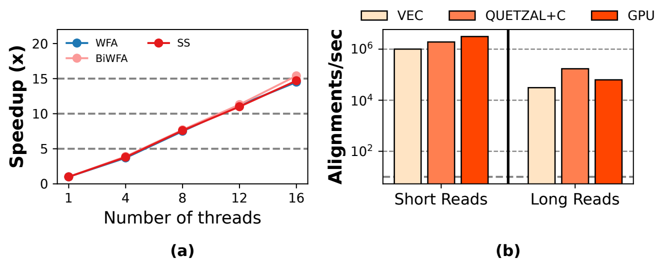

Fig. 4. (a) Multicore scalability using the QUETZAL+C version of each algorithm. (b) Throughput comparison between QUETZAL+C and GPU approaches. Results are reported on a logarithmic scale.

CPU to align all the input datasets listed in Section III. We evaluate multiple alignment parameters for the GPU implementations and report the best-performing results.

Fig. 4.b shows the throughput results obtained. We make three observations: (1) When processing short sequences, the parallelism offered by GPUs can outperform VEC and `Quetzal` designs. However, the NVIDIA A40 GPU consumes >10× more area compared to `Quetzal`. (2) The sequence size limits the parallelism offered by GPUs. With longer sequence lengths, the active working set, encompassing metadata, DP matrix, and other structures, increases significantly. Consequently, the available on-chip memory can serve only a small number of GPU threads, an effect called *low occupancy*, which significantly reduces the performance for long sequences compared to shorter sequences [6], [21], [22]. For example, WFA-GPU outperforms WFA (VEC) by 2.0×, which represents a performance drop of 40% compared to short sequences. (3) As analyzed in Section I-A, when processing long sequences, the execution time of modern genome sequence analysis algorithms is dominated by memory-indexed instructions. `Quetzal` efficiently accelerates these instructions, providing notable performance benefits. On average, `Quetzal` outperforms WFA-GPU by 2.7× for long sequences.

V. Conclusion

We propose `Quetzal`, the first vector acceleration framework capable of efficiently supporting a wide range of modern state-of-the-art genome sequence analysis algorithms. `Quetzal` integrates a cost-effective vector accelerator within a general-purpose ARM CPU's vector datapath, offering both high performance and programmability for modern and emerging workloads. We evaluate `Quetzal` across two use cases: sequence alignment and edit distance approximation. Our results show that `Quetzal` is a power- and area-efficient scratchpad-based design that significantly accelerates genome sequence analysis algorithms for both short and long sequences. Nevertheless, `Quetzal` significantly outperforms to GPU-based algorithms by 2.7× for long input reads.

Acknowledgment

This work has been accepted in the Proceedings of The 51st ACM International Symposium on Computer Architecture (ISCA-51), 2024 [35].

References

[1] F. Minervini, O. Palomar, O. Unsal, E. Reggiani, J. Quiroga, J. Marimon, C. Rojas, R. Figueras, A. Ruiz, A. Gonzalez *et al.*, "Vitruvius+: An Area-Efficient RISC-V Decoupled Vector Coprocessor for High Performance Computing Applications," *TACO*, 2023.

[2] S. Marco-Sola, J. C. Moure, M. Moreto, and A. Espinosa, "Fast gap-affine pairwise alignment using the wavefront algorithm," *Bioinformatics*, vol. 37, no. 4, pp. 456–463, 09 2020. [Online]. Available: https://doi.org/10.1093/bioinformatics/btaa777

[3] Y. Gu, A. Subramaniyan, T. Dunn, A. Khadem, K. Chen, S. Paul, M. Vasimuddin, S. Misra, D. T. Blaauw, S. Narayanasamy, and R. Das, "GenDP: A Framework of Dynamic Programming Acceleration for Genome Sequencing Analysis," in *ISCA*, 2023.

[4] D. S. Cali, G. S. Kalsi, Z. Bingöl, C. Firtina, L. Subramanian, J. S. Kim, R. Ausavarungnirun, M. Alser, J. Gomez-Luna, A. Boroumand *et al.*, "GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis," in *MICRO*, 2020.

[5] Y. Turakhia, S. D. Goenka, G. Bejerano, and W. J. Dally, "Darwin-WGA: A Co-Processor Provides Increased Sensitivity in Whole Genome Alignments With High Speedup," *HPCA*, 2019.

[6] J. Lindegger, D. S. Cali, M. Alser, J. Gómez-Luna, N. M. Ghiasi, and O. Mutlu, "Scrooge: A Fast and Memory-Frugal Genomic Sequence Aligner for CPUs, GPUs, and ASICs," *Bioinformatics*, 2023.

[7] T. F. Smith and M. S. Waterman, "Identification of Common Molecular Subsequences," *JMB*, 1981.

[8] K.-M. Chao, W. R. Pearson, and W. Miller, "Aligning Two Sequences Within a Specified Diagonal Band," *Bioinformatics*, 1992.

[9] Y.-L. Liao, Y.-C. Li, N.-C. Chen, and Y.-C. Lu, "Adaptively Banded Smith-Waterman Algorithm for Long Reads and Its Hardware Accelerator," in *ASAP*, 2018.

[10] M. Alser, J. Rotman, D. Deshpande, K. Taraszka, H. Shi, P. I. Baykal, H. T. Yang, V. Xue, S. Knyazev, B. D. Singer *et al.*, "Technology Dictates Algorithms: Recent Developments in Read Alignment," *Genome Biology*, 2021.

[11] PacBio, "PacBio," Available at https://www.pacb.com/technology/hifi-sequencing/how-it-works/, accessed: 2023-03-23.

[12] M. Alser, Z. Bingöl, D. S. Cali, J. Kim, S. Ghose, C. Alkan, and O. Mutlu, "Accelerating Genome Analysis: A Primer on an Ongoing Journey," *IEEE Micro*, 2020.

[13] J. Daily, "Parasail: SIMD C library for global, semi-global, and local pairwise sequence alignments," *BMC Bioinform.*, vol. 17, p. 81, 2016. [Online]. Available: https://doi.org/10.1186/s12859-016-0930-z

[14] H. Suzuki and M. Kasahara, "Introducing Difference Recurrence Relations for Faster Semi-Global Alignment of Long Sequences," *BMC Bioinformatics*, 2018.

[15] M. Vasimuddin, S. Misra, H. Li, and S. Aluru, "Efficient Architecture-Aware Acceleration of BWA-MEM for Multicore Systems," in *IPDPS*, 2019.

[16] H. Xin, J. Greth, J. Emmons, G. Pekhimenko, C. Kingsford, C. Alkan, and O. Mutlu, "Shifted Hamming Distance: A Fast and Accurate SIMD-friendly Filter to Accelerate Alignment Verification in Read Mapping," *Bioinformatics*, 2015.

[17] Y. Liu, A. Wirawan, and B. Schmidt, "CUDASW++ 3.0: Accelerating Smith-Waterman Protein Database Search by Coupling CPU and GPU SIMD Instructions," *BMC Bioinformatics*, 2013.

[18] N. Ahmed, T. D. Qiu, K. Bertels, and Z. Al-Ars, "GPU Acceleration of Darwin Read Overlapper for De Novo Assembly of Long DNA Reads," *BMC Bioinformatics*, 2020.

[19] NVIDIA, "NVBIO," https://github.com/NVlabs/nvbio, 2014.

[20] N. Ahmed, J. Lévy, S. Ren, H. Mushtaq, K. Bertels, and Z. Al-Ars, "GASAL2: A GPU Accelerated Sequence Alignment Library for High-Throughput NGS Data," *BMC Bioinformatics*, 2019.

[21] Q. Aguado-Puig, S. Marco-Sola, J. C. Moure, C. Matzoros, D. Castells-Rufas, A. Espinosa, and M. Moreto, "WFA-GPU: Gap-Affine Pairwise Alignment Using GPUs," *bioRxiv*, 2022.

[22] S. Park, H. Kim, T. Ahmad, N. Ahmed, Z. Al-Ars, H. P. Hofstee, Y. Kim, and J. Lee, "SALoBa: Maximizing Data Locality and Workload Balance for Fast Sequence Alignment on GPUs," in *IPDPS*, 2022.

[23] M. Alser, J. Lindegger, C. Firtina, N. Almadhoun, H. Mao, G. Singh, J. Gomez-Luna, and O. Mutlu, "From Molecules to Genomic Variations: Accelerating Genome Analysis via Intelligent Algorithms and Architectures," *CSBJ*, 2022.

[24] S. L. Amarasinghe, S. Su, X. Dong, L. Zappia, M. E. Ritchie, and Q. Gouil, "Opportunities and Challenges in Long-Read Sequencing Data Analysis," *Genome Biology*, 2020.

[25] C. Cheng, Z. Fei, and P. Xiao, "Methods to Improve the Accuracy of Next-Generation Sequencing," *Frontiers in Bioengineering and Biotechnology*, 2023.

[26] N. Khiste and L. Ilie, "E-MEM: Efficient Computation of Maximal Exact Matches for Very Large Genomes," *Bioinformatics*, 2014.

[27] S. Giuliani, G. Romana, and M. Rossi, "Computing Maximal Unique Matches With the R-Index," *arXiv*, 2022.

[28] M. Alser, T. Shahroodi, J. Gómez-Luna, C. Alkan, and O. Mutlu, "SneakySnake: a fast and accurate universal genome pre-alignment filter for CPUs, GPUs and FPGAs," *Bioinformatics*, 12 2020, btaa1015. [Online]. Available: https://doi.org/10.1093/bioinformatics/btaa1015

[29] M. Chatzou, C. Magis, J.-M. Chang, C. Kemena, G. Bussotti, I. Erb, and C. Notredame, "Multiple Sequence Alignment Modeling: Methods and Applications," *Briefings in Bioinformatics*, 2016.

[30] A. Bayat, B. Gaëta, A. Ignjatovic, and S. Parameswaran, "Pairwise Alignment of Nucleotide Sequences Using Maximal Exact Matches," *BMC Bioinformatics*, 2019.

[31] J. Lowe-Power, A. M. Ahmad, A. Akram, M. Alian, R. Amslinger, M. Andreozzi, A. Armejach, N. Asmussen, B. Beckmann, S. Bharadwaj *et al.*, "The gem5 simulator: Version 20.0+," *arXiv preprint arXiv:2007.03152*, 2020.

[32] T. Odajima, Y. Kodama, M. Tsuji, M. Matsuda, Y. Maruyama, and M. Sato, "Preliminary performance evaluation of the fujitsu a64fx using hpc applications," in *2020 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2020, pp. 523–530.

[33] S. Marco-Sola, J. M. Eizenga, A. Guarracino, B. Paten, E. Garrison, and M. Moreto, "Optimal gap-affine alignment in o (s) space," *Bioinformatics*, vol. 39, no. 2, p. btad074, 2023.

[34] Q. Aguado-Puig, "WFA-GPU," Available at https://github.com/quim0/WFA-GPU, accessed: 2023-12-21.

[35] J. Pavon, I. V. Valdivieso, C. Rojas, C. Hernandez, M. Aslan, R. Figueras, Y. Yuan, J. Lindegger, M. Alser, F. Moll *et al.*, "Quetzal: Vector acceleration framework for modern genome sequence analysis algorithms," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 597–612.