GMX: Instruction Set Extensions for Fast, Scalable, and Efficient Genome Sequence Alignment

Max Doblas
Barcelona Supercomputing Center
Barcelona, Spain
max.doblas@bsc.es

Oscar Lostes-Cazorla
Barcelona Supercomputing Center
Barcelona, Spain
oscar.lostes@bsc.es

Quim Aguado-Puig
Universitat Autònoma de Barcelona
Bellaterra, Spain
quim.aguado@uab.cat

Nicholas Cebry
Cornell University
Ithaca, New York, USA
nfc35@cornell.edu

Pau Fontova-Musté
Barcelona Supercomputing Center
Barcelona, Spain
pau.fontova@bsc.es

Christopher Batten
Cornell University
Ithaca, New York, USA
cbatten@cornell.edu

Santiago Marco-Sola
Universitat Politècnica de Catalunya
Barcelona Supercomputing Center
Barcelona, Spain
santiago.marco@bsc.es

Miquel Moretó
Universitat Politècnica de Catalunya
Barcelona Supercomputing Center
Barcelona, Spain
miquel.moreto@bsc.es

ABSTRACT

Sequence alignment remains a fundamental problem in computer science with practical applications ranging from pattern matching to computational biology. The ever-increasing volumes of genomic data produced by modern DNA sequencers motivate improved software and hardware sequence alignment accelerators that scale with longer sequence lengths and high error rates without losing accuracy. Furthermore, the wide variety of use cases requiring sequence alignment demands flexible and efficient solutions that can match or even outperform expensive application-specific accelerators.

To address these challenges, we propose GMX, a set of ISA extensions that enable efficient sequence alignment computations based on dynamic programming (DP). GMX extensions provide the basic building-block operations to perform fast tile-wise computations of the DP matrix, reducing the memory footprint and allowing easy integration into widely-used algorithms and tools. Furthermore, we provide an efficient hardware implementation that integrates GMX extensions in a RISC-V-based edge system-on-chip (SoC). Compared to widely-used software implementations, our hardware-software co-design leveraging GMX extensions obtains speed-ups from 25–265×, scaling to megabyte-long sequences. Compared to domain-specific accelerators (DSA), we demonstrate that GMX-accelerated implementations demand significantly less memory bandwidth, requiring less area per processing element (PE). As a result, a single GMX-enabled core achieves a throughput per area between 0.35-0.52× that of state-of-the-art DSAs while being more flexible and reusing the core's resources. Post-place-and-route results for a GMX-enhanced SoC in 22nm technology shows that GMX extensions only account for 1.7% of the overall area while consuming just 8.47mW. We conclude that GMX extensions represent versatile and scalable ISA additions to improve the performance of genome analysis tools and other use cases that require fast and efficient sequence alignment.

CCS CONCEPTS

• **Computer systems organization → Special purpose systems**; • **Hardware → Application specific instruction set processors**; • **Applied computing → Genomics**.

KEYWORDS

ISA extensions, hardware acceleration, genomics, sequence alignment, edit-distance, bioinformatics, microarchitecture

ACM Reference Format:
Max Doblas, Oscar Lostes-Cazorla, Quim Aguado-Puig, Nicholas Cebry, Pau Fontova-Musté, Christopher Batten, Santiago Marco-Sola, and Miquel Moretó. 2023. GMX: Instruction Set Extensions for Fast, Scalable, and Efficient Genome Sequence Alignment. In *56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '23), October 28–November 01, 2023, Toronto, ON, Canada.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3613424.3614306

1 INTRODUCTION

Sequence alignment is a fundamental problem in many application domains, including information retrieval [11, 80], pattern matching [28, 81], natural language processing [40, 99], and others [30, 61, 91]. With the advent of genome sequencing technologies, sequence alignment has acquired special relevance in computational biology and genome sequence analysis [5, 62, 94, 95]. Modern sequencing machines can rapidly produce millions of relatively small DNA reads from a few hundred to a million base pairs (bps) at a low cost [4, 88, 93]. For the past decade, genomic data production has been doubling every 7 months [13], notably outpacing Moore's Law, and even surpassing other big data sources (e.g., YouTube and Twitter [100]). If this trend continues, we will soon be able to sequence billions of whole human genomes yearly, generating exabytes of raw genomic data. This increase in genomic data has been crucial for the development of population-wide genetic studies [111], diagnosis of diseases (e.g., cancer, autism, diabetes) [27] and personalized healthcare [10, 35, 42, 43], effective outbreak tracing (e.g., COVID-19, Ebola) [19, 45, 85, 110], biodiversity preservation [60, 114], and even DNA-based computing and storage systems [15, 101, 102].

Increasing production yields and sequence lengths pose a computational challenge for current genome sequence analysis tools and hardware accelerators. As a result, the performance bottleneck in genome sequence analysis is moving from the physical process of DNA sequencing to the computational post-processing and analysis. In particular, accelerating ubiquitous building blocks, like sequence alignment, has become paramount to bridging the gap between sequence data production and current computing power. However, sequence alignment algorithms rely on DP-based algorithms that scale quadratically in both execution time and memory. As a result, tools requiring sequence alignment quickly become the bottleneck of many genome sequence analyses and fail to scale with longer sequence lengths [6]. To alleviate this problem, edit-distance alignment algorithms have gained popularity as an efficient alternative for comparing low-divergence DNA sequences in tasks such as clustering [86, 112] and pre-filtering [6, 7]. Notwithstanding, genome sequencing analysis is currently limited by the computational power and memory bandwidth of existing systems.

The need for better sequence alignment algorithms has fostered extensive research on algorithms and hardware accelerators for sequence alignment, including solutions based on GPUs [1, 2, 75, 84], FPGAs [16, 24, 46, 49, 115], PIM [23, 31, 48, 57, 79], and ASICs [17, 18, 37, 67, 83, 104]. Notable solutions, like GenAx [37] and GenASM [17], have demonstrated that DSAs can deliver significant performance improvements in computing edit-distance-based sequence alignment. However, optimized software-hardware co-designs are often tailored to specific use cases (e.g., read mapping) and fail to scale to different workloads (e.g., longer sequences). In a rapidly evolving area such as genomics, it is important to design fast and flexible accelerators that can be adapted and easily integrated into various production-ready tools. **Our goal** is to design a fast, scalable, and efficient set of ISA extensions that can accelerate sequence alignment computations in genome sequence analysis and other application domains that require edit-distance alignment.

In this work, we propose GMX, a set of ISA extensions that enable fast, scalable, and efficient computation of the edit-distance sequence alignment. GMX extensions enable the calculation of entire tiles from the DP-matrix in a single operation, leading to a quadratic reduction in executed instructions with respect to tile size. As opposed to inflexible DSAs, GMX's modular and scalable design allows extending DP-based alignment algorithms to exploit GMX extensions without compromising the accuracy of the results. To compute GMX tiles efficiently, we present GMX-Tile, an extension of the bit-parallel Myers (BPM) algorithm tailored for hardware acceleration. Unlike other proposals based on the BPM [22, 53], GMX-Tile removes all the input preprocessing steps and internal lookup tables, and simplifies the bit computations per DP-element.

Additionally, we present a fast and efficient hardware implementation for the GMX extensions (GMX-AC and GMX-TB). We provide an area- and power-efficient hardware design that can be easily integrated into any conventional CPU. Unlike co-processors and stand-alone accelerators, GMX reuses the core resources (e.g., caches and memory), eliminating the need for additional memory controllers and expensive host/device data transfers. We demonstrate that GMX-enhanced implementations of widely-used algorithms for sequence alignment outperform state-of-the-art software. Challenging conventional wisdom, we present a case study demonstrating that carefully designed ISA extensions can be highly competitive with existing DSAs.

Key Results. We evaluate GMX-accelerated implementations of widely-used sequence alignment algorithms against state-of-the-art software, including classic DP algorithms (Smith-Waterman [96] and Needleman-Wunsch [82], like in KSW2/Minimap2 [65]), banded bit-parallel algorithms (BPM [77] and Edlib [108]) and hardware accelerators (like GenASM and Darwin [104]). We find that: (1) GMX extensions allow accelerating software tools by 25–183× aligning short sequences and 48–112× when aligning long and noisy sequences; (2) GMX-accelerated single core achieves a throughput per area between 0.35-0.52× that of state-of-the-art DSAs while being more flexible and requiring a minimal area overhead of 0.0216mm^2. (3) GMX extensions allow 16x memory footprint reduction while reducing the bandwidth to memory and cache pressure, enabling GMX to scale in multicore processors. In summary, this paper makes the following contributions:

- We propose GMX-ISA extensions, an efficient and flexible set of instructions to accelerate the computation of the DP-matrix for different sequence alignment algorithms. To our knowledge, GMX is the first work to propose instructions for the tile-wise computation of the DP-matrix and traceback, reducing the computational requirements and the memory footprint.
- We propose GMX-tile, an algorithmic extension of the BPM technique tailored for hardware acceleration. We present a software-hardware co-design that integrates GMX ISA extensions into three different state-of-the-art alignment algorithms to demonstrate the versatility of GMX extensions. Additionally, we open-source our software implementations and datasets to promote transparent and reproducible research.
- We present an energy- and area-efficient hardware design for the GMX extensions (GMX-AC and GMX-TB). After performing ASIC synthesis and PnR in Global Foundries 22nm technology of a RISC-V based edge SoC integrating the GMX extensions, we demonstrate that our GMX implementation requires 0.0216mm^2 (1.7% of the overall area) and 8.47mW to operate.
- We evaluate the performance of GMX-enhanced algorithms and demonstrate that our proposal outperforms state-of-the-art software implementations and matches the performance of state-of-the-art hardware accelerators. Moreover, we show that our solution scales up to 1Mbp-long sequences.

2 BACKGROUND

In this section, we provide the necessary background on genome sequence data analysis, sequence alignment, and bit-parallel techniques required for the rest of this paper.

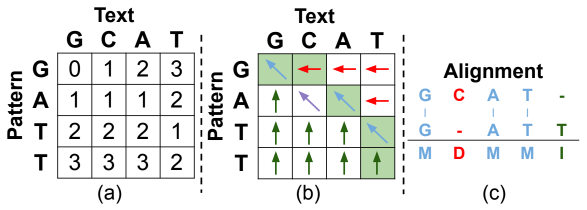


Figure 1: Example of DP-based sequence alignment. (a) Score matrix. (b) Traceback matrix. (c) Optimal alignment.

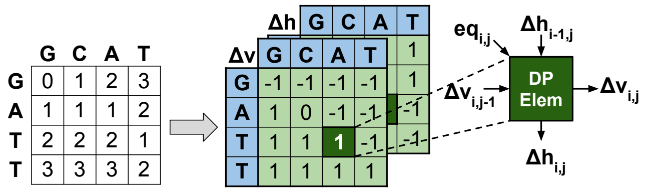


Figure 2: DP-matrix encoding using BPM's differences.

2.1 Genome Sequence Analyses

Since the completion of the first human genome [62], sequencing technologies have rapidly evolved to produce longer sequences (*reads*), increasing the data-production throughput of their machines while reducing operational costs. Modern sequencing technologies can be broadly classified into second- and third-generation sequencing technologies. Second-generation sequencing technologies, like Illumina and Ion Torrent, have dominated the market for the last decade. These technologies generate short sequences (i.e., 100 - 300bps) of high quality (i.e., <1% error rate) [71]. Third-generation sequencing technologies, like Pacific Biosciences (PacBio) and Oxford Nanopore Technologies (ONT), improve upon previous generations to produce megabase-size sequences. Unfortunately, these technologies are more error-prone, producing a typical error rate of 5%-15% [63].

Genome sequencing data requires complex and computationally intensive analyses before they can be meaningful to researchers and clinicians. Probably, the most widely-known sequencing protocol is genome resequencing [54]. A typical resequencing analysis locates the sequenced reads into a pre-existing reference genome (i.e., read mapping [4, 36]) to determine genomic variations (i.e., variant calling [55]). This analysis involves steps like indexing, seeding, pre-filtering, and sequence alignment [64, 65, 74]. Among these steps, sequence alignment is often the most time-consuming.

Beyond genome resequencing, sequence alignment is paramount for performing de-novo assembly [21], whole-genome comparisons [52], sequence clustering [103], metagenomics classification [50], and many other analyses [87]. Not surprisingly, sequence alignment has become the cornerstone of many analyses in sequence biology and genomics.

2.2 Sequence Alignment

Sequence alignment aims to determine the differences (e.g., evolutive variations, mutations, sequencing errors) between two sequences. Given a distance function (or scoring function), the optimal alignment is the sequence of operations (i.e., match, mismatch, insertion, and deletion) that transforms one sequence into the other, minimizing the distance function (or maximizing the score).

Usually, sequence alignment is computed using some variation of DP and consists of two phases: (1) DP-matrix computation and (2) alignment traceback. During the first phase, sequence alignment algorithms compute a DP-matrix of $n \times m$ elements (Figure 1.a). Then, for the traceback phase (Figure 1.b), sequence alignment algorithms trace the list of operations that led to the optimum distance (bottom-right element) back to the beginning (upper-left element); that is, the alignment between the sequences (Figure 1.c). In particular, given the pattern $p = p_0 p_1 \dots p_{n-1}$ and the text $t = t_0 t_1 \dots t_{m-1}$ sequences, the optimum edit distance ($H_{n,m}$) is given by $H_{i,j} = \min\{H_{i-1,j}+1, H_{i,j-1}+1, H_{i-1,j-1}+eq_{i,j}\}$, where $eq_{i,j} = 1$ if $p_{i-1} == t_{j-1}$ and 0 otherwise.

Sequence alignment algorithms and accelerators have been intensively studied for more than 60 years [44, 82, 90, 92, 107, 109, 113] on software [29, 70, 72, 73, 89] and hardware [17, 37, 66, 105] with applications in many areas (e.g., pattern matching [28], natural language processing [40, 97, 99], security [39, 76, 91]). Nevertheless, classical DP-based solutions require quadratic time and memory on the sequence length, posing a challenge to the scalability of these algorithms.

2.3 Bit-Parallel Techniques

Bit-parallel techniques emerged in the 80s as an effective strategy to accelerate sequence alignment algorithms. These techniques exploit bitwise operations to compute multiple elements of the DP-matrix in parallel. Most importantly, they map extremely well to conventional hardware instructions, outperforming other sequence alignment approaches in practice. As a result, these techniques have been widely adopted by many software tools and hardware accelerators.

In 1989, the first bit-parallel algorithm, called Bitap, was proposed. Bitap reformulates the problem using a DP-matrix of $n \times k$ bits, where k is the maximum edit distance supported, requiring k bits per element of the original DP-matrix. Then, it progressively computes the edit distance, recomputing the whole bit-matrix per each character of text t aligned. Thus, the alignment computations are proportional to $O(nkm)$. However, this ingenious reformulation allows computing the elements of each column independently using bitwise operations. Assuming a sufficiently large machine word of w bits ($w \geq n$), Bitap's computation reduces from $O(\frac{n}{w}km)$ to $O(km)$ and requires $7 \cdot k$ bitwise instructions per character aligned.

Bitap was designed to compute the edit distance between short sequences (i.e., that fit in a machine word) with a low alignment error, as its complexity depends on the maximum error supported k. Originally, it was not designed to compute the complete alignment, as it requires storing the m DP-matrices of $n \times k$ bits to perform the traceback phase. Bitap was recently resurrected as the bedrock algorithm for hardware accelerators, like GenASM and SeGraM [18]. These DSAs extended Bitap, using large bit-vectors and tailored bitwise logic to accelerate the computation of Bitap's $n \times k$ bit-matrix per character aligned.

Ten years later after the introduction of Bitap, Myers proposed a novel bit-parallel error-agnostic algorithm that outperforms the Bitap algorithm, scaling to longer sequences and higher error rates.

Broadly known as bit-parallel Myer's (BPM), this algorithm benefits from the observation that differences between adjacent row and column elements are limited to $\{-1, 0, +1\}$. BPM exploits this property encoding vertical differences ($\Delta v_{i,j} = H_{i,j} - H_{i-1,j}$) and horizontal differences ($\Delta h_{i,j} = H_{i,j} - H_{i,j-1}$), requiring only (2×2) bits per element of the original DP-matrix, irrespective of the alignment error (Figure 2). Then, it reformulates the classical DP equations in terms of differences (Eq. 1), where each variable can be encoded using 2 bits.

$$\begin{aligned} \Delta v_{i,j} &= min\{-eq_{i,j}, \Delta v_{i,j-1}, \Delta h_{i-1,j}\} + 1 - \Delta h_{i-1,j} \\ \Delta h_{i,j} &= min\{-eq_{i,j}, \Delta v_{i,j-1}, \Delta h_{i-1,j}\} + 1 - \Delta v_{i,j-1} \end{aligned} \tag{1}$$

The BPM proposes to compute the DP-matrix column-wise, packing the elements of each column in a bit-vector. Using bitwise operations, the BPM computes each bit-encoded column using only 17 CPU instructions per character aligned. As a result, the BPM requires to perform $O(\frac{n}{w}m)$ computations ($O(m)$ if $n < 2w$). In practice, the BPM outperforms other bit-parallel algorithms [80], scaling to longer sequences and higher error rates [116]. Variations of the BPM technique have been adopted by multiple tools in genome sequence analysis like Edlib, Daligner [78], and GEM [74].

2.4 Applications and Limitations of Edit Distance to Genome Sequence Analysis

Different distance functions (or scoring functions) are used in computational biology depending on the application and sequence data properties. When comparing protein sequences, weighted distance functions, like those implemented in Smith-Waterman (SW) or Needleman-Wunsch (NW), are preferred as they can capture meaningful biological insights. Similarly, gap-affine distances are preferred to study complex genome re-arrangements or compare highly-divergent DNA/RNA sequences. Notwithstanding, computing these complex distance functions demands a significant amount of computing and memory. Consequently, multiple tools often incorporate heuristic strategies (e.g., Z-drop, banded) to improve performance at the cost of sacrificing accuracy.

Simpler distance functions, such as edit distance, have gained attention for genome analysis that require comparing similar sequences, like high-quality DNA alignment, alignment pre-filtering, and sequence clustering. For these use cases, the edit distance function produces accurate and meaningful results while being significantly faster to compute. Recent studies (GenAX, GenASM, SeGraM, Edlib, and DAligner) support the applicability of edit distance for genome sequence analysis.

Next, we present the trade-off between throughput and accuracy obtained by the widely-used Edlib (edit distance), and KSW2 and Minimap2 (gap-affine) libraries, aligning real short (Illumina WGS) and high-quality long (PacBio HiFi) sequences from NIST's Genome in a Bottle (GIAB) consortium and PrecisionFDA Truth Challenge. Figure 3 shows results of accuracy (measuring average alignment-score deviation from the optimal gap-affine alignment) and throughput (alignments/s) executed on an Intel Xeon W-2155. Aligning high-quality datasets, we observe that edit distance alignment generally reports the same alignment as gap-affine (i.e., on-par accuracy). Moreover, computing the edit distance is significantly faster than gap-affine. For long sequences, it is even faster than gap-affine using heuristics (e.g., Banded KSW2 in Minimap2).

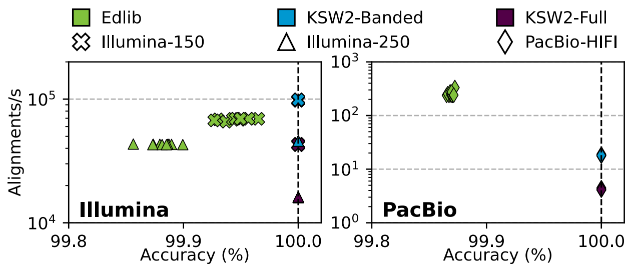


Figure 3: Speed vs. accuracy between edit and gap-affine.

3 MOTIVATION AND GOAL

Although many software and hardware sequence alignment accelerators have been proposed over the years, we find that they present different limitations. Alas, no single solution can meet the performance demands of sequence analysis tools; being fast, scalable, and efficient while producing accurate results. In this section, we discuss the limitations of current software-hardware solutions for sequence alignment.

3.1 Limitations of Existing Accelerators

Performance limitations. Classic DP-based sequence alignment algorithms are heavily restricted by quadratic time and memory requirements. Due to the intrinsic dependencies between computations of the DP-matrix, computational parallelism is significantly constrained. To alleviate this problem, many hardware accelerators rely on bit-parallel techniques due to their convenient mapping to bitwise operations. However, bit-parallel accelerators still suffer from computational bottlenecks and limited memory bandwidth. Accelerators based on Bitap obtain computational parallelism at the expense of increasing computations to $O(\frac{n}{w}km)$ and require implementing large bit-vectors ($w \geq n$) to compensate. Moreover, larger memory requirements (i.e., m DP-matrices of $n \cdot k$ bits) put higher pressure on memory bandwidth. These performance limitations escalate with increasing error rates, as Bitap's complexity is sensitive to alignment error (k). In contrast, BPM's complexity $O(\frac{n}{w}m)$ is not sensitive to the alignment error. Nevertheless, accelerators based on BPM still require performing a quadratic number of operations and storing $4 \times n \times m$ bits. On top of that, bit-parallel techniques require additional preprocessing steps that put a non-negligible toll on performance. Further, the parallelism of these solutions remains limited to processing one column at a time, presenting a reduced computational intensity and large bandwidth requirements.

Scalability limitations. The quadratic complexity of DP-based algorithms poses a challenge to scale with longer sequence lengths and higher error rates. For instance, computing the complete alignment of 10kbp-long sequences, assuming just an error of 0.1%, would require 381.4MB of memory for the classical DP algorithm, 119.2MB for Bitap, and 47.6MB for BPM. As the sequence length increases, bit-parallel based accelerators require larger hardware bit-vectors, increasing chip area and energy consumption. In the

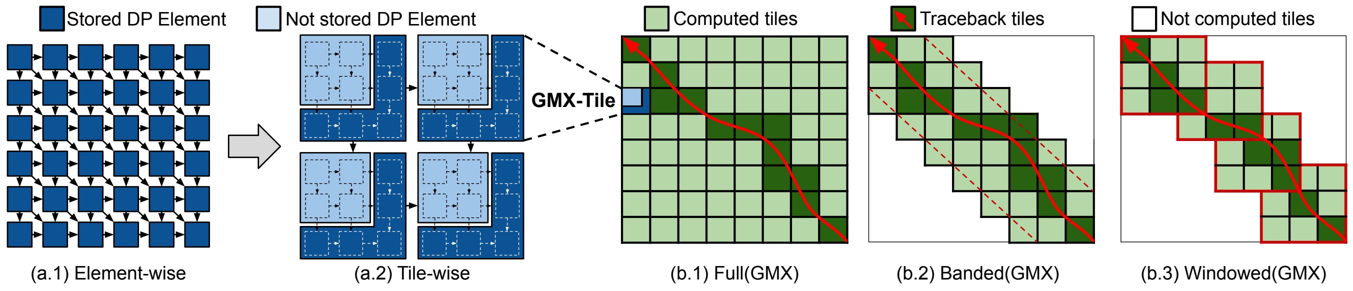


Figure 4: (a.1) Element-by-element computation of the classical DP algorithm. (a.2) Tile-by-tile computation using GMX with a tile size (T) of 3. (b) Tiles computed by different GMX-accelerated alignment algorithms.

case of Bitap, being sensitive to the alignment error poses an additional limitation to scale to high-error rates. To overcome these difficulties, many solutions are limited to computing the alignment distance (not the complete alignment) or fall back to heuristic algorithms. Some heuristic solutions limit the maximum alignment error tolerated k or restrict the computation to small portions of the DP-matrix, potentially compromising the accuracy of the results. As a result, the accuracy of these heuristics is unpredictable and often leads to sub-optimal results.

Efficiency limitations. DSAs show significant performance improvements at the expense of complex and expensive hardware designs, both in area and energy. The efficiency limitations of these accelerators raise questions about whether their benefits outperform their costs. In particular, bit-parallel accelerators implementing large bit-vectors tend to consume significant chip area and energy. Moreover, monolithic hardware accelerators overspecialized in genome resequencing invest many resources to support other functionalities (e.g., indexing and seeding) not required beyond genome mapping. Furthermore, loosely-coupled accelerators often require additional memory controllers and costly hardware designs to implement data coherence and transfers. Concerning the latter, these accelerators often incur expensive data transfers host/device that diminish the performance gains when integrated into production-ready tools.

Applicability limitations. Seeking performance improvements, DSAs tend to focus on specific use cases and input characteristics. Overspecialization often tampers the applicability of these accelerators to related use cases. In practice, integrating these accelerators into production-ready tools can be daunting, requiring extensive modification of the software stack. Monolithic accelerators are often inflexible and cannot easily be repurposed for other applications.

3.2 Our Goal

Our goal is to overcome these limitations by providing a set of flexible ISA extensions that enable fast, scalable, and efficient sequence alignment for multiple applications in genome sequence analysis and other use cases in computer science. To that end, the GMX instruction set enables fast and efficient computation of complete tiles from the DP-matrix, (1) reducing the computational requirements quadratically in the tile size, and (2) decreasing the memory footprint (as GMX only needs to store the elements at the edges of the tile). GMX extensions can be easily exploited to accelerate DP-based sequence alignment algorithms enabling control over the accuracy of the results. The GMX extensions allow for seamless integration into state-of-the-art algorithms and tools, scaling to longer sequences without sacrificing the accuracy of the results. As a result, the GMX extensions facilitate the acceleration of any tool that demands fast, scalable, and efficient sequence alignment.

4 GENOME ALIGNMENT EXTENSIONS

Classical DP-based alignment algorithms, like Smith-Water-man (SW) and Needleman-Wunsch (NW), compute the DP-matrix element by element as shown in Figure 4.a.1. This work presents the Genome alignMent eXtensions (GMX), an instruction set extension that enables the acceleration of sequence alignment by computing tile by tile the DP-matrix (Figure 4.a.2). Implementations using GMX extensions increase the computational intensity and only require storing the elements at the edge of each tile (i.e., internal tile elements are computed on-the-fly and never stored). As a result, the number of instructions is reduced quadratically with the tile size and the memory requirements are significantly reduced. In the following, we motivate the use of GMX's tile-based co-design to different sequence alignment algorithms (Section 4.1) and present GMX-Tile (Section 4.2), an efficient and hardware-friendly algorithm to compute each tile.

4.1 GMX Co-Designed Alignment Algorithms

Building upon the classical algorithms that compute the whole DP-matrix, numerous algorithmic variations have been proposed to accelerate sequence alignment. These variations mainly differ in the regions of the DP-matrix they compute (e.g., Full, Banded, and Windowed) and the underlying algorithmic strategy used to compute those regions (e.g., DP, Bitap, and BPM).

Full algorithms compute the entire DP-matrix and are often preferred when the accuracy of the results is paramount. However, Full(DP) algorithms (i.e., Full DP-matrix computation using classic DP), like SW, NW, and KSW2/Minimap2, are computationally demanding. In contrast, GMX can compute the whole DP-matrix using $\frac{nm}{T^2}$ tiles of $(T \times T)$ elements. Figure 4.b.1 shows the tiles computed by Full(GMX). As opposed to Full(DP), which requires storing all the DP-elements $(n \times m)$ to traceback the alignment, Full(GMX) only stores the DP-elements at the tile edges, requiring $T\times$ less memory. For the traceback, Full(GMX) recomputes the DP-elements between tile edges on-the-fly and retrieves the tile's traceback using a single instruction.

Banded algorithms compute a band of DP-elements along the main diagonal of the DP-matrix (Figure 4.b.2). In practice, these heuristic algorithms significantly reduce the computational cost at the risk of potentially missing the optimal alignment. Compared to Full algorithms, Banded algorithms only compute $m \times B$ DP-elements, where B is the band size. A notable example is the Edlib library, which implements the Banded algorithm, and it is often used for fast filtering and alignment in multiple genome analysis tools [25]. In the same spirit, Banded(GMX) implements the same band heuristic using GMX extensions to reduce computations to $\frac{m \times B}{T^2}$ tiles, storing $\frac{m \times B}{T}$ DP-elements.

Windowed algorithms, proposed in Darwin, implement a dynamic heuristic based on computing small overlapping windows of size $W \times W$. Starting from a window placed at the bottom-right of the DP-matrix, the algorithm computes partial tracebacks, allowing an overlap of size O between windows, until it reaches the top-left of the DP-matrix. This heuristic is implemented by GenASM, using the Bitap algorithm to compute the windows (i.e., Windowed(GenASM)). Notably, for small window sizes, like those used by GenASM, intermediate data can be stored in general-purpose registers, reducing memory accesses to those that store the resulting alignment. The Windowed algorithm can be implemented using GMX to compute overlapping windows (i.e., Windowed(GMX)). Figure 4.b.3 shows the tiles computed by Windowed(GMX), using $W = 3T$ and $O = T$.

4.2 GMX-Tile: Bit-Parallel Tile Computation

A key goal of this work is to provide a fast and resource-efficient solution to compute DP-matrix tiles in hardware. To that end, we introduce GMX-Tile, an extension of the BPM algorithm to compute the $(T \times T)$ DP-elements of a tile tailored for fast and efficient hardware acceleration.

Prior to this work, other hardware accelerators have selected the BPM algorithm to accelerate sequence alignment due to its advantageous algorithmic properties [22, 53]. However, the original BPM was designed to use general-purpose CPU instructions and, in its current formulation, is unsuitable for direct hardware implementation. BPM-based hardware accelerators mimic the original BPM's equations (Eq. 1) and internally implement 17 arithmetic operations (including an integer addition). Moreover, they require an expensive input preprocessing step to generate eq-vectors and large internal lookup tables to store them. Hence, these solutions are often limited to 2-bits encoded sequences to simplify computations and save resources.

In contrast, GMX-Tile has been designed with a hardware implementation in mind. Let $(\Delta v_{in},\Delta h_{in},eq) = (\Delta v_{i,j-1},\Delta h_{i-1,j}, eq_{i,j})$ the inputs and $(\Delta v_{out},\Delta h_{out}) = (\Delta v_{i,j},\Delta h_{i,j})$ the outputs of computing a single DP-element using BPM (Eq. 1). We identify a symmetry between the Δv and Δh computation. Thus, we can condense both computations into a single equation (Eq. 2). This way, $\Delta h_{out} = GMX_{\Delta}(\Delta h_{in}, \Delta v_{in}, eq)$ and $\Delta v_{out} = GMX_{\Delta}(\Delta v_{in}, \Delta h_{in}, eq)$.

$$GMX_{\Delta}(\Delta_0, \Delta_1, eq) = min\{-eq, \Delta_0, \Delta_1\} + 1 - \Delta_1 \tag{2}$$

Then, we encode each Δ value using 2-bits where $\Delta[0] = (\Delta{==}{+1})$ and $\Delta[1] = (\Delta{==}{-1})$. With a brute force enumeration of the 18 possible inputs (i.e., $\Delta_0, \Delta_1 \in \{-1, 0, +1\}$ and $eq \in \{0, 1\}$), one can verify the correctness of Eq. 3.

$$\begin{aligned} GMX_{\Delta}(\Delta_0, \Delta_1, eq)[0] &= !(\Delta_1[0] \mid (\Delta_0[1] \;\&\; eq \;\&\; !\Delta_1[1])) \\ GMX_{\Delta}(\Delta_0, \Delta_1, eq)[1] &= (\Delta_0[1] \mid eq) \;\&\; \Delta_1[0] \end{aligned} \tag{3}$$

GMX-Tile algorithm allows fast and efficient computation of $(T \times T)$ tiles, using 12 bit-operations per each Δ value (i.e., differential-encoded DP-element) computed. Moreover, GMX-Tile design allows the computation of antidiagonal elements in parallel. Also, GMX-Tile does not require preprocessing the input sequences, allowing character comparison of any alphabet size and removing the need for lookup tables.

In comparison, to compute a $(T \times T)$ tile, classical DP algorithms require $(5 \times T^2)$ full-integer instructions. SIMD-enabled implementations can accelerate these algorithms, reducing the number of required instructions. However, practical SIMD implementations typically demand high memory bandwidths and costly SIMD hardware units. As opposed, bit-parallel algorithms allow computing multiple DP-elements in parallel by packing them in bit-vectors. To compute a $(T \times T)$ tile, Bitap requires $(7T \times T^2)$ bit-operations, classical BPM $(17 \times T^2)$ bit-operations, and GMX-Tile only $(12 \times T^2)$ bit-operations. Regarding memory footprint, DP and SIMD algorithms encode $(T \times T)$ DP-elements as regular integers, Bitap utilizes T^3 bits per tile, and BPM uses $4T^2$ bits per tile. In contrast, GMX-Tile only requires $4T$ bits per tile as it only requires storing DP-elements at the edge of the tile.

5 GMX ISA EXTENSIONS

The GMX ISA extension provides specialized instructions to accelerate the two alignment phases: `gmx.v` and `gmx.h` to compute the $(T \times T)$ elements of a tile (Fig. 5.a) and `gmx.tb` to compute the traceback (Fig. 5.b). Let $\Delta V_i = [\Delta v_i \dots \Delta v_{i+T-1}]$ and $\Delta H_i = [\Delta h_i \dots \Delta h_{i+T-1}]$ be vectors of horizontal/vertical Δ values (i.e., differential-encoded DP-elements). GMX instructions can use standard R-type RISC-V encoding, using the reserved custom op-codes.

- `gmx.v rd, rs1, rs2`. Given ΔV_i and ΔH_i, stored in the general purpose registers rs1 and rs2, this instruction computes the tile alignment between the `gmx_text` and `gmx_pattern`, outputting ΔV_o in the register rd.
- `gmx.h rd, rs1, rs2`. Similar to `gmx.v`, this instruction computes ΔH_o and stores it in rd.

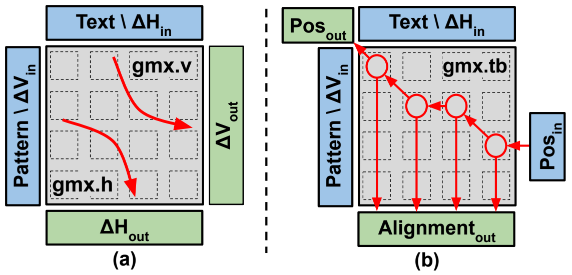


Figure 5: (a) `gmx.v` and `gmx.h` instructions to compute the ΔV_{out} and ΔH_{out} of a tile. (b) `gmx.tb` instruction to compute a tile's traceback.

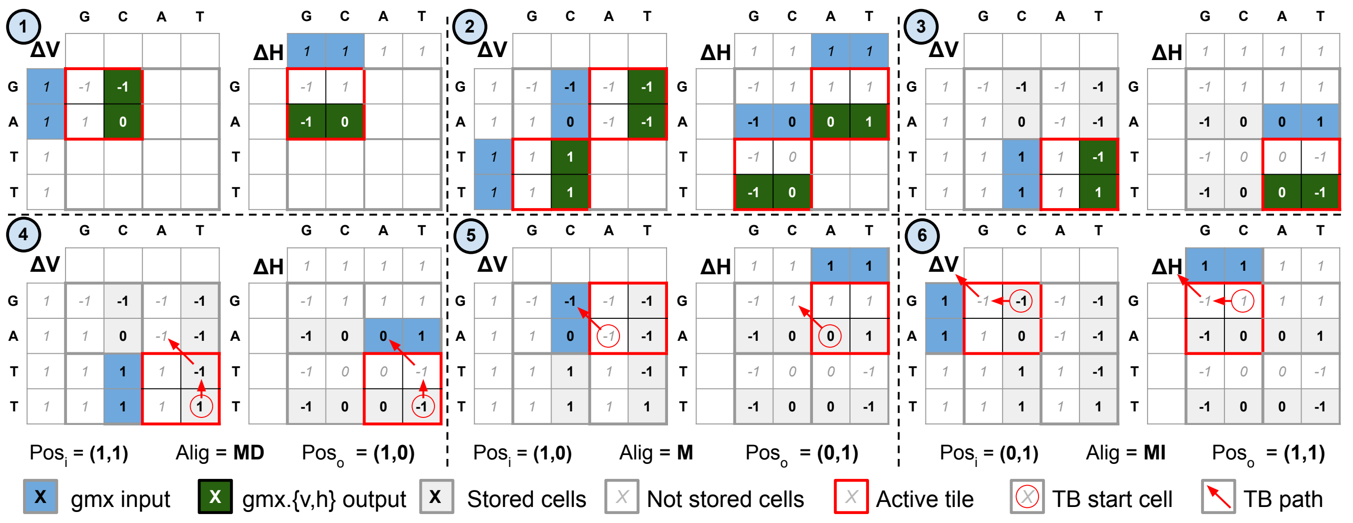


Figure 6: Alignment between two sequences ("GCAT" and "GATT") using GMX's (2×2) tiles. Steps ①-③ show the ΔV and ΔH computation performed using Algorithm 1. Steps ④-⑥ compute the traceback using ΔV and ΔH to generate the optimal alignment ("MIMMD"), as described in Algorithm 2.

- `gmx.tb rs1, rs2`. Given rs1=ΔV_i, rs2=ΔH_i, and `gmx_pos` (traceback starting position), this instruction computes the alignment traceback of a tile between the `gmx_text` and `gmx_pattern`. It produces `gmx_lo` and `gmx_hi`, containing $(2T - 1)$ 2-bit encoded alignment operations, and `gmx_pos` (traceback end position).

GMX requires five architectural state registers of $2T$ bits each (`gmx_text`, `gmx_pattern`, `gmx_pos`, `gmx_lo`, and `gmx_hi`). Architectural registers can be accessed using standard read-and-write instructions implemented in conventional ISAs, like the `csrr`/`csrw` instructions on RISC-V. GMX's *Architectural State Registers* are the following.

- `gmx_pattern`: Stores the pattern used by the GMX unit.
- `gmx_text`: Stores the text used by the GMX unit.
- `gmx_pos`: Stores the traceback's start/end position.
- `gmx_lo`: Stores the T lower bits of the 2-bit encoded traceback's alignment.
- `gmx_hi`: Stores the $(T - 1)$ higher bits of the 2-bit encoded traceback's alignment. The two most significant bits of gmx_hi store the next tile to be computed in the traceback.

Note that the size of the `gmx_pattern` and `gmx_text` architectural registers can be increased to allocate arbitrarily large alphabets (e.g., 1-byte ASCII or even 3-bytes CCCII). In addition, all the architectural registers of GMX can be renamed to allow the implementation of the ISA extension in an out-of-order processor.

It is important to emphasize that the GMX ISA extensions were designed with a simple RISC-like CPU design in mind, equipped with only one destination register port. Due to this limitation, it was necessary to design two separated instructions (`gmx.v` and `gmx.h`) to compute a tile (which introduces redundant work like the `mul` and `mulh` instructions in RISC-V). Notwithstanding, if the target CPU allowed for two destination register ports, it would be possible to merge `gmx.v` and `gmx.h` instructions, improving the efficiency and throughput of the implementation. Furthermore, if the target CPU could write two destination ports per instruction, the `gmx.tb` instruction could write the `gmx_lo` and `gmx_hi` into general-purpose registers instead of using the dedicated CSR.

5.1 Use-Case: Computing Full(GMX)

In this section, we present the Full(GMX) implementation (i.e., the classical DP-based algorithm enhanced with the GMX instructions) to motivate the simplicity and applicability of our proposal. Algorithm 1 presents the tile-wise DP-matrix computation using GMX instructions. Moreover, Figure 6 illustrates the algorithm's steps (for a tile size $T = 2$) to align the sequences "GCAT" and "GATT", computing the DP-matrix (steps ①-③) and traceback (steps ④-⑥).

Given the text, pattern, tile size T, Algorithm 1 computes of all the tiles from the DP-matrix (Figure 6, steps ①-③). Let M be a $(n/T \times m/T)$ matrix containing each tile's ΔV and ΔH vectors. The algorithm proceeds column-wise, computing each $T \times T$ tile. For that, it sets `gmx_pattern` and `gmx_text` with the proper pattern and text's chunks (using the `csrw` instruction) and loads the input differences from the upper tile ($\Delta H_{in} = M[i-1, j].h$) and left tile ($\Delta V_{in} = M[i, j-1].v$). Using `gmx.v` and `gmx.h` instructions, the algorithm generates ΔV_{out} and ΔH_{out}. Note that the algorithm does not need to store all the DP-elements in the tile, just the differences vectors corresponding to the DP-elements at the tiles' edge.

Afterwards, Figure 6, steps ④-⑥, shows the computation of the alignment traceback to retrieve the optimal sequence alignment using Algorithm 2. Starting from the bottom-right corner tile, the traceback proceeds tile-wise until it reaches the top-left corner tile of the DP-matrix. For that, the algorithm uses the `gmx.tb` instruction to compute the tile's traceback from the position set in `gmx_pos`. As a result, instruction `gmx.tb` outputs the alignment encoded in `gmx_hi` and `gmx_lo`, and updates `gmx_pos` with the traceback's ending position. This process is iterated until all the global traceback is computed.

Algorithm 1: DP-matrix computation using GMX.

```
Input: pattern, text, $T$;        # $T$ = GMX Tile size
Output: $M$
n = length of the pattern;
m = length of the text;
for j=1 to m/$T$ do # Loop pattern in $T$-chunks
    csrw gmx_text, text[$j \times T$:$T$];
    for i=1 to n/$T$ do # Loop text in $T$-chunks
        csrw gmx_pattern, pattern[$i \times T$:$T$];
        $(\Delta V_{\text{in}}, \Delta H_{\text{in}}) = (M$[i][j-1]$.v$, $M$[i-1][j]$.h)$;
        gmx.v $\Delta V_{\text{out}}$, $\Delta V_{\text{in}}$, $\Delta H_{\text{in}}$;
        gmx.h $\Delta H_{\text{out}}$, $\Delta V_{\text{in}}$, $\Delta H_{\text{in}}$;
        $M$[i][j]$.v = \Delta V_{\text{out}}$;
        $M$[i][j]$.h = \Delta H_{\text{out}}$;
    end
end
```

Algorithm 2: Traceback computation using GMX.

```
Input: pattern, text, $M$, $T$
Output: alig_vec, alig_ptr
h = m/$T$ - 1;
v = n/$T$ - 1;
csrw gmx_pos, bottom-right_cell;
csrw gmx_text, text[h];
csrw gmx_pattern, pattern[v];
while v $\geq$ 0 && h $\geq$ 0 do
    $(\Delta V_{\text{in}}, \Delta H_{\text{in}}) = (M$[i][j-1]$.v$, $M$[i-1][j]$.h)$;
    gmx.tb $\Delta V_{\text{in}}$, $\Delta H_{\text{in}}$;
    csrr alig_vec[alig_ptr++], gmx_hi;
    csrr alig_vec[alig_ptr++], gmx_lo;
    csrr nextTile, gmx_hi;
    if nextTile is $\nwarrow$ then
        csrw gmx_text, text[--h];
        csrw gmx_pattern, pattern[--v];
    else if nextTile is $\uparrow$ then
        csrw gmx_pattern, pattern[--v];
    else nextTile is $\leftarrow$
        csrw gmx_text, text[--h];
end
```

6 GMX MICROARCHITECTURE

GMX is designed to be an instruction set extension for fast and accurate genome sequence alignment. These hardware extensions must be located inside the processor pipeline and, therefore, the GMX implementation has to meet specific hardware constraints: i) a small area footprint, ii) a short execution latency (few clock cycles), iii) the need to use the processor's general-purpose registers efficiently, and iv) reach the same high-frequency as the processor. To that end, we propose a fast and efficient hardware implementation separated into two modules (GMX-AC and GMX-TB for the Tile and traceback computation, respectively) that can be seamlessly integrated into any conventional CPU.

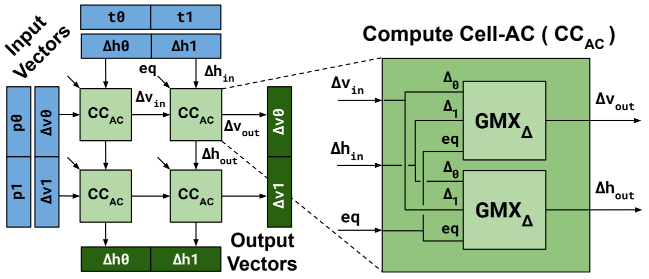


Figure 7: GMX-AC hardware design (T = 2). On the left, the matrix structure of the design. On the right, a single compute cell's structure which contains two GMX_Δ modules.

6.1 GMX-AC: Alignment Microarchitecture

The GMX-AC module takes ΔV_{in}, ΔH_{in}, gmx_pattern, and gmx_text as inputs. It generates ΔV_{out} and ΔH_{out} vectors when executing gmx.v and gmx.h instructions, respectively. We have implemented GMX-AC as a matrix of $(T \times T)$ basic Compute Cores (CC_{AC}) (Figure 7 left). Each CC_{AC} of GMX-AC computes a single DP element (Δv_{out} and Δh_{out}), using left Δv_{in}, upper Δh_{in}, and eq (Figure 7 right). The equality bit eq is generated comparing the corresponding characters from gmx_pattern and gmx_text. Internally, each CC_{AC} implements two identical GMX_Δ modules (Eq. 3). Due to the simplicity of the GMX_Δ function, our design can be implemented using a reduced number of gates, minimizing the propagation delay and the area footprint.

6.2 GMX-TB: Traceback Microarchitecture

The GMX-TB module computes the traceback operations of the alignment tile. The GMX-TB module takes ΔH, ΔV, gmx_pattern, gmx_text, and gmx_pos as inputs. It computes the tile's traceback when executing gmx.tb, storing the 2-bit encoded alignment operations in gmx_lo and gmx_hi and the traceback's end position in gmx_pos. As in previous software implementations [32], because GMX only stores the DP-elements at the edges of the tiles, GMX-TB has to recompute the internal DP-elements to compute the tile's traceback. For that, the GMX-TB module uses a matrix-like structure of GMX-TB Compute Cores (CC_{TB}) (Figure 8) similar to GMX-AC with the CC_{AC}.

Unlike GMX-AC, the dataflow goes from a CC_{TB} in the bottom or right edges towards a CC_{TB} in the top or left edges. Moreover, because the traceback can start at any element of the bottom and right of the tile, the register gmx_pos one-hot encodes the starting position of the traceback. Each CC_{TB} (Figure 8) uses a selector to discriminate which adjacent element belongs to the alignment path (i.e., match/mismatch=$\nwarrow$, insertion=$\leftarrow$, and deletion=$\uparrow$). Each CC_{TB} is connected to the three adjacent CC_{TB} (left, left-up, and up) and enables one depending on Δv, Δh, and eq. This alignment path is propagated until it reaches the left or top edges. The output position of the last CC_{TB} is stored in gmx_pos for the next tile traceback computation. Moreover, the sequence of the CC_{TB} enabled (i.e., the alignment path) is injected into the gmx_hi and gmx_lo registers.

Note that the alignment path only traverses one CC_{TB} on each antidiagonal at most. We exploit this property to simplify the GMX-TB design, storing in gmx_hi and gmx_lo the enabled CC_{TB} on

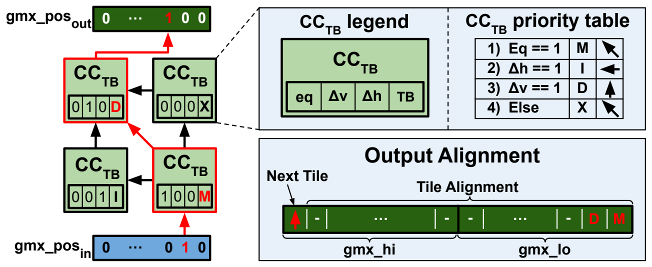


Figure 8: GMX-TB hardware design ($T = 2$). On the left, the matrix structure of the design. On the right, the CC_{TB} legend and properties, along with the output alignment.

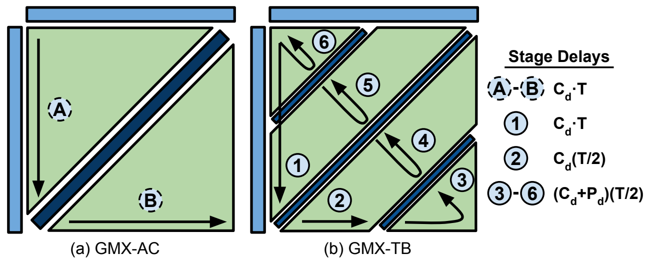


Figure 9: Segmentation strategy for GMX-AC (left) and GMX-TB (right).

each antidiagonal using 2-bits. Like the GMX-AC core modules, the GMX-TB core module can be implemented using a few gates to reduce the area and propagation delay.

6.3 Segmentation and Frequency Analysis

GMX's extensions employ the available general-purpose registers. Thus, we select a suitable tile size T that fully exploits the register's length (e.g., $T = 32$ using 64-bit scalar registers). However, large values of T require a carefully segmented design to work at modern processors' high clock frequencies.

GMX-AC Segmentation: Analyzing the critical path inside GMX-AC's design, we observe that the maximum delay paths start on the top-left cell and finish at the bottom-right cell, traversing $2T - 1$ compute cells. Let C_d be the delay of a CC_{AC}, the critical path of the whole module is $(2T - 1) \cdot C_d$ (e.g., for $T = 32$, the critical path is $63C_d$). Even for small C_d, a single-cycle implementation of GMX-AC cannot reach high frequencies.

The segmentation strategy used in this GMX-AC's design introduces segmentation registers between the matrix antidiagonals, storing up to T elements in the worst case. This design can scale to arbitrarily large values of T by adding more stages and balancing the delay across them. For instance, a two-cycle segmented design for $T = 32$ renders two stages of delay $32 \cdot C_d$ (Figure 9.a).

GMX-TB Segmentation: Analyzing the critical path inside GMX-TB's design, the maximum delay goes from the bottom-right to the top-left corner, going through $2T - 1$ CC_{TB} (each introducing P_d delay). However, the overall delay of the GMX-TB module has to account for the delay in recomputing the tile's inner DP-elements; i.e., a $((2T - 1)(C_d + P_d))$ total traceback delay.

Similarly, GMX-TB needs to be segmented to reach high-frequency operation. The GMX-TB segmentation strategy also involves using antidiagonals segmentation registers. Figure 9.b shows an example of a 4-stage segmentation of the GMX-TB module. First, the differences are computed and stored in all the segmentation registers (①-②). Then, GMX-TB computes the traceback by first calculating the differences (from top to bottom) and then computing the backtrace path (from bottom to top) for each segmented stage. (③-⑥). In practice, GMX-TB must be segmented more times to achieve the same frequency as GMX-AC (i.e., $C_d \sim P_d$). Also, note that the traceback algorithm is inherently sequential. Therefore, the GMX-TB design can be efficiently implemented using a multicycle model, reducing the design complexity.

GMX Tile Size Implications: Unsurprisingly, the tile size T has profound implications for the performance and efficiency of GMX's design. As the tile size T increases, the number of compute cores (i.e., area for CC_{AC} and CC_{TB}) and the computational throughput (DP-elements/cycle) increase quadratically. In contrast, the latency only increases linearly with T.

7 RESULTS

For experimental evaluation, we integrated the GMX extensions into the gem5 simulator [14]. Additionally, we extended a 64-bit Linux-capable RTL processor fabricated in GlobalFoundries 22nm technology node with the GMX extensions. We selected a (T=32)-design to maximize 64-bit registers' usage (i.e., GMX instructions compute 1024 DP-elements per instruction). To achieve the 1 GHz working frequency of the RTL design, we segmented GMX-AC and GMX-TB modules (as shown in Figure 9) to obtain 2 and 6 cycles operation latency, respectively.

7.1 Evaluation Methodology

Cycle-Level Simulations: We evaluated GMX using gem5 to simulate two core models using the syscall emulation execution. The first core (gem5-InOrder) features a simple single-issue in-order pipeline, while the second core (gem5-OoO) is an 8-way superscalar out-of-order similar to the Arm Neoverse V1. Both cores have private L1 (64 kB) and L2 (1 MB) caches and a shared last-level cache (LLC) of 1 MB per core. To test GMX's performance in a multicore system, we used a 16-core network-on-chip (NoC) with two DDR4 memory controllers with a peak bandwidth of 47.8 GB/s.

RTL Simulations: For the RTL-model evaluation, we selected the Sargantana [98] RISC-V edge processor (RTL-InOrder) featuring an in-order single-core, single-issue, with a non-blocking L1 (32 KB) and L2 (512 KB) caches (Table 1 shows the SoC configuration). GMX's performance results were obtained by emulating the SoC in a Xilinx Alveo U280 FPGA.

Synthesis and Physical Design Environment: The RTL-InOrder SoC design with the GMX extensions was synthesized in GlobalFoundries 22nm FD-SOI technology node. Our physical design targets post-routing clock timing 1GHz. We used the Cadence Genus tool v19.11 for the logical synthesis and the Cadence Innovus tool v19.11 for the place-and-route. To extract the design's power consumption, we used the utilization information reported from gate-level simulations of the entire SoC running the alignment benchmarks.

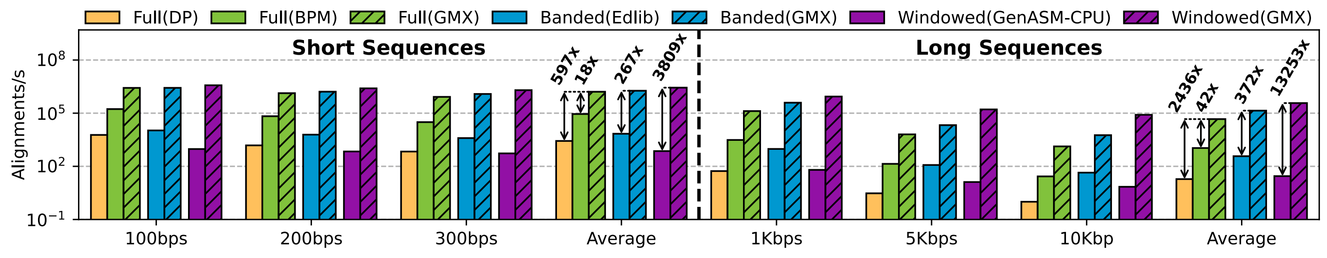


Figure 10: Gem5-InOrder core throughput comparison between software implementations.

Table 1: RTL-InOrder SoC configuration for the evaluation.

RTL-InOrder SoC Configuration & System Parameters	
Pipeline	64-bit RISC-V (RV64G), 7-stages, 128-entry bimodal predictor, 32-entry graduation list
Memory Unit	8-entry LSQ, 8-entry Store Buffer, 16 misses in flight
iTLB & dTLB	Fully associative, 16 entries per TLB
Data cache	32 KB 4-way, 3-cycle, VIPT, 2-entry MSHR
Inst. cache	16 KB 4-way, 2-cycle, VIPT
LLC	512 KBytes, 8-way set associative

Experimental Workloads: For the evaluation, we generated 5 short-sequence datasets and 11 long-sequence datasets, following the same methodology from [73]. The long-sequence datasets have different sequence lengths (1K-10K bases in increments of 1K base) and show error rates of 15%. The short-sequences datasets contain sequences of length 100bps, 150bps, 200bps, 250bps, and 300bps, with an error of 5%.

Software Implementations and Hardware Accelerators: We selected widely used sequence alignment algorithms to evaluate GMX's performance. We evaluated the Full, Banded, and Windowed algorithms leveraging the GMX instructions. For the baseline, we selected state-of-the-art sequence alignment implementations, including Full(DP) (Needleman-Wunsch [82]), Full(BPM) (BPM [77]), Banded(Edlib) (Edlib [108]), and Windowed(GenASM-CPU) (GenASM [17] open-source implementation for CPU). Additionally, we compared GMX with two state-of-the-art hardware accelerators, GenASM [17] and Darwin [104], based on the material reported by these works implemented on 28nm.

7.2 Cycle-Level Simulations

GMX on an In-Order Core: This section presents the performance results of the GMX extensions implemented on the gem5-InOrder core. Figure 10 shows the throughput (alignments per second) obtained by the baseline software implementations (Full, Banded, and Windowed algorithms) and the GMX-accelerated versions, aligning short and long sequences.

Regarding short-sequence alignment, the left side of Figure 10 shows that Full(GMX) improves the throughput 18× compared to Full(BPM) and 597× when compared to classic Full(DP). Compared to Banded algorithms, Banded(GMX) improves the throughput by 267×, while for Windowed algorithms, Windowed(GMX) improves the throughput by 3809×. Regarding long-sequence alignment (Figure 10, right side), Full(GMX) provides 42× more throughput than Full(BPM) and 2436× more throughput than Full(DP). Moreover, Banded(GMX) achieves a 372× throughput improvement compared to Banded algorithms and 13253× throughput improvement compared to Windowed algorithms. Note that GenASM-CPU is a hardware-oriented algorithm not designed to be executed on a CPU.

Overall, these performance improvements result from (1) GMX computing 1024 DP-elements per instruction and (2) only storing 63 DP-elements per tile, whereas other implementations require multiple instructions and more memory to perform the same computation. Interestingly, GMX achieves larger performance improvements when aligning longer sequences due to its more efficient memory usage.

GMX on an Out-of-Order Core: Since GMX is designed as a core extension, the performance improvement depends on the core design. As shown in Figure 11, the throughput obtained using a wide out-of-order (gem5-OoO) is significantly larger than that obtained with the in-order core (gem5-InOrder). Also, we observe a consistent performance speed-up between the baseline software and the GMX-enhanced implementations. In particular, using gem5-OoO core with GMX can lead to a 2.4–6.4× increase in performance compared to the in-order design.

Multicore Scalability and Bandwidth Analysis: We integrated GMX into a multicore by extending each core with one GMX module to increase the throughput of a single gem5-OoO core.

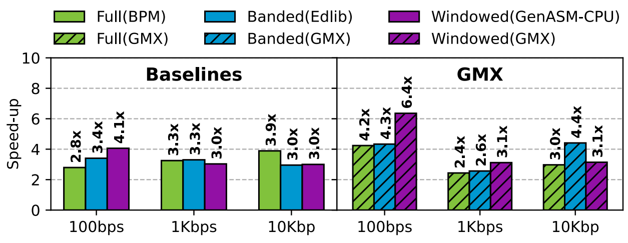


Figure 11: Throughput improvement between a gem5-InOrder and gem5-OoO core.

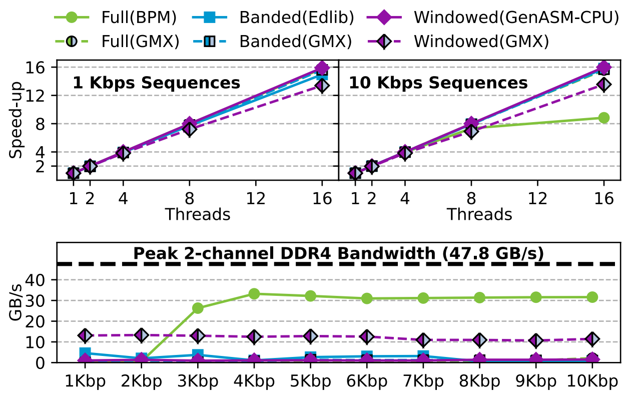


Figure 12: Multithread scalability (top) and memory bandwidth (bottom) on a 16-core processor (gem5-OoO).

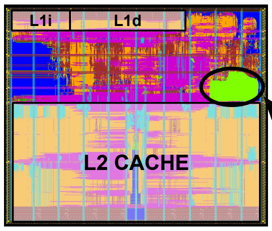
We implemented a data-parallelism strategy (a.k.a. inter-sequence) since each sequence pair can be aligned independently. Multicore results are shown in Figure 12 where the top panel shows the speed-up achieved using different numbers of threads (compared to single-thread execution), and the bottom panel shows the memory bandwidth required for 16-thread executions aligning different sequence lengths. All the implementations demonstrate linear speedup increasing the number of threads, except for Full(BPM) and Windowed(GMX).

In the case of Full(BPM), a high-memory bandwidth is required to read and write the DP-matrices. For small sequence lengths ($\leq$ 1Kbp), DP-matrices can be stored inside the caches. However, for longer sequences (> 10Kbp), the DP-matrices do not fit in caches, and thus the memory bandwidth limits the performance scaling. This memory bandwidth limitation is reflected in the lower panel of Figure 12, where the bandwidth needed from the DDR4 controllers exceeds 65% of the peak capacity.

In the case of Windowed(GMX), the performance scaling increasing the number of threads is nearly optimal. Due to the Windowed strategy and GMX's efficient design, Windowed(GMX) requires minimal computation per character aligned, increasing the pressure on the memory bandwidth (Figure 12, lower panel). In turn, this results in contention in the cache hierarchy, increasing memory request's latency. Since Windowed(GMX) suffers from load-to-load dependencies, the performance on multiple cores slightly decreases.

7.3 RTL Implementation

Area, Frequency, and Power Analysis: The right panel of Figure 13 shows the area and power breakdown of each GMX's module after the place and route sign-off at 1 GHz when implemented in the RTL-InOrder. The area overhead from GMX is 0.0216mm^2 where 0.008 mm^2 corresponds to the GMX-AC module and 0.0108mm^2 to the GMX-TB module. The silicon area occupied by the GMX extensions only represents 1.7% of the entire SoC. Interestingly, each GMX-AC and GMX-TB module consumes an area similar to that of a 2-cycle 64-bit integer multiplier. Regarding power consumption, GMX's modules increase by 8.47 mW the SoC power consumption (2.1% of the total power consumption). The left panel



Module	Area (mm^2)	Power (mW)
GMX-AC	0.0087	3.01
GMX-TB	0.0108	3.22
Total GMX	**0.0216**	**8.47**
64-bits Mul	0.0091	2.09
Total SoC	**1.2437**	**397.44**

Figure 13: Place and Route (left) and area/power breakdown (right) of the SoC.

Figure 13 depicts the floorplan of the chip where GMX's modules are highlighted in green. It is important to note that the fillers cells inside GMX's modules are also highlighted.

GMX RTL-InOrder Performance: Figure 10 shows the throughput achieved by the baseline software implementations (Full, Banded, and Windowed algorithms) and the GMX-accelerated versions executed on the RTL design (RTL-InOrder). The performance results are consistent with those obtained by the gem5 simulations (gem5-InOrder core). On edge devices, like the RTL-InOrder core, equipped with a limited memory hierarchy, GMX's reduction in memory usage increases the performance benefits. For instance, we observe that Full(BPM) executions are strongly limited by the memory bandwidth on the RTL SoC, whereas GMX-enhanced implementation Full(GMX) alleviates memory contention, providing an average improvement on the throughput of 45.2 (1.5× more than on the gem5-InOrder).

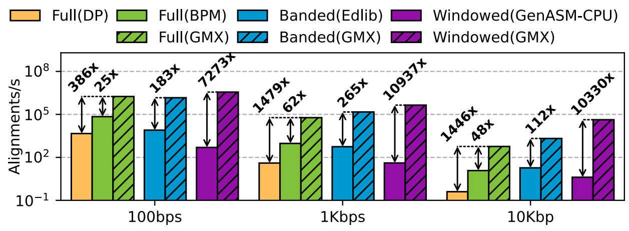


Figure 14: RTL-InOrder throughput comparison between software implementations.

7.4 GMX and other Specialized Accelerators

For the comparison with DSAs, we selected the state-of-the-art DSAs Darwin and GenASM. For a fair comparison, we compared a single RTL-InOrder core (equipped with one GMX PE), one GenASM vault (i.e., PE), and one Darwin's GACT PE with a 64-element array. We used the same Windowed algorithm used by Darwin and GenASM ($W = 96$ and $O = 32$). Due to the Windowed strategy, none of these executions is limited memory bandwidth. In particular, GMX benefits from the cache hierarchy inside the RTL SoC, while GenASM and GACT exploit internal SRAMs.

Figure 15 shows the alignment throughput per PE obtained by the three hardware accelerators, aligning short and long sequences. On average, GMX performs 1.3–1.9× better than GenASM and 7.2–16.2× better than Darwin. Yet, GMX introduces a minimal silicon

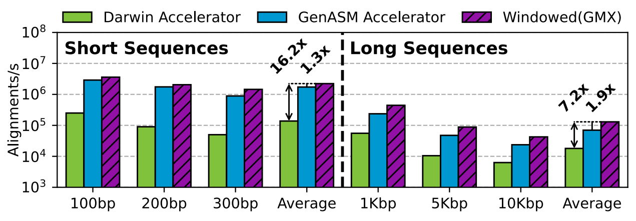


Figure 15: Hardware accelerators' throughput comparison with GMX on the RTL-InOrder.

area overhead to the existing SoC design (15.46× less extra area than GenASM and 26.29× less than Darwin).

To demonstrate GMX's scalability, we aligned 1Mbps-long sequences with 15% of error using Banded(GMX) and Windowed(GMX) implementations using the RTL-InOrder core. We excluded Full(GMX) from this evaluation as it would require more than 10GB of memory and the RTL SoC is limited to 1GB. In this scenario, Banded(GMX) achieves 20 alignments/s, while Windowed(GMX) reaches 374 alignments/s. Compared to the GenASM accelerator, we achieve 1.58× more throughput without the need to modify any design parameter or increase internal SRAM memories. Unlike traditional loosely-coupled co-processors and accelerators, leveraging GMX ISA extensions eliminates data transfers host/device (and its energy cost) and allows exploiting data locality in sequence analysis pipelines.

8 RELATED WORK

Due to its importance, many works have explored the acceleration of sequence alignment on general-purpose hardware. Many proposals are focused on algorithmic improvements [8, 65, 108], exploitation of SIMD instructions [29, 33], multicore parallel-processing [41, 106], and GPUs acceleration [47, 68, 69]. In the same spirit as GMX, Nvidia proposed the DPX instruction set extension to accelerate DP-matrix computations on GPU. Unlike GMX, DPX extensions are limited to computing a single DP-element per instruction, exploiting the multiple computing units of the GPU to perform DP-matrix computations in parallel.

Motivated by the critical need for faster solutions, many domain-specific hardware accelerators have been proposed [24, 26, 51, 56, 59] to accelerate sequence alignment. Relevant studies include SeedEx [38], GenAx, Darwin, GenASM, and SeGraM. GenAx and SeedEx (Banded) propose using an FPGA-based accelerator based on a hardware automaton to calculate the edit distance. On the other hand, GenASM and SeGraM (Windowed) propose ASIC accelerators based on the Bitap algorithm (edit distance). Darwin (Windowed) suggests a specialized ASIC design for accelerating gap-affine alignment through heuristics. Unlike GMX, these accelerators are designed as co-processors outside the CPU pipeline.

Other works have explored using processing in memory (PIM) for sequence alignment [9, 57, 58]. Notably, RAPID [48] accelerates the DP-matrix computation by processing complete antidiagonals in parallel. Similarly, BioHD [117] uses hyper-dimensional exploiting PIM architectures parallelism. While these architectures often share part of the memory hierarchy with the CPU (as GMX does), they are not integrated into the CPU pipeline.

Table 2: Peak GCUPS (PGCUPS) per processing engine (PE) reported by each study. †Gap-affine implementations.

Study	Device	PE	Area/PE	$\frac{\text{PGCUPS}}{\text{PE}}$
GMX Unit	**ASIC**	**1 PE**	**0.02mm^2**	**1024.0**
Core+GMX	**ASIC**	**1 PE**	**1.24mm^2**	**1024.0**
GenASM [17]	ASIC	32 PE	0.33mm^2	64.0
ABSW [66]	ASIC	1 PE	5.51mm^2	61.4
GenAX [37]	ASIC	4 PE	1.34mm^2	112.0
Darwin [104]	ASIC	64 PE	1.34mm^2	†54.2
ASAP [12]	FPGA	1 PE	277K LUTs	51.2
FPGASW [34]	FPGA	1 PE	58K LUTs	†105.9
DPX	GPU	132 SM	–	†42.4
GASAL2 [3]	GPU	28 SM	–	†2.3
BPM-GPU [20]	GPU	8 SM	–	287.5
NVBio	GPU	15 SM	–	66.6

Comparing different alignment accelerators is a difficult task due to the differences in the algorithms, heuristics, architectures, and physical technologies. Notwithstanding, GCUPS (Giga Cells Updated Per Second) is a commonly used metric to provide a measure of peak performance, reporting the maximum DP-elements that a solution is capable of computing per second. Table 2 shows the most notable accelerators evaluated under this metric, considering the number of Processing Engines (PE). Overall, GXM offers the highest GCUPS per PE compared to other state-of-the-art proposals. This is largely due to the highly efficient implementation of the GMX_Δ modules, which enables GMX to compute 1024 DP-elements per cycle.

9 CONCLUSIONS

In this paper, we present the Genome alignMent eXtensions (GMX), an instruction set extension that enables the acceleration of sequence alignment by tile-wise computing the DP-matrix. Moreover, we propose an area- and energy-efficient hardware implementation of GMX that can be integrated into any CPU. After integrating GMX in an in-order RISC-V edge processor, we demonstrate that GMX-accelerated algorithms outperform state-of-the-art software tools and domain-specific accelerators both in performance and efficiency without degrading accuracy and scalability.

Undoubtedly, sequence alignment will remain as a central component of many genome sequence alignment applications. We expect that GMX will pave the way for fast, scalable, accurate, and efficient genome sequence analysis tools. Furthermore, we hope that this work will contribute to the discussion on the potential benefits of domain-specific ISA extensions in future computer architectures.

ACKNOWLEDGMENTS

This work has been partially supported by the European HiPEAC Network of Excellence, by the Spanish Ministry of Science and Innovation [PID2019-107255GB-C21, TED2021-132634A-I00, and PID2020-113614RB-C21], by the Generalitat de Catalunya [2021-SGR-00763], by the European Union within the framework of the ERDF of Catalonia 2014-2020 under the DRAC project [001-P-001723] and by the European NextGenerationEU/PRTR, by Lenovo-BSC Contract-Framework Contract (2022), by National Science Foundation (NSF) within the PPoSS Award #2118709, and by the eProcessor project which received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956702. The JU receives support from the European Union's Horizon 2020 research and innovation programme and the respective national research organisations from Spain (PCI2021-121991/ MCIN/AEI/10.13039/501100011033), Sweden, Greece, Italy, France, and Germany. The eProcessor project is also co-funded by the UE NextGenerationEU/PRTR. M. Doblas has been supported through an FPU fellowship [FPU20/04076]; Q. Aguado-Puig has been supported by PRE2021-101059 (founded by MCIN/AEI/10.13039/50110 0011033 and FSE+).

REFERENCES

[1] Quim Aguado-Puig, Santiago Marco-Sola, Juan Carlos Moure, David Castells-Rufas, Lluc Alvarez, Antonio Espinosa, and Miquel Moreto. 2022. Accelerating edit-distance sequence alignment on GPU using the wavefront algorithm. *IEEE Access* 10 (2022), 63782–63796.

[2] Quim Aguado-Puig, Santiago Marco-Sola, Juan Carlos Moure, Christos Matzoros, David Castells-Rufas, Antonio Espinosa, and Miquel Moreto. 2022. WFA-GPU: Gap-affine pairwise alignment using GPUs. *bioRxiv* (2022).

[3] Nauman Ahmed, Jonathan Lévy, Shanshan Ren, Hamid Mushtaq, Koen Bertels, and Zaid Al-Ars. 2019. GASAL2: a GPU accelerated sequence alignment library for high-throughput NGS data. *BMC bioinformatics* 20, 1 (2019), 1–20.

[4] Mohammed Alser, Zülal Bingöl, Damla Senol Cali, Jeremie Kim, Saugata Ghose, Can Alkan, and Onur Mutlu. 2020. Accelerating genome analysis: A primer on an ongoing journey. *IEEE Micro* 40, 5 (2020), 65–75.

[5] Mohammed Alser, Joel Lindegger, Can Firtina, Nour Almadhoun, Haiyu Mao, Gagandeep Singh, Juan Gomez-Luna, and Onur Mutlu. 2022. From molecules to genomic variations: Accelerating genome analysis via intelligent algorithms and architectures. *Computational and Structural Biotechnology Journal* (2022).

[6] Mohammed Alser, Joel Lindegger, Can Firtina, Nour Almadhoun, Haiyu Mao, Gagandeep Singh, Juan Gomez-Luna, and Onur Mutlu. 2022. Going from molecules to genomic variations to scientific discovery: intelligent algorithms and architectures for intelligent genome analysis. *arXiv preprint arXiv:2205.07957* (2022).

[7] Mohammed Alser, Taha Shahroodi, Juan Gómez-Luna, Can Alkan, and Onur Mutlu. 2020. SneakySnake: a fast and accurate universal genome pre-alignment filter for CPUs, GPUs and FPGAs. *Bioinformatics* 36, 22-23 (2020), 5282–5290.

[8] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. 1990. Basic local alignment search tool. *Journal of Molecular Biology* (1990).

[9] Shaahin Angizi, Jiao Sun, Wei Zhang, and Deliang Fan. 2020. PIM-Aligner: A processing-in-MRAM platform for biological sequence alignment. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 1265–1270.

[10] Euan A Ashley. 2016. Towards precision medicine. *Nature Reviews Genetics* 17, 9 (2016), 507–522.

[11] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. 1999. *Modern information retrieval*. Vol. 463. ACM press New York.

[12] Subho Sankar Banerjee, Mohamed El-Hadedy, Jong Bin Lim, Zbigniew T. Kalbarczyk, Deming Chen, Steven S. Lumetta, and Ravishankar K. Iyer. 2019. ASAP: Accelerated short-read alignment on programmable hardware. *IEEE Trans. Comput.* 68, 3 (mar 2019), 331–346.

[13] Bonnie Berger, Noah M Daniels, and Y William Yu. 2016. Computational biology in the 21st century: Scaling with compressive algorithms. *Commun. ACM* 59, 8 (2016), 72–80.

[14] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. 2011. The gem5 simulator. *ACM SIGARCH Computer Architecture News* 39, 2 (2011), 1–7.

[15] James Bornholt, Randolph Lopez, Douglas M Carmean, Luis Ceze, Georg Seelig, and Karin Strauss. 2016. A DNA-based archival storage system. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*. 637–649.

[16] Liangwei Cai, Qi Wu, Tongsheng Tang, Zhi Zhou, and Yuan Xu. 2019. A design of FPGA acceleration system for Myers bit-vector based on OpenCL. In *2019 International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS)*. IEEE, 305–312.

[17] Damla Senol Cali, Gurpreet S Kalsi, Zülal Bingöl, Can Firtina, Lavanya Subramanian, Jeremie S Kim, Rachata Ausavarungnirun, Mohammed Alser, Juan Gomez-Luna, Amirali Boroumand, et al. 2020. Genasm: A high-performance, low-power approximate string matching acceleration framework for genome sequence analysis. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 951–966.

[18] Damla Senol Cali, Konstantinos Kanellopoulos, Joël Lindegger, Zülal Bingöl, Gurpreet S Kalsi, Ziyi Zuo, Can Firtina, Meryem Banu Cavlak, Jeremie Kim, Nika Mansouri Ghiasi, et al. 2022. SeGraM: a universal hardware accelerator for genomic sequence-to-graph and sequence-to-sequence mapping. *arXiv preprint arXiv:2205.05883* (2022).

[19] Darlan S Candido, Ingra M Claro, Jaqueline G De Jesus, William M Souza, Filipe RR Moreira, Simon Dellicour, Thomas A Mellan, Louis Du Plessis, Rafael HM Pereira, Flavia CS Sales, et al. 2020. Evolution and epidemic spread of SARS-CoV-2 in Brazil. *Science* 369, 6508 (2020), 1255–1260.

[20] Alejandro Chacón, Santiago Marco-Sola, Antonio Espinosa, Paolo Ribeca, and Juan Carlos Moure. 2014. Thread-cooperative, bit-parallel computation of levenshtein distance on GPU. In *Proceedings of the 28th ACM international conference on Supercomputing*. 103–112.

[21] Mark JP Chaisson, Richard K Wilson, and Evan E Eichler. 2015. Genetic variation and the de novo assembly of human genomes. *Nature Reviews Genetics* 16, 11 (2015), 627–640.

[22] Chuan-Yu Chen, Shih-Hao Huang, and Yi-Chang Lu. 2022. A Hardware Accelerator for Long Sequence Alignment with the Bit-Vector Scoring Scheme and Divide-and-Conquer Traceback. In *2022 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. IEEE, 467–471.

[23] Fan Chen, Linghao Song, Yiran Chen, et al. 2020. PARC: A processing-in-CAM architecture for genomic long read pairwise alignment using ReRAM. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 175–180.

[24] Peng Chen, Chao Wang, Xi Li, and Xuehai Zhou. 2014. Accelerating the next generation long read mapping with the FPGA-based system. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 11, 5 (2014), 840–852.

[25] Ying Chen, Fan Nie, Shang-Qian Xie, Ying-Feng Zheng, Qi Dai, Thomas Bray, Yao-Xin Wang, Jian-Feng Xing, Zhi-Jian Huang, De-Peng Wang, et al. 2021. Efficient assembly of nanopore reads via highly accurate and intact error correction. *Nature Communications* 12, 1 (2021), 60.

[26] Yu-Ting Chen, Jason Cong, Jie Lei, and Peng Wei. 2015. A novel high-throughput acceleration engine for read alignment. In *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*. 199–202.

[27] Lynda Chin, Jannik N Andersen, and P Andrew Futreal. 2011. Cancer genomics: from discovery science to personalized medicine. *Nature Medicine* 17, 3 (2011), 297–303.

[28] Peter Christen. 2006. A comparison of personal name matching: Techniques and practical issues. In *Sixth IEEE International Conference on Data Mining-Workshops (ICDMW'06)*. IEEE, 290–294.

[29] Jeff Daily. 2016. Parasail: SIMD C library for global, semi-global, and local pairwise sequence alignments. *BMC Bioinformatics* 17, 1 (2016), 1–11.

[30] Hercules Dalianis. 2018. *Clinical text mining: Secondary use of electronic patient records*. Springer Nature.

[31] Safaa Diab, Amir Nassereldine, Mohammed Alser, Juan Gómez Luna, Onur Mutlu, and Izzat El Hajj. 2022. High-throughput pairwise alignment with the wavefront algorithm using processing-in-memory. *arXiv preprint arXiv:2204.02085* (2022).

[32] Jordan M Eizenga and Benedict Paten. 2022. Improving the time and space complexity of the WFA algorithm and generalizing its scoring. *bioRxiv* (2022), 2022–01.

[33] Michael Farrar. 2006. Stripe Smith-Waterman speeds database searches six times over other SIMD implementations. *Bioinformatics* (2006).

[34] Xia Fei, Zou Dan, Lu Lina, Man Xin, and Zhang Chunlei. 2018. FPGASW: accelerating large-scale Smith–Waterman sequence alignment application with backtracking on FPGA linear systolic array. *Interdisciplinary Sciences: Computational Life Sciences* 10, 1 (2018), 176–188.

[35] Mauricio Flores, Gustavo Glusman, Kristin Brogaard, Nathan D Price, and Leroy Hood. 2013. P4 medicine: how systems medicine will transform the healthcare sector and society. *Personalized Medicine* 10, 6 (2013), 565–576.

[36] Nuno A Fonseca, Johan Rung, Alvis Brazma, and John C Marioni. 2012. Tools for mapping high-throughput sequencing data. *Bioinformatics* 28, 24 (2012), 3169–3177.

[37] Daichi Fujiki, Arun Subramaniyan, Tianjun Zhang, Yu Zeng, Reetuparna Das, David Blaauw, and Satish Narayanasamy. 2018. GenAx: A genome sequencing accelerator. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 69–82.

[38] Daichi Fujiki, Shunhao Wu, Nathan Ozog, Kush Goliya, David Blaauw, Satish Narayanasamy, and Reetuparna Das. 2020. SeedEx: A genome sequencing accelerator for optimal alignments in subminimal space. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 937–950.

[39] Ji Gao, Jack Lanchantin, Mary Lou Soffa, and Yanjun Qi. 2018. Black-box generation of adversarial text sequences to evade deep learning classifiers. In *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE, 50–56.

[40] Albert Gatt and Emiel Krahmer. 2018. Survey of the state of the art in natural language generation: Core tasks, applications and evaluation. *Journal of Artificial Intelligence Research* 61 (2018), 65–170.

[41] Evangelos Georganas, Aydin Buluç, Jarrod Chapman, Leonid Oliker, Daniel Rokhsar, and Katherine Yelick. 2015. merAligner: A fully parallel sequence aligner. In *2015 IEEE International Parallel and Distributed Processing Symposium*. 561–570.

[42] Geoffrey S Ginsburg and Kathryn A Phillips. 2018. Precision medicine: from science to value. *Health Affairs* 37, 5 (2018), 694–701.

[43] Geoffrey S Ginsburg and Huntington F Willard. 2009. Genomic and personalized medicine: foundations and applications. *Translational Research* 154, 6 (2009), 277–287.

[44] Osamu Gotoh. 1982. An improved algorithm for matching biological sequences. *Journal of Molecular Biology* 162, 3 (1982), 705–708.

[45] Alexander L Greninger, Samia N Naccache, Scot Federman, Guixia Yu, Placide Mbala, Vanessa Bres, Doug Stryke, Jerome Bouquet, Sneha Somasekar, Jeffrey M Linnen, et al. 2015. Rapid metagenomic identification of viral pathogens in clinical samples by real-time nanopore sequencing analysis. *Genome Medicine* 7, 1 (2015), 1–13.

[46] Venkateshwarlu Yellaswamy Gudur, Sidharth Maheshwari, Swati Bhardwaj, Amit Acharyya, and Rishad Shafik. 2022. Hardware-algorithm codesign for fast and energy efficient approximate string matching on FPGA for computational biology. In *2022 44th Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*. IEEE, 87–90.

[47] Sree Charan Gundabolu, TN Vijaykumar, and Mithuna Thottethodi. 2021. FastZ: accelerating gapped whole genome alignment on GPUs. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–13.

[48] Saransh Gupta, Mohsen Imani, Behnam Khaleghi, Venkatesh Kumar, and Tajana Rosing. 2019. RAPID: A ReRAM processing in-memory architecture for DNA sequence alignment. In *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 1–6.

[49] Abbas Haghi, Santiago Marco-Sola, Lluc Alvarez, Dionysios Diamantopoulos, Christoph Hagleitner, and Miquel Moreto. 2021. An FPGA accelerator of the wavefront algorithm for genomics pairwise alignment. In *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 151–159.

[50] Jo Handelsman. 2004. Metagenomics: application of genomics to uncultured microorganisms. *Microbiology and molecular biology reviews* 68, 4 (2004), 669–685.

[51] Brandon Harris, Arpith C. Jacob, Joseph M. Lancaster, Jeremy Buhler, and Roger D. Chamberlain. 2007. A banded Smith-Waterman FPGA accelerator for Mercury BLASTP. In *2007 International Conference on Field Programmable Logic and Applications*. 765–769.

[52] Robert S Harris. 2007. *Improved pairwise alignment of genomic DNA*. The Pennsylvania State University.

[53] Jörn Hoffmann, Dirk Zeckzer, and Martin Bogdan. 2016. Using FPGAs to accelerate Myers bit-vector algorithm. In *XIV Mediterranean Conference on Medical and Biological Engineering and Computing 2016: MEDICON 2016, March 31st-April 2nd 2016, Paphos, Cyprus*. Springer, 535–541.

[54] Xuehui Huang, Qi Feng, Qian Qian, Qiang Zhao, Lu Wang, Ahong Wang, Jianping Guan, Danlin Fan, Qijun Weng, Tao Huang, et al. 2009. High-throughput genotyping by whole-genome resequencing. *Genome Research* 19, 6 (2009), 1068–1076.

[55] Sohyun Hwang, Eiru Kim, Insuk Lee, and Edward M Marcotte. 2015. Systematic comparison of variant calling pipelines using gold standard personal exome variants. *Scientific Reports* 5, 1 (2015), 1–8.

[56] Xianyang Jiang, Xinchun Liu, Lin Xu, Peiheng Zhang, and Ninghui Sun. 2007. A reconfigurable accelerator for Smith–Waterman algorithm. *IEEE Transactions on Circuits and Systems II: Express Briefs* 54, 12 (2007), 1077–1081.

[57] Roman Kaplan, Leonid Yavits, and Ran Ginosasr. 2020. Bioseal: In-memory biological sequence alignment accelerator for large-scale genomic data. In *Proceedings of the 13th ACM International Systems and Storage Conference*. 36–48.

[58] S. Karen Khatamifard, Zamshed Chowdhury, Nakul Pande, Meisam Razaviyayn, Chris H. Kim, and Ulya R. Karpuzcu. 2021. GeNVoM: read mapping near non-volatile memory. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* (2021), 1–1.

[59] Yeseong Kim, Mohsen Imani, Niema Moshiri, and Tajana Rosing. 2020. GenieHD: Efficient DNA pattern matching accelerator using hyperdimensional computing. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 115–120.

[60] Henrik Krehenwinkel, Aaron Pomerantz, and Stefan Prost. 2019. Genetic biomonitoring and biodiversity assessment using portable sequencing technologies: current uses and future directions. *Genes* 10, 11 (2019), 858.

[61] Karen Kukich. 1992. Techniques for automatically correcting words in text. *Acm Computing Surveys (CSUR)* 24, 4 (1992), 377–439.

[62] Eric S Lander, Lauren M Linton, Bruce Birren, Chad Nusbaum, Michael C Zody, Jennifer Baldwin, Keri Devon, Ken Dewar, Michael Doyle, William Fitzhugh, et al. 2001. Initial sequencing and analysis of the human genome. *Nature* 409, 6822 (2001), 860–921.

[63] Dandan Lang, Shilai Zhang, Pingping Ren, Fan Liang, Zongyi Sun, Guanliang Meng, Yuntao Tan, Xiaokang Li, Qihua Lai, Lingling Han, et al. 2020. Comparison of the two up-to-date sequencing technologies for genome assembly: HiFi reads of Pacific Biosciences Sequel II system and ultralong reads of Oxford Nanopore. *Gigascience* 9, 12 (2020), giaa123.

[64] Ben Langmead and Steven L Salzberg. 2012. Fast gapped-read alignment with Bowtie 2. *Nature Methods* 9, 4 (2012), 357–359.

[65] Heng Li. 2018. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics* 34, 18 (2018), 3094–3100.

[66] Yi-Lun Liao, Yu-Cheng Li, Nae-Chyun Chen, and Yi-Chang Lu. 2018. Adaptively banded Smith-Waterman algorithm for long reads and its hardware accelerator. In *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 1–9.

[67] Joël Lindegger, Damla Senol Cali, Mohammed Alser, Juan Gómez-Luna, Nika Mansouri Ghiasi, and Onur Mutlu. 2022. Scrooge: a fast and memory-frugal genomic sequence aligner for CPUs, GPUs, and ASICs. *arXiv preprint arXiv:2208.09985* (2022).

[68] Yongchao Liu and Bertil Schmidt. 2015. GSWABE: faster GPU-accelerated sequence alignment with optimal alignment Retrieval for Short DNA Sequences. *Concurrency and Computation: Practice and Experience* 27, 4 (mar 2015), 958–972.

[69] Yongchao Liu, Adrianto Wirawan, and Bertil Schmidt. 2013. CUDASW++ 3.0: accelerating Smith-Waterman protein database search by coupling CPU and GPU SIMD instructions. *BMC Bioinformatics* (2013).

[70] Joshua Loving, Yozen Hernandez, and Gary Benson. 2014. BitPAl: a bit-parallel, general integer-scoring sequence alignment algorithm. *Bioinformatics* 30, 22 (2014), 3166–3173.

[71] Leigh J Manley, Duanduan Ma, and Stuart S Levine. 2016. Monitoring error rates in Illumina sequencing. *Journal of biomolecular Techniques: JBT* 27, 4 (2016), 125.

[72] Santiago Marco-Sola, Jordan M Eizenga, Andrea Guarracino, Benedict Paten, Erik Garrison, and Miquel Moreto. 2022. Optimal gap-affine alignment in O (s) space. *bioRxiv* (2022).

[73] Santiago Marco-Sola, Juan Carlos Moure, Miquel Moreto, and Antonio Espinosa. 2021. Fast gap-affine pairwise alignment using the wavefront algorithm. *Bioinformatics* 37, 4 (2021), 456–463.

[74] Santiago Marco-Sola, Michael Sammeth, Roderic Guigó, and Paolo Ribeca. 2012. The GEM mapper: fast, accurate and versatile alignment by filtration. *Nature Methods* 9, 12 (2012), 1185–1188.

[75] Yasuaki Mitani, Fumihiko Ino, and Kenichi Hagihara. 2016. Parallelizing exact and approximate string matching via inclusive scan on a GPU. *IEEE Transactions on Parallel and Distributed Systems* 28, 7 (2016), 1989–2002.

[76] Mahmood Moghimi and Ali Yazdian Varjani. 2016. New rule-based phishing detection method. *Expert Systems With Applications* 53 (2016), 231–242.

[77] Gene Myers. 1999. A fast bit-vector algorithm for approximate string matching based on dynamic programming. *Journal of the ACM (JACM)* 46, 3 (1999), 395–415.

[78] Gene Myers. 2014. Efficient local alignment discovery amongst noisy long reads. In *International Workshop on Algorithms in Bioinformatics*. Springer, 52–67.

[79] Anirban Nag, CN Ramachandra, Rajeev Balasubramonian, Ryan Stutsman, Edouard Giacomin, Hari Kambalasubramanyam, and Pierre-Emmanuel Gaillardon. 2019. Gencache: Leveraging in-cache operators for efficient sequence alignment. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 334–346.

[80] Gonzalo Navarro. 2001. A guided tour to approximate string matching. *ACM Computing Surveys (CSUR)* 33, 1 (2001), 31–88.

[81] Gonzalo Navarro and Mathieu Raffinot. 2002. *Flexible pattern matching in strings: practical on-line search algorithms for texts and biological sequences*. Cambridge University Press.

[82] Saul B Needleman and Christian D Wunsch. 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology* 48, 3 (1970), 443–453.

[83] Nuno Neves, Nuno Sebastião, David Matos, Pedro Tomás, Paulo Flores, and Nuno Roma. 2014. Multicore SIMD ASIP for next-generation sequencing and alignment biochip platforms. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 23, 7 (2014), 1287–1300.

[84] Lucas SN Nunes, Jacir L Bordim, Koji Nakano, and Yasuaki Ito. 2015. A fast approximate string matching algorithm on GPU. In *2015 Third international symposium on computing and networking (CANDAR)*. IEEE, 188–192.

[85] Joshua Quick, Nicholas J Loman, Sophie Duraffour, Jared T Simpson, Ettore Severi, Lauren Cowley, Joseph Akoi Bore, Raymond Koundouno, Gytis Dudas, Amy Mikhail, et al. 2016. Real-time, portable genome sequencing for Ebola surveillance. *Nature* 530, 7589 (2016), 228–232.

[86] Cyrus Rashtchian, Konstantin Makarychev, Miklos Racz, Siena Ang, Djordje Jevdjic, Sergey Yekhanin, Luis Ceze, and Karin Strauss. 2017. Clustering billions of reads for DNA data storage. *Advances in Neural Information Processing Systems* 30 (2017).

[87] Mikko Rautiainen and Tobias Marschall. 2020. GraphAligner: rapid and versatile sequence-to-graph alignment. *Genome Biology* 21, 1 (2020), 1–28.

[88] Jason A Reuter, Damek V Spacek, and Michael P Snyder. 2015. High-throughput sequencing technologies. *Molecular Cell* 58, 4 (2015), 586–597.

[89] Torbjørn Rognes and Erling Seeberg. 2000. Six-fold speed-up of Smith–Waterman sequence database searches using parallel processing on common microprocessors. *Bioinformatics* 16, 8 (2000), 699–706.

[90] David Sankoff. 1972. Matching sequences under deletion/insertion constraints. *Proceedings of the National Academy of Sciences* 69, 1 (1972), 4–6.

[91] Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, and Stefan Mangard. 2017. Malware guard extension: Using SGX to conceal cache attacks. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 3–24.

[92] Peter H Sellers. 1974. On the theory and computation of evolutionary distances. *SIAM J. Appl. Math.* 26, 4 (1974), 787–793.

[93] Damla Senol Cali, Jeremie S Kim, Saugata Ghose, Can Alkan, and Onur Mutlu. 2019. Nanopore sequencing technology and tools for genome assembly: computational analysis of the current state, bottlenecks and future directions. *Briefings in Bioinformatics* 20, 4 (2019), 1542–1559.

[94] Jay Shendure, Shankar Balasubramanian, George M Church, Walter Gilbert, Jane Rogers, Jeffery A Schloss, and Robert H Waterston. 2017. DNA sequencing at 40: past, present and future. *Nature* 550, 7676 (2017), 345–353.

[95] Barton E Slatko, Andrew F Gardner, and Frederick M Ausubel. 2018. Overview of next-generation sequencing technologies. *Current Protocols in Molecular Biology* 122, 1 (2018), e59.

[96] Temple F Smith and Michael S Waterman. 1981. Identification of common molecular subsequences. *Journal of Molecular Biology* 147, 1 (1981), 195–197.

[97] Matthew Snover, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul. 2006. A study of translation edit rate with targeted human annotation. In *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas: Technical Papers*. 223–231.

[98] Víctor Soria-Pardos, Max Doblas, Guillem López-Paradís, Gerard Candón, Narcís Rodas, Xavier Carril, Pau Fontova-Musté, Neiel Leyva, Santiago Marco-Sola, and Miquel Moretó. 2022. Sargantana: A 1 GHz+ in-order RISC-V processor with SIMD vector extensions in 22nm FD-SOI. In *2022 25th Euromicro Conference on Digital System Design (DSD)*. IEEE, 254–261.

[99] Peter Stanchev, Weiyue Wang, and Hermann Ney. 2019. EED: Extended edit distance measure for machine translation. In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*. 514–520.

[100] Zachary D Stephens, Skylar Y Lee, Faraz Faghri, Roy H Campbell, Chengxiang Zhai, Miles J Efron, Ravishankar Iyer, Michael C Schatz, Saurabh Sinha, and Gene E Robinson. 2015. Big data: astronomical or genomical? *PLoS Biology* 13, 7 (2015), e1002195.

[101] Kendall Stewart, Yuan-Jyue Chen, David Ward, Xiaomeng Liu, Georg Seelig, Karin Strauss, and Luis Ceze. 2018. A content-addressable DNA database with learned sequence encodings. In *International Conference on DNA Computing and Molecular Programming*. Springer, 55–70.

[102] Christopher N Takahashi, Bichlien H Nguyen, Karin Strauss, and Luis Ceze. 2019. Demonstration of end-to-end automation of DNA data storage. *Scientific reports* 9, 1 (2019), 1–5.

[103] Julie D Thompson, Desmond G Higgins, and Toby J Gibson. 1994. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research* 22, 22 (1994), 4673–4680.

[104] Yatish Turakhia, Gill Bejerano, and William J Dally. 2018. Darwin: A genomics co-processor provides up to 15,000 x acceleration on long read assembly. *ACM SIGPLAN Notices* 53, 2 (2018), 199–213.

[105] Yatish Turakhia, Sneha D. Goenka, Gill Bejerano, and WIlliam J. Dally. 2019. Darwin-WGA: A co-processor provides increased sensitivity in whole genome alignments with High Speedup. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 359–372.

[106] Md. Vasimuddin, Sanchit Misra, Heng Li, and Srinivas Aluru. 2019. Efficient architecture-aware acceleration of BWA-MEM for multicore systems. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 314–324.

[107] Taras K Vintsyuk. 1968. Speech discrimination by dynamic programming. *Cybernetics* 4, 1 (1968), 52–57.

[108] Martin Šošić and Mile Šć. 2017. Edlib: a C/C++ library for fast, exact sequence alignment using edit distance. *Bioinformatics* 33, 9 (01 2017), 1394–1395. arXiv:https://academic.oup.com/bioinformatics/article-pdf/33/9/1394/25151249/btw753.pdf

[109] Robert A Wagner and Michael J Fischer. 1974. The string-to-string correction problem. *Journal of the ACM (JACM)* 21, 1 (1974), 168–173.

[110] Jing Wang, Nicole E Moore, Yi-Mo Deng, David A Eccles, and Richard J Hall. 2015. MinION nanopore sequencing of an influenza genome. *Frontiers in Microbiology* 6 (2015), 766.

[111] Ting Wang, Lucinda Antonacci-Fulton, Kerstin Howe, Heather A Lawson, Julian K Lucas, Adam M Phillippy, Alice B Popejoy, Mobin Asri, Caryn Carson, Mark JP Chaisson, et al. 2022. The Human Pangenome Project: a global resource to map genomic diversity. *Nature* 604, 7906 (2022), 437–446.

[112] Xiao Shaun Wang, Yan Huang, Yongan Zhao, Haixu Tang, XiaoFeng Wang, and Diyue Bu. 2015. Efficient genome-wide, privacy-preserving similar patient query based on private edit distance. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 492–503.

[113] Michael S Waterman, Temple F Smith, and William A Beyer. 1976. Some biological sequence metrics. *Advances in Mathematics* 20, 3 (1976), 367–387.

[114] Mrinalini Watsa, Gideon A Erkenswick, Aaron Pomerantz, and Stefan Prost. 2020. Portable sequencing as a teaching tool in conservation and biodiversity research. *PLoS biology* 18, 4 (2020), e3000667.

[115] Chi Wai Yu, KH Kwong, Kin-Hong Lee, and Philip Heng Wai Leong. 2003. A Smith-Waterman systolic cell. In *International Conference on Field Programmable Logic and Applications*. Springer, 375–384.

[116] Jikai Zhang, Haidong Lan, Yuandong Chan, Yuan Shang, Bertil Schmidt, and Weiguo Liu. 2019. BGSA: a bit-parallel global sequence alignment toolkit for multi-core and many-core architectures. *Bioinformatics* 35, 13 (2019), 2306–2308.

[117] Zhuowen Zou, Hanning Chen, Prathyush Poduval, Yeseong Kim, Mahdi Imani, Elaheh Sadredini, Rosario Cammarota, and Mohsen Imani. 2022. BioHD: An efficient genome sequence search platform using hyperDimensional Memorization. In *Proceedings of the 49th Annual International Symposium on Computer Architecture* (New York, New York) *(ISCA '22)*. Association for Computing Machinery, New York, NY, USA, 656–669.