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.moret@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:

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 $25-183 \times$ aligning short sequences and $48-112 \times$ 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 0.0216mm². (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 0.0216mm² (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.
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., evo-lutive 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 \( p = p_0p_1 \ldots p_{n-1} \) and the text \( t = t_0t_1 \ldots t_{m-1} \) 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-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 \( O(\frac{n^2}{w})km \) 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 x 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.

$$\Delta v_{i,j} = \min\{-eq_{i,j}, \Delta v_{i,j-1}, \Delta h_{i-1,j}\} + 1 - \Delta h_{i-1,j}$$

$$\Delta h_{i,j} = \min\{-eq_{i,j}, \Delta v_{i,j-1}, \Delta h_{i-1,j}\} + 1 - \Delta v_{i,j-1}$$ (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 Genomic 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, SeGrA,M, 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 (GIB) 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).

### 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 $O(\frac{2}{k}km)$ 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 10kb-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
As a result, the accuracy of these heuristics is unpredictable and of the tile). GMX extensions can be easily exploited to accelerate footprint (as GMX only needs to store the elements at the edges elements quadratically in the tile size, and (2) decreasing the memory tiles from the DP-matrix, (1) reducing the computational require-
ment by element as shown in Figure 4.a.1. This work presents the Classical DP-based alignment algorithms, like Smith-Waterman (SW) and Needleman-Wunsch (NW), compute the DP-matrix ele-
moments quadratically in the tile size and the memory requirements are signiﬁcantly reduced. In the tile elements are computed on-the-
y and never stored). 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 outper-
form their costs. In particular, bit-parallel accelerators implement-
ing 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 acceler-
ators 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 ﬂex-
ible 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 require-
ments 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-Waterman (SW) and Needleman-Wunsch (NW), compute the DP-matrix ele-
ment 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 comput-
ing 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 algo-

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 \( \frac{mn}{T^2} \) 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

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.
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 \( \frac{m_B}{2} \) tiles, storing \( \frac{m_B}{2} \) 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 backtraces, 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 \( (\Delta_{in}, \Delta_{out}, eq) = (\Delta_{i,j-1}, \Delta_{j,i-1}, eq_{i,j}) \) the inputs and \( (\Delta_{out}, \Delta_{out}) = (\Delta_{i,j}, \Delta_{j,i}) \) the outputs of computing a single DP-element using BPM (Eq. 1). We identify a symmetry between the \( \Delta \) and \( \Delta_h \) computation. Thus, we can condense both computations into a single equation (Eq. 2). This way, \( \Delta_{out} = GMX_{A}(\Delta_{in}, \Delta_{in}, eq) \) and \( \Delta_{out} = GMX_{A}(\Delta_{in}, \Delta_{in}, eq) \).

\[
GMX_{A}(\Delta_0, \Delta_1, eq) = min(-eq, \Delta_0, \Delta_1) + 1 - \Delta_1
\]  

(2)

Then, we encode each \( \Delta \) 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.

\[
GMX_{A}(\Delta_0, \Delta_1, eq)[0] = (!\Delta[0][1] \& \& eq \& \& \neg \Delta[1][1])
\]

(3)

GMX-Tile algorithm allows fast and efficient computation of \((T \times T)\) tiles, using 12 bit-operations per each \( \Delta \) 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 \((S \times T^2)\) 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 \(4T^2\) 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: \texttt{gmx.v} and \texttt{gmx.h} to compute the \((T \times T)\) elements of a tile (Fig. 5.a) and \texttt{gmx.tb} to compute the traceback (Fig. 5.b). Let \( \Delta_{ij} = \langle \Delta_{ij}, \ldots \Delta_{ij,T-1} \rangle \) and \( \Delta_{ij} = \langle \Delta_{ij}, \ldots \Delta_{ij,T-1} \rangle \) be vectors of horizontal/vertical \( \Delta \) values (i.e., differential-encoded DP-elements). GMX instructions can use standard \( R \)-type RISC-V encoding, using the reserved custom op-codes.

- \texttt{gmx.v rd, rs1, rs2}. Given \( \Delta_{ij} \) and \( \Delta_{ij} \), stored in the general purpose registers \( rs1 \) and \( rs2 \), this instruction computes the tile alignment between the \texttt{gmx_text} and \texttt{gmx_pattern}, outputting \( \Delta_{ij} \) in the register \( rd \).
- \texttt{gmx.h rd, rs1, rs2}. Similar to \texttt{gmx.v}, this instruction computes \( \Delta_{ij} \) and stores it in \( rd \).

![Figure 5: (a) \texttt{gmx.v} and \texttt{gmx.h} instructions to compute the \( \Delta_{ij} \) and \( \Delta_{ij} \) of a tile. (b) \texttt{gmx.tb} instruction to compute a tile’s traceback.](image-url)
GMX: Instruction Set Extensions for Fast, Scalable, and Efficient Genome Sequence Alignment

Due to this limitation, it is not possible to merge target CPU allowed for two destination register ports, it would be cant bits of gmx_hi store gmx_pattern gmx_lo gmx_pos gmx_pattern gmx.{v,h} output gmx_text alignment (“MIMMD”), as described in Algorithm 2.

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, rs2=ΔH, 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 × m/T) matrix containing each tile’s ΔV and ΔH vectors. The algorithm proceeds column-wise, computing each T × 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 (ΔHin = M[i−1, j], h) and left tile (ΔVin = M[i, j−1], o). Using gmx.v and gmx.h instructions, the algorithm generates ΔVout and ΔHout. 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
1 n = length of the pattern;
2 m = length of the text;
3 for j=1 to m/T do # Loop pattern in T-chunks
4 csrw gmx_text, text[j × T:T];
5 for i=1 to n/T do # Loop text in T-chunks
6 csrw gmx_pattern, pattern[i × T:T];
7 (ΔV_in, ΔH_in) = (M[i][j-1].a, M[i-1][j].h);
8 gmx.v ΔV_out, ΔV_in, ΔH_in;
9 gmx.h ΔH_out, ΔV_in, ΔH_in;
10 M[i][j].a = ΔV_out;
11 M[i][j].h = ΔH_out;
end
end

Algorithm 2: Traceback computation using GMX.

Input: pattern, text, M, T
Output: ali2vec, ali2ptr
1 h = m/T - 1;
2 v = n/T - 1;
3 csrw gmx_pos, bottom-right_cell;
4 csrw gmx_text, text[h];
5 csrw gmx_pattern, pattern[v];
6 while v ≥ 0 & & h ≥ 0 do
7 (ΔV_in, ΔH_in) = (M[i][j-1].a, M[i-1][j].h);
8 gmx.tb ΔV_in, ΔH_in;
9 csrr ali2vec[ali2ptr++], gmx_hi;
10 csrr ali2vec[ali2ptr++], gmx_lo;
11 csrr nextTile, gmx_hi;
12 if nextTile is ⇠ then
13 csrw gmx_text, text[--h];
14 csrw gmx_pattern, pattern[--v];
15 else if nextTile is ↑ then
16 csrw gmx_pattern, pattern[--v];
17 else nextTile is ←
18 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.

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 × T) basic Compute Cores (CC_AC) (Figure 7 left). Each CC_AC of GMX-AC computes a single DP element (Δout and Δhout), using left Δvin, upper Δhin, 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_A modules (Eq. 3). Due to the simplicity of the GMX_A 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=\(\text{\textbackslash}\), insertion=\(\text{\textbackslash}\), and deletion=\(\text{\textbackslash}\)). 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
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 \( (\overline{1} - \overline{2}) \). 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. \((\overline{3} - \overline{6})\). 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 \((T=32)\)-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.
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.
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 (<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 0.0216 mm² where 0.008 mm² corresponds to the GMX-AC module and 0.0108 mm² 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

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

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

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

**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 1.3–1.9x better than GenASM and 7.2–16.2x better than Darwin. Yet, GMX introduces a minimal silicon
area overhead to the existing SoC design (15.46× less 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 1.58x 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 SeGaRm. 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 SeGaRm (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.

<table>
<thead>
<tr>
<th>Study</th>
<th>Device</th>
<th>PE</th>
<th>Area/PE</th>
<th>PGCUPS/PE</th>
</tr>
</thead>
<tbody>
<tr>
<td>GMX Unit</td>
<td>ASIC</td>
<td>1 PE</td>
<td>0.02mm²</td>
<td>1024.0</td>
</tr>
<tr>
<td>Core+GMX</td>
<td>ASIC</td>
<td>1 PE</td>
<td>1.24mm²</td>
<td>1024.0</td>
</tr>
<tr>
<td>GenASM [17]</td>
<td>ASIC</td>
<td>32 PE</td>
<td>0.33mm²</td>
<td>64.0</td>
</tr>
<tr>
<td>ABSW [66]</td>
<td>ASIC</td>
<td>1 PE</td>
<td>5.11mm²</td>
<td>61.4</td>
</tr>
<tr>
<td>GenAX [37]</td>
<td>ASIC</td>
<td>4 PE</td>
<td>1.34mm²</td>
<td>112.0</td>
</tr>
<tr>
<td>Darwin [104]</td>
<td>ASIC</td>
<td>64 PE</td>
<td>1.34mm²</td>
<td>154.2</td>
</tr>
<tr>
<td>ASAP [12]</td>
<td>FPGA</td>
<td>1 PE</td>
<td>277K LUTs</td>
<td>51.2</td>
</tr>
<tr>
<td>FPGASW [34]</td>
<td>FPGA</td>
<td>1 PE</td>
<td>58K LUTs</td>
<td>105.9</td>
</tr>
<tr>
<td>DPX</td>
<td>GPU</td>
<td>132 SM</td>
<td>–</td>
<td>42.4</td>
</tr>
<tr>
<td>BPM-GPU [20]</td>
<td>GPU</td>
<td>8 SM</td>
<td>–</td>
<td>287.5</td>
</tr>
<tr>
<td>NVBio</td>
<td>GPU</td>
<td>15 SM</td>
<td>–</td>
<td>66.6</td>
</tr>
</tbody>
</table>

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, GMX 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 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 PPOS 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 2020 research and innovation programme and the respective national research organisations from Spain (PCI2021-121991/MCIN/AEI/10.13039/5011001011033), 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/10.13039/50110011033 and FSE+).

REFERENCES


wavefront algorithm for genomics pairwise alignment. In 2015 IEEE International Parallel and Distributed Processing Symposium. 561–570.


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


