SMX: Heterogeneous Architecture for Universal Sequence Alignment Acceleration

Max Doblas
Barcelona Supercomputing Center (BSC)
Barcelona, Barcelona, Spain
Universitat Politècnica de Catalunya (UPC)
Barcelona, Barcelona, Spain
max.doblas@bsc.es

Po Jui Shih
Cornell University
Ithaca, NY, USA
ps2229@cornell.edu

Oscar Lostes-Cazorla
Universitat Politècnica de Catalunya (UPC)
Barcelona, Barcelona, Spain
Barcelona Supercomputing Center (BSC)
Barcelona, Barcelona, Spain
oscar.lostes@bsc.es

Miquel Moreto
Universitat Politècnica de Catalunya (UPC)
Barcelona, Barcelona, Spain
Barcelona Supercomputing Center (BSC)
Barcelona, Barcelona, Spain
miquel.moreto@bsc.es

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

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

Abstract

Sequence alignment is a fundamental building block for critical applications across multiple fields, such as computational biology and information retrieval. The rapid advancement of genome sequencing technologies and breakthrough generative AI tools, like AlphaFold, has driven an exponential increase in sequence-data production, creating a pressing need for fast and efficient sequence alignment tools to analyze ever-growing biological sequence databases. Notwithstanding the numerous accelerators proposed, from general-purpose architectures (CPUs and GPUs) to domain-specific designs (FPGAs and ASICs), the most efficient solutions suffer from over-specialization and fail to adapt to the wide variety of irregular use cases demanded by practical sequence alignment applications. Thus, it remains a challenge to design an architecture that can balance efficiency and flexibility to meet the demands of real-world alignment applications. This work introduces SMX, a heterogeneous architecture designed for high-performance sequence alignment that supports various configurations for different sequence types (DNA, protein, and ASCII text) and alignment models (including weighted gaps and substitution matrices). SMX integrates an ISA extension (SMX-1D) for irregular and sequential tasks and a specialized coprocessor (SMX-2D) to accelerate regular and parallel tasks, both orchestrated by the general-purpose core to enable seamless integration with state-of-the-art sequence alignment algorithms. Our results demonstrate that SMX's heterogeneous architecture accelerates different sequence alignment use cases by 256–744× compared to state-of-the-art software implementations when aligning real datasets. Compared to specialized hardware accelerators, SMX delivers up to 18.5× more peak performance per area added while providing greater flexibility to accelerate different use cases. Physical design results targeting a 22nm technology node estimate SMX's area at 0.34mm^2, which is only 30% of a single-issue in-order CPU. SMX offers a high-performance and efficient heterogeneous architecture for accelerating practical sequence alignment algorithms, providing a scalable and flexible solution tailored to meet the needs of modern sequence-analysis tools. Furthermore, an SMX case study explores the frontier between flexibility and efficiency in domain-specific architectures and accelerators.

CCS Concepts

• **Applied computing** → **Bioinformatics**; • **Computer systems organization** → **Heterogeneous (hybrid) systems**.

Keywords

Sequence alignment, ISA extensions, hardware acceleration, genomics, heterogeneous, bioinformatics, microarchitecture

MICRO '25, Seoul, Republic of Korea

ACM ISBN 979-8-4007-1573-0/25/10
https://doi.org/10.1145/3725843.3756018

ACM Reference Format:
Max Doblas, Po Jui Shih, Oscar Lostes-Cazorla, Miquel Moreto, Christopher Batten, and Santiago Marco-Sola. 2025. SMX: Heterogeneous Architecture for Universal Sequence Alignment Acceleration. In *58th IEEE/ACM International Symposium on Microarchitecture (MICRO '25), October 18–22, 2025, Seoul, Republic of Korea.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3725843.3756018

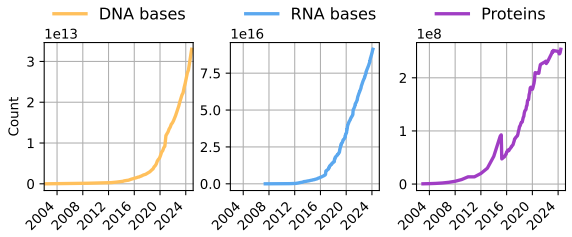


Figure 1: Growth of DNA (GenBank [78]), RNA (SRA [79]), and protein (Uniprot [25]) databases in the last two decades.

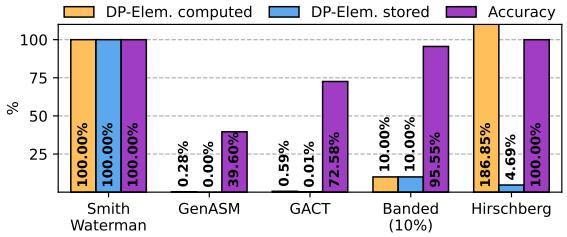


Figure 2: Percentage of the total DP-elements computed and stored, and alignment recall (percentage of correctly aligned sequences over the whole dataset) for different alignment algorithms on Oxford Nanopore DNA sequences [11].

1 Introduction

Sequence alignment assesses the similarity between two sequences and is a fundamental building block across many application domains, including information retrieval [9, 23, 80, 81], computational biology [7, 32, 57, 86, 100], natural language processing [39, 96], and others [26, 59, 90]. In computational biology, recent advances in sequencing technologies have made sequence alignment a critical component of genome sequence analysis [4, 60, 91, 93]. Moreover, breakthroughs in generative-AI-based methods, such as AlphaFold [54] for protein structure prediction, have revolutionized our understanding of biology and contributed to an exponential growth of biological sequence databases. Figure 1 shows the exponential growth of sequence databases over the past decades.

This increase in genomic data has been crucial for the development of population-wide pangenome studies [38, 64, 107], personalized healthcare [8, 22, 35, 41, 42, 52, 75], and effective COVID-19 outbreak tracing, to name a few. However, the rapid growth in sequence data production and biological databases places a significant computational burden on data-analysis tools. As a result, accelerating key computational tasks (e.g., sequence alignment) has become key for scaling data analyses to large volumes of data.

Most sequence alignment algorithms are based on dynamic programming (DP) [82, 94] and involve computing an $m \times n$ matrix (DP-matrix) of integer elements (DP-elements), being m and n the sequences' length. Hence, DP-based sequence alignment algorithms have quadratic complexity in time and memory, posing scalability challenges for comparing long sequences (e.g., needing 4TB of DP-matrix for aligning a 1Mbp long ONT sequence [53]). To address this challenge, a wide array of diverse algorithms use heuristics that narrow computation to specific regions of the DP-matrix (e.g., banded [103]) or stop calculations if the alignment score drops below a predefined threshold (e.g., Xdrop [7, 63, 98]). Some algorithms trade memory for additional computation, recomputing regions of the DP-matrix recursively instead of storing them (e.g., Hirschberg algorithm [103]). Modern sequence alignment tools [31, 44, 95] combine variations of these algorithms and heuristics to balance efficiency and practicality in real-world applications. Figure 2 shows the trade-offs between performance (DP-elements computed), memory requirements (DP-elements stored), and accuracy for different alignment algorithms. Ultimately, all these tools require fast computation of part of, if not the entire, DP-matrix, highlighting the need for new architectural solutions to accelerate DP computations while remaining adaptable to diverse algorithms and use cases.

The demand for faster sequence alignment tools has driven research into hardware accelerators, including solutions based on GPUs [1, 2, 74, 84], FPGAs [14, 21, 46, 48, 69, 92, 109], PIM [20, 28, 47, 56, 70, 77], and ASICs [5, 15, 16, 36, 67, 83, 101]. Recently, new ISA extensions have been proposed to accelerate sequence alignment [30, 85], which leverages existing CPU/GPU hardware to reduce both area and power costs. These ISA extensions provide greater flexibility compared to monolithic, standalone domain-specific accelerators (DSA), enabling the efficient acceleration of various sequence alignment algorithms. However, ISA extensions are limited by the core performance and often struggle to fully utilize their custom functional units while meeting strict constraints on area, latency, and data width. In contrast, highly specialized standalone DSAs, such as Darwin [101] and GenASM [15], can exploit high parallelism and have shown their ability to achieve significant performance and efficiency improvements compared to general-purpose accelerators. However, DSAs often struggle to accelerate irregular and sequential operations, are limited in the input sizes they can handle, and typically require a non-negligible area. Moreover, optimized DSA designs are usually tailored for specific use cases, implementing fixed algorithms and heuristics and even limiting the sequence alphabet (e.g., 2-bit for aligning DNA sequence only). As a result, standalone DSAs are typically inflexible and unable to adapt to different applications in rapidly evolving fields like modern genomics and sequence biology.

Our goal is to design a high-performance and flexible architecture for accelerating different sequence alignment algorithms across multiple application domains.

In this work, we propose SMX, an efficient heterogeneous architecture that enables fast and scalable sequence alignment acceleration of different algorithms (e.g., banded, Xdrop, Hirschberg) and applications (e.g., DNA, protein, ASCII-text). SMX speeds up DP-based algorithms by integrating (1) SMX-1D, a flexible ISA-extended CPU for irregular and latency-sensitive DP operations, and (2) SMX-2D, a high-performance coprocessor that accelerates regular and compute-intensive 2D DP-matrix operations, all coordinated by the general-purpose core. SMX introduces an improved encoding scheme that compresses DP-elements to 2, 4, 6, or 8 bits (configurable), reducing memory footprint and cache pressure. Our narrow-width encoding enables packing multiple DP-elements, increasing the computational parallelism and reducing the area of the design by around 90% compared to other DSAs. At the core of SMX, we introduce the SMX-engine, an optimized design to accelerate the computation of complete tiles of the DP-matrix for different alignment configurations. Equipped with several SMX-workers

to process independent sequence alignment tasks in parallel, our SMX-engine achieves a near 100% utilization, computing a peak of 1024 DP-elements per cycle. As opposed to inflexible, monolithic, and standalone DSAs, SMX presents an architectural design that balances efficiency and flexibility for numerous use cases.

Key Results. We evaluate SMX-accelerated implementations of real-world algorithms in different practical scenarios, including DNA, protein, and ASCII-text alignment. We demonstrate that (1) SMX's heterogeneous architecture allows accelerating state-of-the-art software implementations by 256-428× and 744× aligning real DNA and protein datasets, respectively. (2) SMX heterogeneous system achieves a peak throughput per area added of 15.5-18.6× higher that of state-of-the-art standalone DSAs while being more flexible and requiring a minimal area overhead of 0.34 mm^2; (3) SMX-engine allows a 4-64× memory footprint reduction while reducing the bandwidth to memory and cache pressure, enabling SMX to scale in multi-accelerator SoC. In summary, this paper makes the following contributions:

- We propose the SMX-1D ISA extension, a flexible instruction set optimized to accelerate sequence alignment. SMX-1D ISA uses an improved differential-encoding scheme to compress multiple variable-size DP-elements, minimizing memory footprint and enabling higher parallelism.
- We propose the SMX-2D, a high-performance coprocessor designed to compute complete tiles of the DP-matrix in parallel. Using SMX differential-encoding, SMX-2D implements a 2D computing engine that processes multiple tiles in parallel, relieving the core from compute-intensive and regular tasks.
- We present SMX, to our knowledge, the first heterogeneous architecture for accelerating different sequence alignment algorithms and applications. We propose a co-design that integrates SMX-1D ISA and SMX-2D coprocessor to (a) accelerate regular DP-matrix computations, (b) handle irregular alignment tasks, and (c) adapt to different algorithms and applications.
- We conduct a thorough performance analysis using a cycle-level simulator to evaluate SMX's performance to accelerate widely-used sequence alignment algorithms across different use cases. Moreover, we evaluate SMX scalability in a multicore system, demonstrating near-linear scalability for multiple alignment workloads.
- We integrated SMX-1D and SMX-2D into a RISC-V in-order core at the RTL level and performed a physical design implementation using a 22nm technology node. The results indicate that SMX-1D and SMX-2D occupy 0.015 mm^2 and 0.328 mm^2, respectively, corresponding to only 1.4% and 29.7% of the total design area.

2 Background

2.1 Sequence Alignment

Given two input sequences, reference $R = r_0 r_1 \dots r_{m-1}$ and query $Q = q_0 q_1 \dots q_{n-1}$ of length m and n, and a scoring function, the optimal alignment is the sequence of operations (i.e., match, mismatch, insertion, and deletion) that transforms one sequence into the other, maximizing the score. Sequence alignment is usually computed using some variation of DP and requires performing two steps: (1) DP-matrix computation and (2) alignment traceback.

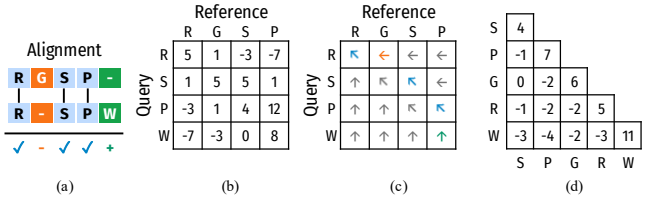


Figure 3: Simple example of a DP-based sequence alignment of two proteins. (a) Final alignment. (b) DP-matrix. (c) Traceback matrix. (d) Partial BLOSUM-62 substitution matrix. The insertion and deletion penalties are both set to $I = D = -4$.

Sequence alignment algorithms first compute an $m \times n$ DP-matrix of integer DP-elements (Figure 3.b). Using the gap scoring function, the matrix is computed with the DP-recurrence relations in Eq 2, where I and D represent insertion/deletion costs, and $S(q_i, r_j)$ (Figure 3.d) denotes the substitution cost of character q_j with r_i. After computing the DP-matrix, the bottom-right DP-element $M_{m,n}$ contains the optimal alignment score.

In the traceback step (Figure 3.c), the optimal alignment is derived by tracing back the DP-elements that originated the optimal alignment score, from $M_{m,n}$ to $M_{0,0}$, following the path of maximum score DP-elements. As a result, the traceback produces the alignment between the two sequences (Figure 3.a).

$$M_{i,0} = i \cdot I \qquad M_{0,j} = j \cdot D \tag{1}$$

$$M_{i,j} = max\{M_{i-1,j-1} + S(q_{i-1}, r_{j-1}), M_{i-1,j} + I, M_{i,j-1} + D\} \tag{2}$$

Computing the DP-matrix has quadratic $O(m \times n)$ complexity in time and space. Parallelism can be exploited within each antidiagonal, as all DP-elements on a single antidiagonal are independent and can be computed simultaneously. In contrast, the traceback step is inherently sequential, with $O(m + n)$ complexity, as each step depends on the previous one to reconstruct the alignment path from $M_{m,n}$ to $M_{0,0}$. This process involves branch-heavy execution, frequent mispredictions, and loop-carried dependencies, limiting parallelism. Certain applications, like protein alignment and sequence pre-filters, only require the alignment score (i.e., $M_{m,n}$), avoiding the need to store the full DP-matrix.

2.2 Sequence Alignment Models

Depending on the application, parameters from Eq 2 are adjusted to model each sequence alignment problem adequately.

For example, the *edit model* is commonly used for applications such as NLP [51], spell-checking [73], comparing ASCII code/text strings, and analyzing genome DNA/RNA sequences. The edit model assigns a unit cost to insertions and deletions (i.e., $I = D = 1$) and defines the substitution cost as $S(q_i, r_j) = 0$ if $q_i == r_j$, and $S(q_i, r_j) = 1$ otherwise.

Other weighted *gap models* employ different values of I, D, and $S(q_i, r_j)$ (where $S(q_i, r_j) = M$ if $q_i == r_j$, and $S(q_i, r_j) = X$ otherwise) to address complex alignment problems, such as genomic evolutionary events, genetic mutations, and other DNA/RNA variations. More complex models, like *protein models*, assign a different penalty to each character's substitutions based on a predefined substitution-matrix. Broadly used protein-alignment tools, like

BLAST [6], BLAT [58], and DIAMOND [13], use well-established substitution-matrices (e.g., BLOSUM [49] and PAM [27]) to estimate the likelihood of mutations between amino acids in protein alignments, helping to compute biologically significant alignments.

2.3 Practical Sequence Alignment Algorithms

Practical sequence alignment algorithms rely on sophisticated heuristics and drop strategies to reduce the number of DP computations to certain regions of the DP-matrix. The most notable example is the **banded heuristic** [19, 31, 43, 63, 95, 98, 102], which calculates a narrow band around the main diagonal of the DP-matrix. Based on the observation that optimal alignments often lie near the main diagonal (especially for similar sequences), the banded heuristic achieves significant speed-up while maintaining the accuracy. In the same spirit, **drop strategies** [7, 63, 110] terminate the alignment computation when the score of the computed DP-elements drops below a threshold, indicating that the regions being aligned have low similarity and little relevance to the application.

Notwithstanding, alignment algorithms still demand significant memory. **Hirschberg's algorithm** [31, 50, 71] is a well-known divide-and-conquer strategy to reduce memory usage at the cost of recomputing parts of the DP-matrix. In a nutshell, Hirschberg works by recursively breaking the alignment into smaller subproblems, solving each one independently, and combining the results to reconstruct the full alignment. Hirschberg requires linear memory to compute each subproblem breakpoint at the cost of partially recomputing the DP-matrix at each step.

While alignment algorithms, heuristics, and drop techniques vary widely, their core operation can be reduced to computing specific regions of the DP-matrix, called DP-blocks. Despite being regular and parallelizable, computing DP-blocks is usually the most computational-demanding task in practical alignment algorithms. Depending on each algorithm's idiosyncrasies (e.g., steps, heuristics, drops), algorithms require computing different workloads of DP-blocks. As a result, orchestrating the computation of different DP-blocks while implementing sophisticated heuristics and drops becomes a highly irregular task involving data dependencies and heavy-control flow operations.

2.4 Differential Encoding

Efficient CPU-based sequence alignment leverages SIMD instructions to accelerate DP-matrix computation. However, these implementations typically use 16-bit or 32-bit integers to store DP-elements, which limit SIMD parallelism since fewer values can fit in a register. Longer input sequences further increase DP-element values, requiring larger integers that reduce parallelism and increase memory usage. To mitigate these limitations, studies [76, 99] propose differential encoding, which stores differences between consecutive DP-matrix values instead of absolute values. Consequently, we define $\Delta v_{i,j} = M_{i,j} - M_{i-1,j}$ and $\Delta h_{i,j} = M_{i,j} - M_{i,j-1}$ and reformulate the DP-recurrence equations to compute the differential values, as shown in Eq. 3 and 4.

$$\Delta v_{i,j} = max\{S(q_{i-1}, r_{j-1}) - \Delta h_{i-1,j}, \Delta v_{i,j-1} - \Delta h_{i-1,j} + I, D\} \quad (3)$$

$$\Delta h_{i,j} = max\{S(q_{i-1}, r_{j-1}) - \Delta v_{i,j-1}, I, \Delta h_{i-1,j} - \Delta v_{i,j-1} + D\} \quad (4)$$

As shown in [63, 99], difference values are typically small and can often be stored in 8 bits or less. This makes differential encoding a crucial optimization in alignment tools like Minimap2 [63], enhancing SIMD performance while reducing memory usage.

3 Motivation and Goal

Contrary to common belief, sequence alignment is not a single problem but a family of problems defined across different alignment models, heuristics, alphabets, and more. Some examples are: ① NeoDisc [52], a proteogenomic pipeline for neoantigen discovery, uses BWA [62] (DNA-gap + Xdrop) for DNA read alignment , STAR [29] (RNA-gap/DNA-gap + Banded) for RNA-seq analysis, and BLAST [17] (protein + BLOSUM + Xdrop) for protein alignment; ② MEDUSA, a pipeline [75] for metagenomic analysis, uses Bowtie2 [61] (DNA-gap + Full) and DIAMOND [12] (protein + BLOSUM + Banded); ③ Racoon [105], an assembly pipeline uses Edlib [95] (DNA-edit + Banded + Hirschberg) for read filtering and SPOA [105] (DNA-gap + Full) for partial order alignment.

Although multiple software and hardware sequence alignment accelerators have been proposed over the years, each has its limitations. Unfortunately, none of the existing solutions covers all requirements or performance across the diverse range of sequence alignment scenarios. This section discusses the benefits and limitations of current architectures used for sequence alignment.

CPU architectures offer great flexibility for executing different sequence alignment algorithms. Modern sequence alignment tools implement complex algorithms and sophisticated heuristics optimized for general-purpose processors, often leveraging SIMD instructions [31, 34, 63, 65, 68, 88, 99, 108]. However, CPU-based solutions present performance limitations due to the lack of specific support to accelerate DP computations and the overhead of extra CPU instructions, such as memory access and control flow. In addition, CPUs take up significantly more silicon area per core (compared to GPUs or DSAs), resulting in lower performance per unit area. Furthermore, as core counts increase in many-core systems, memory scalability can become a significant bottleneck.

Massively parallel GPU architectures deliver better performance than traditional CPU architectures, scaling with the number of cores while maintaining reasonable flexibility for implementing different sequence alignment algorithms. As a result, GPU-based sequence alignment implementations [1, 18, 40] often deliver higher throughput than CPU implementations. However, GPU architectures are optimized for processing regular and parallelizable workloads. Thus, GPUs are not well-suited for inherently sequential tasks, like the traceback, or irregular tasks, like X-drop, as these often lead to warp divergence and performance bottlenecks. Moreover, similar to CPUs, GPUs face performance overheads from irregular memory and control flow instructions.

Custom ISA extensions for CPUs [30] and GPUs [85, 89] enhance sequence alignment performance while preserving general-purpose flexibility. These ISA extensions introduce specialized instructions to improve the performance of the DP-matrix computation. Moreover, these ISA functional units consume minimal area, making them cost-effective. Despite these advantages, the architectural constraints of general-purpose cores, such as area and latency,

limit ISA extensions' performance and the complexity of the operations they can implement. For instance, in the case of the GMX, the overheads from control-flow and memory instructions on the CPU limit the utilization of GMX specialized functional units to just 10–20% (Section 11). Another example, in terms of operation complexity, is Nvidia DPX [85], which focuses on a limited set of operations, primarily combining maximum functions.

Specialized standalone DSAs [15, 36, 37, 66, 101] seek to address performance and efficiency limitations of general-purpose architectures, proposing tailored architectural designs optimized for specific end-to-end sequence alignment algorithms. DSA's specialized design enables high throughput DP-matrix computations and energy efficiency, utilizing parallel processing units and optimized control logic. As standalone designs, DSAs can scale up easily by replicating their processing units. Despite their efficiency, DSAs have limited flexibility and reusability since they are highly specialized for specific alignment models, algorithms, and applications (e.g., edit-distance alignment for 2-bit DNA [15]). Designs like GACT (Darwin's aligner) and GenASM achieve high efficiency by using heuristics, such as the window heuristic, to reduce memory usage. However, this restricts their use to cases where some loss in accuracy is acceptable (e.g., adaptive sampling), and limits their applicability to accelerate full genomics pipelines [52, 75, 105].

While standalone DSAs aim to minimize the DP-elements stored for traceback reconstruction, they dedicate significant silicon area to traceback logic and memory (e.g., 79.4% in GACT and 81.4% in GenASM). Although necessary for obtaining the alignment path, this logic and memory consume substantial resources without significantly accelerating computation, as traceback is an inherently sequential and irregular operation. Moreover, in designs like GACT and GenASM, the traceback of each window is mandatory to determine the next window position to be computed, even when the alignment path is unnecessary for the specific use case.

3.1 Our Goal

Our goal is to overcome the limitations of traditional accelerators and leverage their advantages by designing a heterogeneous architecture (SMX) that combines: (1) SMX-1D, a flexible yet efficient ISA extension to accelerate irregular and sequential alignment tasks, like traceback; and (2) SMX-2D, a highly optimized coprocessor that accelerates regular and highly-parallel tasks, like computing DP-matrix. This architecture must provide flexibility to accelerate modern sequence alignment algorithms while being versatile enough to adapt to a wide range of real-world use cases. Furthermore, this study aims to contribute to the ongoing discussion on the trade-offs between flexibility and efficiency in DSAs.

4 SMX-1D: A Flexible Alignment ISA

Classical DP-based alignment algorithms, such as Smith-Waterman (SW) and Needleman-Wunsch (NW), typically compute the DP-matrix element by element, as illustrated in Figure 4.a. We propose SMX-1D, an efficient and flexible instruction set extension designed to accelerate sequence alignment. SMX-1D enables the computation of an entire column vector, containing VL DP-elements, in a single operation (Figure 4.b). The value of VL depends on the size of each DP-element and can be configured to 2-bit, 4-bit, 6-bit, or 8-bit element width (EW) based on the specific requirements of each sequence alignment problem. Therefore, a 64-bit processor utilizing SMX-1D can compute 32, 16, 10, or 8 DP-elements concurrently, depending on the selected configuration. As a result, SMX-1D reduces the number of executed instructions by a factor of 8×-32× and, thanks to the differential encoding, reduces the memory footprint by 2×-8× compared to traditional methods that use 32-bits per DP-element.

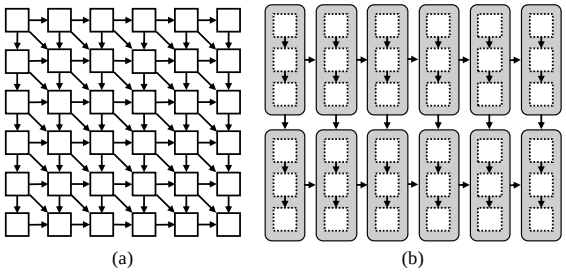


Figure 4: (a) Element-by-element computation of the classical DP algorithm. (b) SMX-1D column-vector instruction computation using a VL of 3 elements.

4.1 SMX Differential Encoding

Computing and storing integer values of Eq. 2 results in low computation efficiency as the maximum DP-element value increases linearly with the DP-matrix size. To address this, we employ a differential-encoding scheme [63, 99] which expresses each DP-element as a difference relative to its neighbors and allows packing multiple elements into an SMX-1D vector to efficiently process more DP-elements per SMX-1D operation.

Software implementations using differential-encoding are constrained to a minimum of signed 8-bit representation for each DP-element [63, 99]. To improve hardware efficiency, we apply linear transformation on Eq. 3 and 4 by shifting the Δv and Δh values, so that all intermediate results are non-negative, and propose a design that adapts to a runtime-configurable element width (EW ∈ 2-bit, 4-bit, 6-bit, 8-bit), balancing flexibility and hardware cost.

The shifting of Eq. 3 and 4, similar to [33, 72], leads us to new non-negative differentially-encoded scores $\Delta v' = \Delta v - D$ and $\Delta h' = \Delta h - I$ and new substitution-matrix penalties $S'(q_{i-1}, r_{j-1}) = S(q_{i-1}, r_{j-1}) - D - I$. Eq. 5 and 6 show the new recurrence equations to compute the DP-matrix.

$$\Delta v'_{i,j} = max\{S'(i-1, j-1) - \Delta h'_{i-1,j}, \Delta v'_{i,j-1} - \Delta h'_{i-1,j}, 0\} \quad (5)$$

$$\Delta h'_{i,j} = max\{S'(i-1, j-1) - \Delta v'_{i,j-1}, 0, \Delta h'_{i-1,j} - \Delta v'_{i,j-1}\} \quad (6)$$

By construction, $\Delta v'$ and $\Delta h'$ lie in the range $[0, \theta]$, where $\theta = S_{\max} - I - D$ and $S_{\max}$ is the maximum substitution-matrix penalty. Each value therefore requires at most $\lceil \log_2(\theta + 1) \rceil$ bits. Using this bounded representation, we can select the minimal bit-width needed for each alignment task, ensuring that no truncation or overflow occurs, as long as the bit-width is configured to cover the range up to θ (see proof on [63, 99]). In typical applications, θ and the alphabet size remain within 8 bits; BLOSUM and PAM matrices require 5–6 bits, and gap penalties require fewer, making narrow-width encoding both practical and precise.

This transformation also simplifies the datapath. Since operands in Eq. 5 and 6 are non-negative and one term is always zero, the max operation can be implemented using four subtractions and a pair of 3-to-1 multiplexers. As shown in Fig. 5, the sign bits from these subtractions directly control the selection lines, avoiding the four explicit signed comparisons otherwise required for Eq. 3 and 4. Furthermore, the equations are mutually dependent. If the first term is selected in one equation, it is also selected in the other, allowing reuse of control logic across both $\Delta v'$ and $\Delta h'$ computations.

4.2 SMX-1D ISA Semantics and Architecture

The SMX-1D ISA extension provides two specialized instructions (`smx.v` and `smx.h`) to accelerate the DP-matrix computation, two support operation instructions (`smx.redsum` and `smx.pack`), 3 configuration registers (`smx_query`, `smx_reference`, `smx_config`, and one 78 × 64-bit memory `smx_submat`). SMX-1D instructions are register-to-register instructions and use standard R-type RISC-V encoding with reserved custom opcodes.

Let $\Delta V_i = [\Delta v_i \dots \Delta v_{i+\mathtt{VL}-1}]$ be a vector containing `VL` elements of `EW` bits representing the vertical Δ values (i.e., differential-encoded DP-elements), and Δh_i be a single horizontal Δ value of `EW` bits. SMX-1D implements the following instructions.

- `smx.v rd, rs1, rs2`. Computes a column vector ΔV_o of `VL` elements, each `EW` bits wide, from ΔV_i and Δh_i, stored in registers `rs1` and `rs2`, using `smx_query` and `smx_reference`. Writes the result to register `rd`.
- `smx.h rd, rs1, rs2`. Computes a scalar Δh_o value from ΔV_i and Δh_i, stored in registers `rs1` and `rs2`, using `smx_query` and `smx_reference`. Writes the result to register `rd`.
- `smx.redsum rd, rs1`. Computes the sum reduction of all `VL` differentially encoded scores Δ_i stored in register `rs1`. Writes the result to register `rd`.
- `smx.pack rd, rs1`. Packs an 8-character ASCII string from the source register `rs1` into a packed representation based on the element width (`EW`) specified in the configuration register. Writes the result to register `rd`.

SMX requires three 64-bit architectural state registers (`smx_query`, `smx_reference`, and `smx_config`). Architectural registers are accessed using standard read-and-write instructions implemented in conventional ISAs, such as the `csrr`/`csrw` instructions on RISC-V. SMX-1D implements the following *Architectural State Registers*:

- `smx_query`: Stores a packed query subsequence of `VL` elements to be used by the SMX-1D unit.
- `smx_reference`: Stores a packed reference subsequence of `VL` elements to be used by the SMX-1D unit.
- `smx_config`: Stores the configuration of the SMX-1D functional unit. The SMX-1D configuration includes the element with (`EW`), score model (match/mismatch or substitution-matrix), and match, mismatch, and indel scores.

The `smx_query` and `smx_reference` registers are frequently written, requiring the same recovery mechanisms used for general-purpose registers, such as register renaming. This allows SMX-1D to be implemented in out-of-order processors. In contrast, the `smx_config` register and `smx_submat` memory are rarely modified, as their values are reused across all alignments within the same application. Therefore, to simplify the implementation, these registers can be updated at commit, eliminating the need for recovery mechanisms. This reduces both the implementation and verification complexity in a complex processor. SMX-1D uses two separate instructions (`smx.v` and `smx.h`) to compute DP-matrix columns, making the extensions suitable for simple RISC-like CPUs with a single destination register port. However, this approach does introduce redundancy, similar to the `mul` and `mulh` instructions in RISC-V. For CPUs with two destination register ports, `smx.v` and `smx.h` can be merged, enhancing encoding efficiency and throughput.

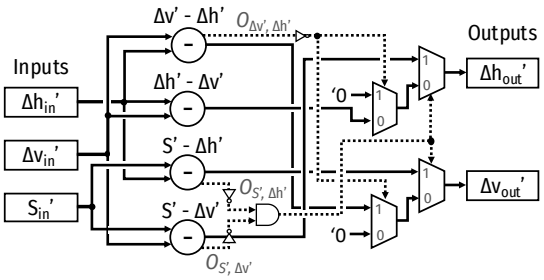


Figure 5: SMX-PE microarchitecture.

Along with its architectural registers, the SMX-1D unit requires a 78 × 64-bit memory block (`smx_submat`) to store the packed form of a 26 × 26 × 6-bit substitution matrix. This matrix is used for aligning sequences with a 26-character alphabet, with match/mismatch scores encoded using 6-bit values. The full matrix is serialized into 78×64-bit words, with 3 words allocated per column.

4.3 SMX-1D ISA Hardware Implementation

SMX-1D is designed to be an ISA extension implemented within the processor pipeline. Therefore, SMX-1D implementation has to meet specific hardware constraints: (1) small area footprint, (2) short execution latency (i.e., few clock cycles), (3) use the processor's general-purpose registers efficiently, and (4) operate at the same high-frequency as the processor. The SMX-1D implementation adopts the structure of a functional unit, connected similarly to the other functional units in the CPU, such as the ALU.

4.3.1 SMX Processing Element (SMX-PE). The SMX Processing Element (SMX-PE) is the hardware module responsible for computing a single DP-element of `EW` bits (encoded in Δv_{out} and Δh_{out}) using left Δv_{in}, upper Δh_{in}, and S'_{in} (Figure 5). It implements the optimized differential equations 5 and 6 with four hardware subtractors and two multiplexers, one for each Δ. The multiplexers' selection depends on overflow bits generated by the four subtractions. The overflow bit $O_{x,y}$ is defined as the `EW`-th bit of the x and y subtraction, which then controls the multiplexers as detailed in Figure 5.

4.3.2 SMX-1D Array Implementation. SMX-1D computation module is implemented by one array of SMX-PE$_{EW}$ units for every `EW` configuration. Since SMX supports `EW` values of 2-bit, 4-bit, 6-bit, and 8-bit, the SMX-1D module includes four corresponding arrays: 32×SMX-PE$_2$, 16×SMX-PE$_4$, 10×SMX-PE$_6$, and 8×SMX-PE$_8$. This design allows the module to adapt to different sequence alignment scenarios (e.g., DNA, protein, text) and efficiently handle various arithmetic precisions (e.g., edit-distance, gap model, protein model).

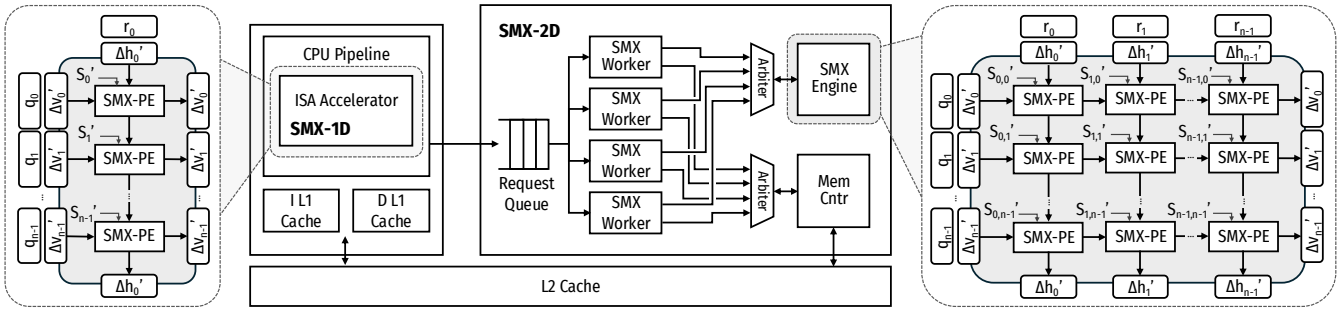


Figure 6: High-level diagram of the heterogeneous SMX architecture.

The left side of Figure 6 shows the SMX-PE interconnection in the SMX-1D module. The input $\Delta V'$ is obtained from register `rs1` and passed to the corresponding SMX-PE. The input $\Delta h'$ is obtained from register `rs2` and provided to the first SMX-PE, while each subsequent SMX-PE receives its $\Delta h'$ from the previous unit's output. The results are stored in `rd`: the `smx.v` instruction collects all $\Delta v'$ outputs, whereas the `smx.h` instruction stores only the last computed $\Delta h'$.

4.3.3 Efficient $S'(i, j)$ Computation. The input S' for each SMX-PE is generated dynamically using query and reference values in the `smx.query` and `smx.reference` registers. The SMX-1D module supports two different configurations for generating these substitution scores based on the sequence alignment problem being solved.

1. The match-mismatch configuration assigns fixed penalties to matching or mismatching characters and is commonly used in edit-distance calculations and algorithms like Needleman-Wunsch for comparing DNA, RNA, and ASCII sequences. This configuration is implemented as an array of comparators that evaluate characters from the `smx.query` and `smx.reference` registers. Each comparator in the array outputs $M - I - D$ (i.e., θ) if the reference and query characters match at that position or $X - I - D$ otherwise.

2. The substitution-matrix configuration assigns penalties to character substitutions using matrices like BLOSUM and PAM, which contain 26×26 penalty values ranging from -6 to 15. These matrices are typically used with indel costs between 5 and 12, leading to a maximum θ of 39, which can be encoded in 6 bits.

To support this, SMX-1D implements a dedicated memory structure storing 26×26 elements, each 6 bits wide, totaling $26 \times 26 \times 6$ bits. To optimize access, the matrix is structured into 26 rows, each corresponding to a reference character, with 26 words per row representing query characters. This layout allows for efficient SRAM implementation, as only one reference character needs to be accessed at a time. The memory access process first selects the SRAM row for the given reference character, outputting a 26×6-bit array. Then, the relevant `VL` elements corresponding to query characters are extracted, forming a `VL` $\times$ 6-bit vector. This organization minimizes latency and ensures rapid access during protein alignment.

5 SMX-2D: A High-Performance Coprocessor

SMX-1D ISA extensions enhance DP-matrix computation by processing VL DP-elements per operation while minimizing memory footprint to `EW`. However, SMX-1D faces certain limitations to scale efficiently to process more DP-elements. As an ISA extension, its implementation is constrained by the resources and capabilities of the general-purpose core it integrates with (e.g., area, latency, and power). Also, the core still requires executing memory and control-flow instructions, which ultimately limits the core from reaching the peak performance of SMX-1D functional units (i.e., reaching up 20% of the peak throughput).

To further accelerate the $O(n^2)$ DP-matrix computation, we propose SMX-2D, a high-performance coprocessor for computing entire DP-matrix blocks (DP-block). Our design allows the core to offload the computation of arbitrarily large DP-blocks to the SMX-2D coprocessor to accelerate the most compute-intensive step of sequence alignment algorithms. Internally, SMX-2D is designed as a coprocessor connected to the L2 cache that scales up the SMX-1D array design into a 2D matrix of SMX Processing Elements (SMX-PE). SMX-2D computes tiles of DP-elements (DP-tiles) using its 2D matrix array until the entire DP-block is calculated (see Figure 6). Thanks to its 2D array, specialized control logic and memory pipeline, the SMX-2D coprocessor is expected to deliver up to 128× higher throughput than the SMX-1D ISA.

Storing all computed DP-elements for large DP-blocks would heavily strain memory resources. To mitigate this, SMX-2D retains only the border DP-elements of each DP-tile, enabling efficient computation of subsequent tiles and allowing on-demand recomputation of inner DP-elements during traceback. Therefore, SMX-2D can reduce the memory footprint up to 32× compared to SMX-1D, and up to 256× compared to the software implementation.

5.1 SMX-2D Coprocessor Architecture

SMX-2D coprocessor architecture consists of three main components: SMX-engine, SMX-workers, and the memory controller (central part of Figure 6). First, the SMX-engine is responsible for computing DP-tiles of size `VL`×`VL` elements until the complete DP-block is calculated. Meanwhile, SMX-workers manage DP-block execution by partitioning it into tiles, issuing computations to the SMX-engine, and handling memory transfers. SMX-2D implements multiple SMX-workers to allow multiple DP-block computations in parallel and maximize the SMX-engine utilization. The memory controller facilitates communication with the L2 cache, efficiently sharing a single L2 request port with the CPU using an arbiter. Experimental results show that even at full occupancy, SMX-2D utilizes only 25% of the L2 port, while the CPU's usage remains at 2%, ensuring minimal performance impact.

In more detail, SMX-2D execution begins when the core offloads a DP-block computation to a specific SMX-worker by writing the configuration registers to set reference/query addresses, sizes, Δ matrix addresses, and other parameters (e.g., EW and penalties). Then, the SMX-worker starts by generating a read request to the memory controller to fetch segments of the input sequences and the Δ matrices. Once all required data is received, the SMX-worker issues DP-tile computation requests to the SMX-engine. Once completed, the SMX-engine returns the computed results (i.e., Δ values of the tile borders) to the SMX-worker to store them via the memory controller. The SMX-worker continues this process, sending computations to the SMX-engine, until the entire DP-block is computed. Finally, the SMX-worker signals completion, allowing the core to retrieve results for further processing, such as computing the final alignment score or performing traceback.

5.2 SMX-Engine Design

Extending the SMX-1D compute array design, the SMX-engine consists of a 2D matrix of SMX-PE$_{EW}$ units for every EW configuration. Since SMX allows EW to be configured to 2-bit, 4-bit, 6-bit, and 8-bit, the SMX-engine contains four arrays of 32×32 SMX-PE$_2$, 16×16 SMX-PE$_4$, 10×10 SMX-PE$_6$, and 8×8 SMX-PE$_8$. As a result, the SMX-engine computes an entire DP-tile of size 32×32, 16×16, 10×10, and 8×8 elements per cycle, depending on the EW configured.

Interconnection between SMX-PEs within the SMX-engine is shown in the right side of Figure 6. Each SMX-PE generates $\Delta h'$ and $\Delta v'$ as outputs, which serve as inputs to the adjacent SMX-PE below and to the right, respectively. The input vector $\Delta V'_{in}$ (loaded from rs1 register) supplies values to the leftmost column of SMX-PEs, and the input vector $\Delta H'_{in}$ (loaded from the rs2 register) initializes the top row of SMX-PEs. Similarly, SMX-engine generates the output vector $\Delta V'_{out}$ (output $\Delta v'$ values from the rightmost column of SMX-PE) and the output vector $\Delta H'_{out}$ (output $\Delta h'$ values from the bottom row of SMX-PE).

The SMX-engine is segmented into multiple pipelined stages since the SMX-engine contains a large 2D matrix of SMX-PE units and each unit adds a non-negligible propagation delay. Specifically, our SMX-engine design incorporates segmentation registers along the 2D matrix antidiagonals to effectively reduce propagation delay while maintaining a throughput of one DP-tile per cycle.

For efficient computation of the $S'(i, j)$ values, the SMX-engine extends the approach described for the SMX-1D ISA implementation. When using the match-mismatch model, the SMX-engine uses a 2D matrix of comparators to compare reference and query characters and determine whether to assign either a match or mismatch value to each SMX-PE. When aligning 8-bit amino acids using a substitution-matrix, the SMX-engine has to select $S'(i, j)$ values to feed the 10×10 matrix of SMX-PEs. For that, the SMX-engine first accesses simultaneously 10 columns from the substitution-matrix, each corresponding to a character in the reference subsequence. Then, for each of these 10 rows of 26×6-bit values, the SMX-engine selects the corresponding value based on the query characters (as in the SMX-1D implementation). Due to the need to perform 10 accesses in parallel, using SRAMs is not an option, as SRAMs typically support only single-row access at a time. Therefore, our SMX-engine design stores the substitution-matrix in registers.

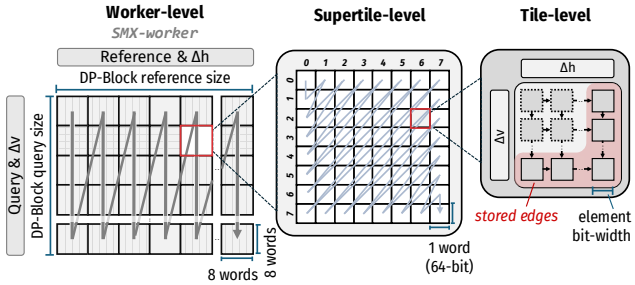


Figure 7: SMX-Worker computation pattern.

5.3 SMX-Workers: Maximizing Utilization

SMX-engine's design has a peak throughput of one DP-tile computed per cycle, but 100% utilization is challenging due to stalls from data dependencies between tiles and memory access delays.

To mitigate these issues, we introduce the SMX-worker as a specialized control unit designed to manage DP-blocks computations and guarantee a continuous flow of requests to the SMX-engine. The SMX-worker comprises dedicated memory and custom control logic to coordinate a DP-block alignment.

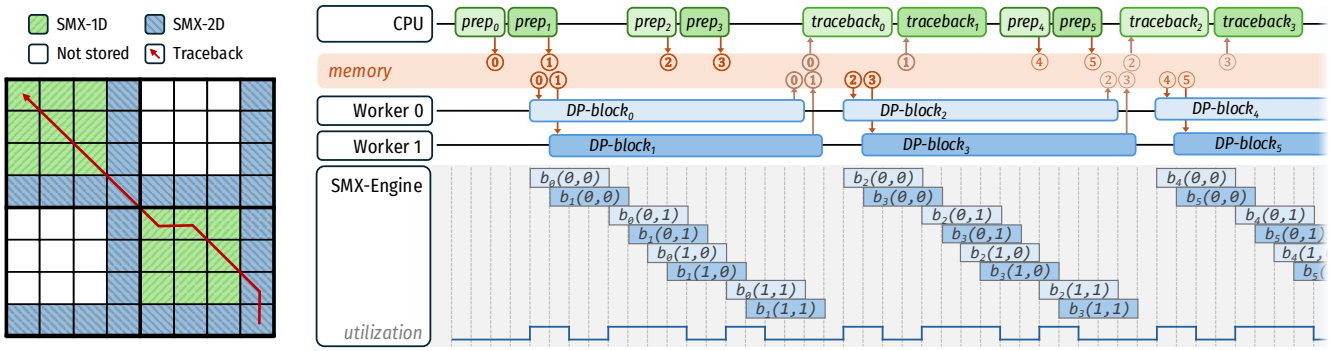
To exploit memory locality, the SMX-worker groups DP-tiles that share reference and query cache lines into *SMX-supertiles* (see Figure 7). It first loads a full cache line with reference and query data along with supertile boundary Δ values, then processes all inner DP-tiles using the SMX-engine while storing tile edges in internal SRAM. Tiles within supertiles are processed along antidiagonals to maximize parallelism. SMX-worker stores the computed tile borders, properly arranged into cache line transfers, minimizing the memory operations used.

Although the SMX-worker improves SMX-engine utilization, stalls may still occur due to memory access delays, task switching, and the variable number of parallel DP-tiles in first and last antidiagonals. To further improve utilization, SMX-2D incorporates multiple SMX-workers that process different DP-blocks in parallel, reducing idle cycles and hiding memory latencies.

6 SMX Heterogeneous Architecture

SMX heterogeneous architecture combines a CPU extended with the SMX-1D ISA with an SMX-2D coprocessor to accelerate different sequence alignment algorithms. While SMX-2D specializes in high-throughput computations of 2D DP-blocks, the SMX-1D-enhanced core manages irregular and control-intensive tasks, including traceback computation, work scheduling, and heuristics.

Figure 8.a illustrates the computation of a complete DP-block, represented as a 2×2 array of DP-tiles. In the example, the tile size is 4x4 DP-elements. The core first packs the input sequences using a custom packing instruction and offloads the DP-block computation to the SMX-2D coprocessor. The coprocessor computes all tiles, storing only their borders (blue elements in Figure 8.a). The core then performs traceback, starting from the bottom-right DP-element. Since inner DP-elements are not stored, the core selectively recomputes them using SMX-1D instructions (green elements) only



(a) (b)

Figure 8: (a) Computation of a DP-block (2x2 DP-tiles of 4x4 DP-elements) using SMX heterogeneous architecture. (b) Timeline of SMX architecture processing DP-blocks of 2x2 DP-tiles. The SMX-2D has two SMX-workers and a 4-stage SMX-engine. $prep_i$ indicates the sequence preprocessing (e.g., packing the strings) needed for $DP{-}block_i$. $traceback_i$ indicates that the CPU is computing the traceback of $DP{-}block_i$. $b_i(x,y)$ indicates that the SMX-engine is computing the tile (x,y) of the $DP{-}block_i$.

for tiles involved in the traceback. This minimizes redundant computations while reconstructing the alignment path (red arrows), ultimately reaching the top-left corner.

For the applications where only the alignment score is needed, the SMX-2D coprocessor stores only the last column of the DP-block using differential encoding. The core then sums all Δh values along the first row and Δv along the last column to obtain the alignment score, corresponding to the bottom-right element of the DP-block.

The SMX-2D coprocessor and the core interleave tasks to execute simultaneously, enhancing performance. Figure 8.b illustrates a timeline where the core and SMX-2D process multiple small DP-blocks of 2×2 tiles. In this illustration, SMX-2D features two SMX-workers and a single SMX-engine with a 4-stage pipeline. First, the CPU packs the reference and query sequences for two DP-blocks and dispatches them to the coprocessor, assigning one DP-block to each worker. The workers then issue tile computation requests to the SMX-engine while handling data dependencies (e.g., tiles (0,0) and (1,0)). If a tile depends on a previously computed one, the worker stalls until the dependency is resolved. Conversely, independent tiles (e.g., tiles (1,0) and (0,1)) can be processed concurrently within the SMX-engine pipeline. In this example, the SMX-engine achieves around 50% utilization due to the limited number of tiles and workers. Increasing DP-block size and adding more workers can push utilization closer to 100%. Once a DP-block is computed, the core can initiate the traceback using the SMX-1D ISA. The heterogeneous SMX design enables overlapping sequence packing and traceback computation in the core with DP-block computation in SMX-2D, ensuring continuous execution of both units and maximizing overall throughput.

7 Experimental Methodology

Cycle-Accurate Simulations: For experimental evaluation, we implement the SMX heterogeneous architecture into the gem5 RISC-V simulator [10]. Using gem5, we simulate an SoC with 8 cores extended with SMX-1D ISA and 8 SMX-2D coprocessors connected to each core's private L2 cache. Each RISC-V core is an 8-way superscalar out-of-order with the configuration shown in Table 1. The SMX-2D coprocessor is configured with 4 SMX-workers to maximize SMX-engine utilization while minimizing the silicon area utilization (see Section 10 for area and frequency results).

Synthesis and Physical Design Environment: We integrate an RTL implementation of SMX-1D and SMX-2D into a RISC-V 64-bit Linux-capable edge processor featuring an in-order single-core processor fabricated in GlobalFoundries' 22nm technology node (Table 2 shows the processor configuration). To achieve the 1 GHz target frequency of the RTL design, we configure the SMX-engine module to obtain a design with 7, 5, 4, and 3 cycles operation latency for the 2-bit, 4-bit, 6-bit, and 8-bit configurations, respectively. The processor with SMX is synthesized in GlobalFoundries' 22nm FD-SOI using Cadence Genus v19.11 and placed and routed with Innovus v22.33, targeting a 1 GHz post-PnR clock.

Sequence alignment configurations: For evaluating SMX on different use cases, we define four sequence alignment model configurations: DNA-edit (2-bit characters using edit distance), DNA-gap (4-bit characters using a linear gap model), Protein (6-bit characters using a linear gap model and BLOSUM50 substitution-matrix), and ASCII (8-bit characters using edit distance).

Implementations: We evaluate SMX's performance by comparing different implementations, computing only the alignment

Table 1: Gem5 Out-of-order processor configuration.

Pipeline	64-bit RISC-V (RV64GV), 8-wide out-of-order, 8k-entry Bi-Mode predictor, 256-bit SIMD unit
Memory Unit	2 loads and 1 store per cycle, 96-entry LSQ, 96-entry SB
iTLB & dTLB	Fully associative, 148 entries per TLB
Data L1	64 KB, 4-way, 3 cycles latency, 256 MSHR
Inst. L1	64 KB, 4-way, 2 cycles latency, 256 MSHR
Private L2	1MB, 8-way, 160 MSHR
Shared LLC	1MB per core, 16-way, 256 MSHR
Main memory	8 GB of DDR4 with 23.9 GB/s bandwidth

Table 2: RTL In-Order processor configuration

Pipeline	64-bit RISC-V (RV64GV), 7-stages, 128-entry bimodal predictor, 32-entry graduation list, 128-bit SIMD unit
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, PIPT, 16-entry MSHR
Inst. cache	16 KB 4-way, 2-cycle, VIPT

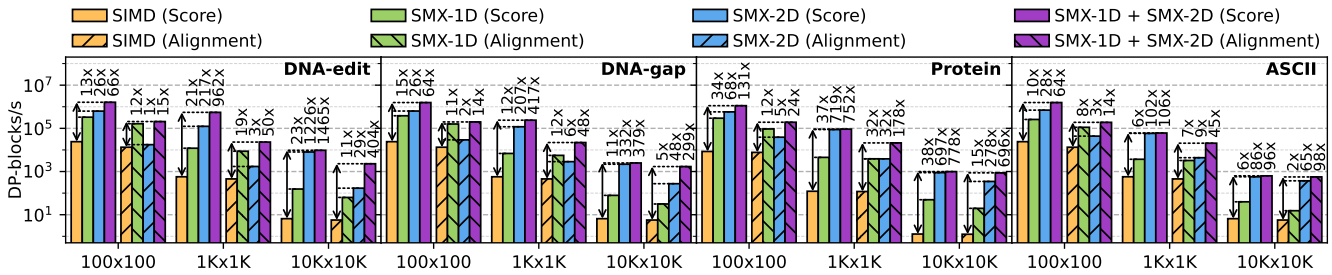


Figure 9: Throughput SMX-1D and SMX-2D aligning different length DP-block for 4 sequence alignment configurations.

score (Score) and the full alignment (Alignment). Our baseline, *SIMD*, is the highly optimized and 128-bit vectorized KSW2 implementation found at the core of Minimap2 [63], one of the most used state-of-the-art read mappers. *SMX-1D* implementation uses the SMX-1D ISA extension to accelerate both DP-block computations and traceback. *SMX-2D* implementation accelerates only the DP-block computations using the SMX-2D coprocessor, while the general-purpose core performs the traceback step. Lastly, *SMX* implementation uses the SMX-2D coprocessor for DP-matrix computation and the SMX-1D ISA for pre/post-processing tasks, such as the traceback computation.

Experimental Datasets: Using the methodology from [44, 71, 72], we generated different datasets, each corresponding to a use case (i.e., DNA-edit, DNA-gap, Protein, and ASCII), containing sequences of lengths of 100, 1K, and 10K base-pairs (bps). Additionally, for the evaluation of practical sequence alignment algorithms, we used two real DNA-sequencing datasets (≈15Kbps sequences of PacBio-HiFi [104] and ≈50Kbps sequences of ONT-Nanopore [11]) and a protein dataset containing various pairwise alignments generated by querying proteins against the UniProt database [24].

8 SMX Performance Analysis

This section presents performance results aligning DP-blocks of different sizes in various use cases. The experiments are evaluated on the gem5 simulator using a single SMX-enhanced out-of-order core. Figure 9 shows the throughput (DP-blocks per second) obtained when aligning DP-blocks of sizes 100×100, 1K×1K, and 10K×10K; using the different implementations (SIMD, SMX-1D, SMX-2D, and SMX, which combines SMX-1D and SMX-2D) across four configurations (DNA-edit, DNA-gap, Protein, and ASCII).

When computing the alignment score, all implementations fit data in the cache hierarchy. Compared to the SIMD baseline, SMX-1D accelerates DNA-edit, DNA-gap, protein, and ASCII alignments by up to 23×, 11×, 16×, and 6×, respectively, thanks to its efficient packing of 8 to 32 DP-elements per word, enabling parallel computation in a single instruction. In contrast, SIMD packs 16 DP-elements per vector but requires 9 arithmetic SIMD instructions per computed vector, explaining SMX-1D's performance advantage. Protein alignment benefits particularly from SMX-1D, as the SIMD baseline suffers from frequent random accesses to the substitution matrix, significantly degrading performance. SMX-2D and SMX (SMX-1D + SMX-2D) achieve similar peak speed-ups across all use cases, reaching up to 1465×, 379×, 778×, and 96× over SIMD for 10K×10K DP-blocks. These results demonstrate the advantages of the 2D systolic array (up to 32x higher parallelism than SMX-1D) and optimized control logic in improving system utilization (up to 4x higher utilization compared to SMX-1D). For smaller DP-blocks, however, SMX-2D alone does not match SMX's performance due to pre/post-computation bottlenecks in the core, which are efficiently handled by SMX-1D. As shown in Figure 9, SMX-based implementations achieve the highest speed-ups for long sequences, where most execution time is spent on DP computation. The results also highlight the importance of configurable `EW` and `VL` values for performance. Specifically, for long sequences, SMX-1D scales linearly in performance, while SMX-2D and SMX exhibit quadratic scalability as `VL` increases.

When computing full alignments (including traceback), SMX-1D improves performance over the SIMD baseline by up to 18×, 12×, 8×, and 7× for DNA-edit, DNA-gap, protein, and ASCII alignments, respectively. These speedups hold when aligning 1K×1K DP-blocks that fit within the cache hierarchy. For longer sequences, neither SIMD nor SMX-1D can keep DP-blocks in cache, leading to significant performance degradation. SMX-2D accelerates DP-matrix computations but remains bottlenecked by the CPU, which handles inner DP-element recomputation and traceback. This limitation is more pronounced for small DP-blocks and lower `EW` values, where the number of tiles to be recomputed is larger. However, as sequence length increases, DP-matrix computation dominates runtime, increasing SMX-2D's speedup. The SMX implementation overcomes these traceback bottlenecks by using the SMX-1D ISA to efficiently recompute inner DP-elements within tiles. Compared to the SIMD baseline, it achieves up to 404×, 299×, 696×, and 98× speedups for DNA-edit, DNA-gap, protein, and ASCII alignments, respectively.

8.1 SMX-2D Utilization Analysis

We evaluate SMX-engine utilization in the SMX-2D coprocessor with 1 to 8 workers to determine the optimal configuration. To accurately measure peak utilization, we compute only DP-block scores, minimizing CPU overhead. Figure 10 shows utilization across different worker configurations, model setups, and sequence lengths. A single worker achieves at most 30–45% utilization when aligning large DP-blocks, while increasing to 4 workers raises utilization around 90%, optimizing resource use and performance. Beyond 4 workers, performance gains are marginal, making the area increase unjustifiable. For very small DP-blocks (e.g., 100×100), utilization drops significantly due to high communication overhead.

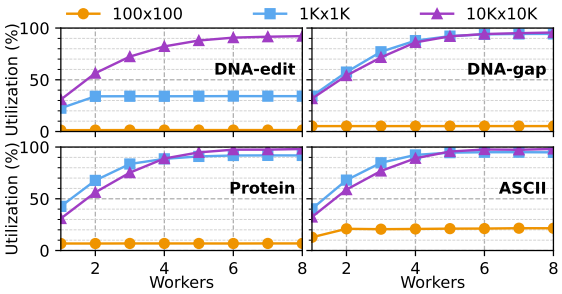


Figure 10: SMX-engine utilization by worker count

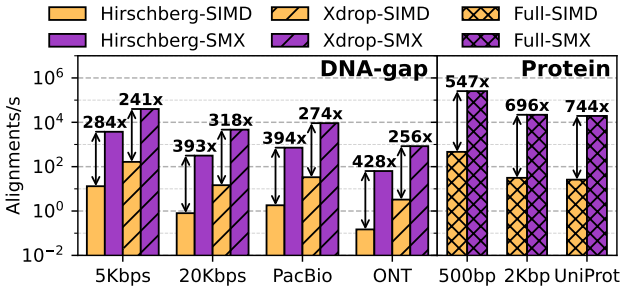


Figure 11: Throughput of SMX-accelerated algorithms.

9 SMX-Accelerated Algorithms

In this section, we show the acceleration of different practical algorithms leveraging SMX's heterogeneous architecture. As shown in Figure 11, SMX accelerates Hirschberg's recursive algorithm, achieving a 390× speedup on real DNA datasets. This is because Hirschberg's method spends most of its execution time recomputing arbitrarily large DP-blocks to divide the alignment into smaller subproblems, a task where SMX-2D excels. Similarly, Xdrop-SMX, an SMX-accelerated version of the banded Xdrop algorithm (as used in BLAST [7]), improves performance by 256× on real DNA sequencing datasets. While effective, its speedup is slightly lower than Hirschberg's due to the smaller DP-block sizes (columns sized by the supertile's width), which introduce additional CPU-coprocessor communication overhead. Lastly, Full-SMX, SMX-accelerated implementation of protein-to-protein alignment, yields a 744× speedup compared to the SIMD baseline. In this case, the SIMD baseline requires random access to the substitution-matrix for each DP-element, significantly reducing the SIMD performance.

9.1 SMX-Algorithms Multicore Scalability

Figure 12 (left panel) shows the scalability of SMX in a multicore system where each core is extended with the SMX-1D ISA and connected to an SMX-2D coprocessor via the private L2 cache. We observe that all SMX-accelerated alignment implementations scale linearly with the number of cores. Moreover, we can expect the same scalability in larger systems since all these different algorithms fit computed DP-blocks inside the private caches. Note that the Xdrop version shows slightly less efficient scaling than the other implementations due to the higher communication overheads between the CPU and the SMX-2D. Such overheads can lead to some congestion in the cache hierarchy, impacting scalability.

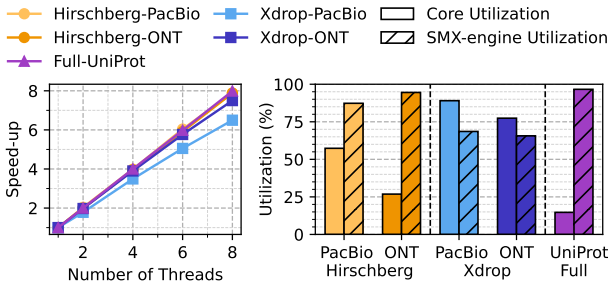


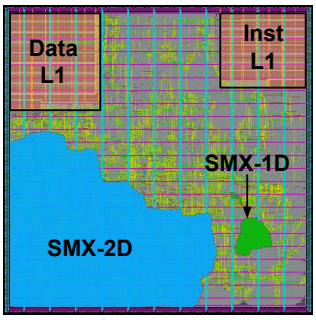
Figure 12: SMX-accelerated algorithms scalability on multicore (left) and core/SMX-engine utilization (right).

9.2 SMX-Algorithms Work Balance

This section shows the work balance between the core (with SMX-1D) and the SMX-2D coprocessor while executing various applications aligning real sequence datasets. Figure 12 (right panel) shows the percentage of time the core spends processing data (i.e., not waiting for the SMX-2D) and the utilization of the SMX-engine (i.e., the percentage of cycles where the SMX-engine receives new computation requests). In the case of SMX-accelerated Hirschberg, executions achieve high utilization of the SMX-engine by alternating low-intensive core-coprocessor coordination tasks and high-intensive traceback computations. Aligning larger sequences requires performing more subproblem divisions, reducing core activity during these steps. As can be observed in Figure 12, aligning the ONT dataset, which contains very long sequences, shows less core utilization than aligning the PacBio dataset, which contains shorter sequences. When executing the SMX-accelerated banded algorithm with Xdrop, we observe high utilization of both the core and the SMX-engine. In this case, the SMX-2D coprocessor constantly processes DP-blocks belonging to the DP-matrix band, and the core processes the results to determine whether the alignment should continue or be dropped while sending new DP-blocks to compute. Finally, when computing protein alignments using the UniProt dataset, the SMX-engine shows very high utilization while the CPU remains underutilized, since its role is limited to performing simple operations to process the results (i.e., a DP column reduction to calculate the alignment score).

9.3 Accelerating End-to-End Applications

In this section, we show the performance improvements that SMX can provide to end-to-end applications such as Minimap2 [63] for DNA read mapping and DIAMOND [12] for sensitive protein alignment. In the case of Minimap2, the alignment phase (DNA-gap + Banded + Xdrop) accounts for 70–76% of the total execution time when aligning PacBio sequences [55]. SMX accelerates Minimap2's alignment algorithm by 274×, resulting in an end-to-end application speedup of 3.3–4.1×. Similarly, in the case of DIAMOND protein aligner, alignment (Protein + BLOSUM + Banded) consumes around 99% of the total execution time [12]. SMX accelerates this DIAMOND's alignment algorithm by 744×, resulting in an end-to-end application speedup of 88.3×.



(a) Floorplan of SMX

Module	Area (mm^2)
SMX-PE array	0.005 (0.47%)
Other instructions	0.002 (0.13%)
Total SMX-1D	**0.015 (1.37%)**
SMX-engine	0.114 (10.28%)
SMX-PE matrix	0.072 (6.50%)
S'(i,j) generation	0.041 (3.71%)
SMX-worker (×4)	0.148 (13.36%)
Total SMX-2D	**0.328 (29.66%)**
Processor without SMX	0.763
64-bits Mul	0.009
Data L1 Cache	0.154
Total Processor	**1.106**

(b) Area breakdown

Figure 13: Physical design of the SMX-enhanced Processor

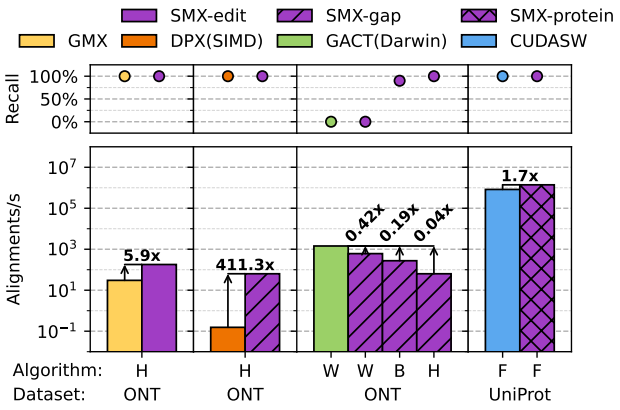


Figure 14: Performance comparison (alignments/s and recall) of SMX against the state-of-the-art methods. (H) Hirschberg, (X) Banded with Xdrop of 8%, (W) Window (W=320, O=128), (F) Full alignment.

10 SMX Synthesis, Area, and Frequency

Table 13b shows the area breakdown of each SMX module after place and route at 1 GHz when implemented in an in-order single-issue RISC-V core using a 22nm technology node (Table 2 details the processor configuration). SMX-1D adds a minimal area overhead of 0.0152mm^2, just 1.37% of the processor area, which is comparable to a 2-cycle 64-bit integer multiplier. The SMX-2D coprocessor occupies 0.3280mm^2, with 0.1136mm^2 for the SMX-Engine and 0.0369mm^2 per SMX-Worker. Overall, SMX-2D accounts for 29.66% of the processor area, equivalent to 2.13× the 32KB L1 data cache. Regarding the power consumption, SMX consumes 0.342 mW assuming a 20% gate activity factor.

11 Performance Comparison with the SotA

This section compares SMX's performance against state-of-the-art methods. We conduct direct comparisons using identical datasets, use cases, and algorithms, leveraging SMX's flexibility to adapt to diverse sequence alignment scenarios. Figure 14 shows SMX's performance against GMX [30] and DPX [89] ISA extensions (DPX on CPU SIMD); GACT (from Darwin [101]) decoupled accelerator; and CUDASW++ [89] GPU library (using DPX). For the evaluation, we use ONT (DNA) and UniProt (protein) datasets. For fair comparison, GMX and DPX instructions were integrated into the same gem5-simulated processor as SMX, executing Hirschberg and Xdrop algorithms. GMX was added to the CPU's scalar pipeline using its original design [30]. DPX, originally designed for GPU vector units, was implemented on the CPU's SIMD unit.

Compared to GMX (ISA extension), SMX shows 5.9× higher throughput for the Hirschberg algorithm aligning DNA in edit distance. While both use 32x32 tiles, SMX achieves 82.4% tile occupancy versus GMX's 11.1%. As GMX is a pure ISA extension, CPU limitations (e.g., pipeline contention and data dependencies) prevent issuing one GMX instruction per cycle, lowering its throughput.

Compared to DPX (implemented as SIMD), SMX shows 411.3× higher throughput while accelerating the same Hirschberg algorithm in gapped DNA alignment. DPX's performance is limited by its design, which merges a few operations (e.g., max of three) into one instruction, resulting in only a 1.07× speedup over baseline KSW2. In contrast, SMX processes hundreds of DP-elements per cycle, significantly improving parallelism and throughput.

Compared to GACT (Darwin DSA), SMX is 2.4× slower when computing the same window heuristic (window size of W=320 and overlap of O=128), despite comparable silicon areas (GACT: 1.34mm^2 @40nm; ≈0.3mm^2 @22nm [97]). This is due to SMX's low efficiency when computing the small DP-blocks found in the window heuristic. However, as shown in Figure 14, the window heuristic fails to compute any optimal alignments from ONT sequences, resulting in zero recall. Unlike GACT, SMX's flexibility allows accelerating other algorithms, like the banded algorithm, which is 5.2× slower than GACT but correctly aligns 90% of ONT sequences. Moreover, SMX can also accelerate Hirschberg algorithm, which is 22.7× slower than GACT but guarantees to compute a 100% correct alignment (full recall), demonstrating SMX's performance-accuracy trade-off.

Compared to CUDASW++ (GPU library with DPX) running on an Nvidia H100 GPU (at 2 GHz), a 72-core SMX-enhanced Grace CPU (at 1 GHz) is projected to achieve 1.7× higher throughput computing protein alignments. Note that the die area of the Nvidia H100 GPU is comparable to that of the 72-core Grace CPU (both 800 mm^2), and that the area overhead of the SMX accelerators' (at 5 nm) is negligible compared to the Grace CPU's die area. Furthermore, the SMX-enhanced CPU configuration aligns better with current genomics software ecosystems, which are predominantly CPU-based, with limited support for GPU-accelerated applications. This makes SMX a more flexible and practical choice for accelerating a wide range of mainstream genomic applications and pipelines.

12 Related Work

Sequence alignment is critical in computational biology, leading to numerous efforts to accelerate it using general-purpose hardware. Many proposals focus on algorithmic improvements [98, 99], exploitation of SIMD instructions [63, 68], multicore parallel-processing [87], and GPUs acceleration [3, 89].

Recent ISA extension proposals, DPX [85, 89] for GPU and GMX [30] for CPUs, provide specialized instructions to enhance

Table 3: Peak GCUPS (PGCUPS) per Processing Units (PU).

Study	Device	Application* E	G	P	T	CPU	PGCUPS ✠ / CPU	mm² § / CPU
KSW2 [63]	CPU	✓	✓	✓	✓	1	1.8	–
BlockAligner [68]	CPU	✓	✓	✓	✓	1	3.6	–
GMX [30]	ISA	✓			✓	1	1024.0	0.02
Study	**Device**	**E**	**G**	**P**	**T**	**SM†**	**PGCUPS ✠ / SM**	**mm² § / SM**
GASAL2 [3]	GPU	✓	✓		✓	28	2.3	–
CUDASW++4 [89]	GPU (ISA)	✓	✓	✓		132	63.3	–
Study	**Device**	**E**	**G**	**P**	**T**	**Chip**	**PGCUPS ✠ / Chip**	**mm² § / Chip**
BioSEAL [56]	PIM	✓	✓	✓		15	6046.7	230.0
Study	**Device**	**E**	**G**	**P**	**T**	**PU†**	**PGCUPS ✠ / PU**	**mm² § / PU**
GENASM [15]	DSA	✓			✓	32	64.0	0.33
DARWIN [101]	DSA	✓	✓		✓	64	54.2	1.34
GenDP [45]	DSA	✓	✓		✓	64	4.7	5.39
Mao-Jan Lin [66]	DSA	✓	✓	✓	✓	1	91.4	5.72
Talco-XDrop [106]	DSA	✓	✓	✓	✓	32	12.8	1.82
SMX **DNA-edit**	ISA + Coproc.	✓			✓	1	1024.0	0.34
SMX **DNA-gap**		✓	✓		✓		256.0	
SMX **Protein**		✓	✓	✓	✓		100.0	
SMX **ASCII**		✓	✓		✓		64.0	

*E = Edit Model, G = Gap Model, P = Protein Model, T = Supports Traceback.
†SM = Streaming Multiprocessor, PU = Processing Units
✠ Peak GCUPS (Giga DP-elements computed per second) per processing unit.
§ Additional area (mm^2) required per processing unit.

sequence alignment efficiency. DPX introduces 3-way maximum and minimum operations, applicable across various DP-based applications, while GMX offers high-performance instructions for computing DP-tiles using the edit distance model for DNA alignment. In comparison, SMX-1D obtains a balance by delivering higher performance than DPX while offering greater flexibility than GMX across different sequence alignment algorithms.

Driven by the demand for faster sequence alignment solutions, researchers have proposed numerous domain-specific hardware accelerators [15, 37, 45, 66, 101]. SeedEx and GenASM optimize edit-distance DNA alignment, while Darwin accelerates DNA alignment using a gap model in an ASIC design. However, these accelerators rely on custom algorithms with restrictive heuristics tailored to specific applications in mind. On the other side, GenDP [45] presents a more flexible DSA thanks to its programmable ISA, allowing it to accelerate other DP problems. However, its performance per mm² is significantly lower than other sequence alignment DSAs. Despite the significance of protein alignment, few DSAs support the substitution matrices required for protein models. Mao-Jan Lin et al. [66] propose an accelerator that computes the entire DP-matrix, while Talco-XDrop [106] introduces a tiling-based method to reduce the number of computed and stored cells in protein alignment. Unlike other accelerators, SMX is not restricted to a single alignment model and can be configured to maximize performance across diverse scenarios. Moreover, SMX operators can be used to accelerate different DP-based alignment algorithms and heuristics.

Comparing sequence alignment proposals is challenging due to variations in algorithms, architectures, and technologies. GCUPS (Giga Cells/DP-elements Updated Per Second) remains a key metric for peak performance, reflecting the number of DP-elements processed per second. Table 3 summarizes key proposals, detailing Processing Units (PU), silicon area, and flexibility. In this comparison, we account for both the logic and the private memory required by each solution. For SMX, we include all logic and memory components of SMX-1D and SMX-2D. Overall, SMX achieves high GCUPS per processing unit compared to highly specialized DSAs while maintaining flexibility across sequence alignment models. It minimizes area overhead by using reduced datatypes and leveraging the core's memory hierarchy, eliminating the need for dedicated traceback memory. By storing only tile borders, SMX-2D significantly reduces memory footprint and bandwidth, optimizing system efficiency and enabling the core's memory hierarchy reuse.

13 Conclusions

In this paper, we present the SMX, a heterogeneous architecture for sequence alignment acceleration that combines flexible SMX-1D ISA extensions to accelerate irregular and sequential alignment tasks, and an efficient SMX-2D coprocessor to efficiently accelerate the regular and highly-parallel DP-matrix computations. We demonstrate the flexibility and efficiency of the SMX heterogeneous architecture by accelerating practical sequence alignment algorithms in different real-world use cases, exploring the trade-offs between flexibility and efficiency in domain-specific accelerators.

Acknowledgments

This work has been partially funded by the Spanish Ministry of Science and Innovation MCIN/AEI/10.13039/501100011033 and FSE+ [grant number PID2019-107255GB-C21, PID2020-113614RB-C21, TED2021-132634A-I00, and PID2023-146511NB-I00], the Spanish Ministry of Economy, Industry, and Competitiveness through an FPU fellowship [grant number FPU20-04076 to M.D.], Lenovo-BSC Contract-Framework Contract (2020), the NSF CSSI Award #2311890 and NSF PPoSS Award #2118709, the Spanish Ministry for Digital Transformation and Public Service, and the European Union – NextGenerationEU through the Càtedra Chip UPC project [grant number TSI-069100-2023-0015], the Spanish Ministry for Digital Transformation and Public Service, within the framework of the Recovery, Transformation and Resilience Plan - NextGenerationEU [REGAGE22e00058408992], and the Generalitat de Catalunya GenCat-DIUiE (GRR) [grant numbers 2021-SGR-00763, and 2021-SGR-00574].

References

[1] Quim Aguado-Puig, Max Doblas, Christos Matzoros, Antonio Espinosa, Juan Carlos Moure, Santiago Marco-Sola, and Miquel Moreto. 2023. WFA-GPU: gap-affine pairwise read-alignment using GPUs. *Bioinformatics* 39, 12 (11 2023), btad701. https://doi.org/10.1093/bioinformatics/btad701

[2] 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.

[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 (25 Oct 2019), 520. https://doi.org/10.1186/s12859-019-3086-9

[4] 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).

[5] 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.

[6] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. 1990. Basic local alignment search tool. *Journal of molecular biology* 215, 3 (1990), 403–410.

[7] Stephen F Altschul, Thomas L Madden, Alejandro A Schäffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J Lipman. 1997. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic acids research* 25, 17 (1997), 3389–3402.

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

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

[10] 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.

[11] Rory Bowden, Robert W Davies, Andreas Heger, Alistair T Pagnamenta, Mariateresa de Cesare, Laura E Oikkonen, Duncan Parkes, Colin Freeman, Fatima Dhalla, Smita Y Patel, et al. 2019. Sequencing of human genomes with nanopore technology. *Nature communications* 10, 1 (2019), 1869.

[12] Benjamin Buchfink, Klaus Reuter, and Hajk-Georg Drost. 2021. Sensitive protein alignments at tree-of-life scale using DIAMOND. *Nature methods* 18, 4 (2021), 366–368.

[13] Benjamin Buchfink, Chao Xie, and Daniel H Huson. 2015. Fast and sensitive protein alignment using DIAMOND. *Nature methods* 12, 1 (2015), 59–60.

[14] 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.

[15] 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.

[16] 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. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. 638–655.

[17] Christiam Camacho, George Coulouris, Vahram Avagyan, Ning Ma, Jason Papadopoulos, Kevin Bealer, and Thomas L Madden. 2009. BLAST+: architecture and applications. *BMC bioinformatics* 10 (2009), 1–9.

[18] 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* (Munich, Germany) *(ICS '14)*. Association for Computing Machinery, New York, NY, USA, 103–112. https://doi.org/10.1145/2597652.2597677

[19] Kun-Mao Chao, William R. Pearson, and Webb Miller. 1992. Aligning two sequences within a specified diagonal band. *Bioinformatics* 8, 5 (10 1992), 481–487. https://doi.org/10.1093/bioinformatics/8.5.481

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] The UniProt Consortium. 2022. UniProt: the Universal Protein Knowledgebase in 2023. *Nucleic Acids Research* 51, D1 (11 2022), D523–D531. https://doi.org/10.1093/nar/gkac1052

[25] UniProt Consortium. 2025. UniProtKB Release Statistics (All Releases Until 2025-01). https://ftp.uniprot.org/pub/databases/uniprot/previous_releases/ Accessed: 2025-02-13.

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

[27] Margaret Dayhoff, R Schwartz, and B Orcutt. 1978. A model of evolutionary change in proteins. *Atlas of protein sequence and structure* 5 (1978), 345–352.

[28] 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).

[29] Alexander Dobin, Carrie A Davis, Felix Schlesinger, Jorg Drenkow, Chris Zaleski, Sonali Jha, Philippe Batut, Mark Chaisson, and Thomas R Gingeras. 2013. STAR: ultrafast universal RNA-seq aligner. *Bioinformatics* 29, 1 (2013), 15–21.

[30] Max Doblas, Oscar Lostes-Cazorla, Quim Aguado-Puig, Nick Cebry, Pau Fontova-Musté, Christopher Frances Batten, Santiago Marco-Sola, and Miquel Moretó. 2023. Gmx: Instruction set extensions for fast, scalable, and efficient genome sequence alignment. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*. 1466–1480.

[31] Max Doblas, Oscar Lostes-Cazorla, Quim Aguado-Puig, Cristian Iñiguez, Miquel Moretó, and Santiago Marco-Sola. 2024. QuickEd: High-performance exact sequence alignment based on bound-and-align. *bioRxiv* (2024), 2024–09.

[32] Robert C Edgar. 2004. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic acids research* 32, 5 (2004), 1792–1797.

[33] 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.

[34] Michael Farrar. 2006. Striped Smith–Waterman speeds database searches six times over other SIMD implementations. *Bioinformatics* 23, 2 (11 2006), 156–161. https://doi.org/10.1093/bioinformatics/btl582

[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] 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.

[37] 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. https://doi.org/10.1109/MICRO50266.2020.00080

[38] Yang Gao, Xiaofei Yang, Hao Chen, Xinjiang Tan, Zhaoqing Yang, Lian Deng, Baonan Wang, Shuang Kong, Songyang Li, Yuhang Cui, Chang Lei, Yimin Wang, Yuwen Pan, Sen Ma, Hao Sun, Xiaohan Zhao, Yingbing Shi, Ziyi Yang, Dongdong Wu, Shaoyuan Wu, Xingming Zhao, Binyin Shi, Li Jin, Zhibin Hu, Chuangxue Mao, Shaohua Fan, Qiang Gao, Juncheng Dai, Fengxiao Bu, Guanglin He, Yang Wu, Huijun Yuan, Jinchen Li, Chao Chen, Jian Yang, Chaochun Wei, Xin Jin, Xia Shen, Yan Lu, Jiayou Chu, Kai Ye, Shuhua Xu, and Chinese Pangenome Consortium (CPC). 2023. A pangenome reference of 36 Chinese populations. *Nature* 619, 7968 (01 Jul 2023), 112–121. https://doi.org/10.1038/s41586-023-06173-7

[39] 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.

[40] Giulia Gerometta, Alberto Zeni, and Marco D. Santambrogio. 2023. TSUNAMI: A GPU Implementation of the WFA Algorithm. In *2023 32nd International Conference on Parallel Architectures and Compilation Techniques (PACT)*. 150–161. https://doi.org/10.1109/PACT58117.2023.00021

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

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

[43] Ragnar Groot Koerkamp. 2024. A*PA2: Up to 19× Faster Exact Global Alignment. In *24th International Workshop on Algorithms in Bioinformatics (WABI 2024) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 312)*, Solon P. Pissis and Wing-Kin Sung (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 17:1–17:25. https://doi.org/10.4230/LIPIcs.WABI.2024.17

[44] Ragnar Groot Koerkamp and Pesho Ivanov. 2024. Exact global alignment using a* with chaining seed heuristic and match pruning. *Bioinformatics* (2024), btae032.

[45] Yufeng Gu, Arun Subramaniyan, Tim Dunn, Alireza Khadem, Kuan-Yu Chen, Somnath Paul, Md Vasimuddin, Sanchit Misra, David Blaauw, Satish Narayanasamy, et al. 2023. GenDP: A framework of dynamic programming acceleration for genome sequencing analysis. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–15.

[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] 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.

[48] 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.

[49] Steven Henikoff and Jorja G Henikoff. 1992. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences* 89, 22 (1992), 10915–10919.

[50] D. S. Hirschberg. 1975. A linear space algorithm for computing maximal common subsequences. *Commun. ACM* 18, 6 (June 1975), 341–343. https://doi.org/10.1145/360825.360861

[51] Yifei Hu, Xiaonan Jing, Youlim Ko, and Julia Taylor Rayz. 2020. Misspelling correction with pre-trained contextual language model. In *2020 ieee 19th international conference on cognitive informatics & cognitive computing (icci* cc)*. IEEE, 144–149.

[52] Florian Huber, Marion Arnaud, Brian J Stevenson, Justine Michaux, Fabrizio Benedetti, Jonathan Thevenet, Sara Bobisse, Johanna Chiffelle, Talita Gehert, Markus Müller, et al. 2024. A comprehensive proteogenomic pipeline for neoantigen discovery to advance personalized cancer immunotherapy. *Nature biotechnology* (2024), 1–13.

[53] Miten Jain, Sergey Koren, Karen H Miga, Josh Quick, Arthur C Rand, Thomas A Sasani, John R Tyson, Andrew D Beggs, Alexander T Dilthey, Ian T Fiddes, et al. 2018. Nanopore sequencing and assembly of a human genome with ultra-long reads. *Nature biotechnology* 36, 4 (2018), 338–345.

[54] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Kathryn Tunyasuvunakool, Olaf Ronneberger, Russ Bates, Augustin Žídek, Alex Bridgland, et al. 2020. AlphaFold 2. *Fourteenth Critical Assessment of Techniques for Protein Structure Prediction* (2020).

[55] Saurabh Kalikar, Chirag Jain, Md Vasimuddin, and Sanchit Misra. 2022. Accelerating minimap2 for long-read sequencing applications on modern CPUs. *Nature Computational Science* 2, 2 (2022), 78–83.

[56] 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.

[57] Kazutaka Katoh, Kazuharu Misawa, Kei-ichi Kuma, and Takashi Miyata. 2002. MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform. *Nucleic acids research* 30, 14 (2002), 3059–3066.

[58] W James Kent. 2002. BLAT—the BLAST-like alignment tool. *Genome research* 12, 4 (2002), 656–664.

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

[60] 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.

[61] Ben Langmead, Cole Trapnell, Mihai Pop, and Steven L Salzberg. 2009. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome biology* 10 (2009), 1–10.

[62] Heng Li . 2013. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv preprint arXiv:1303.3997* (2013).

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

[64] Wen-Wei Liao, Mobin Asri, Jana Ebler, Daniel Doerr, Marina Haukness, Glenn Hickey, Shuangjia Lu, Julian K. Lucas, Jean Monlong, Haley J. Abel, Silvia Buonaiuto, Xian H. Chang, Haoyu Cheng, Justin Chu, Vincenza Colonna, Jordan M. Eizenga, Xiaowen Feng, Christian Fischer, Robert S. Fulton, Shilpa Garg, Cristian Groza, Andrea Guarracino, William T. Harvey, Simon Heumos, Kerstin Howe, Miten Jain, Tsung-Yu Lu, Charles Markello, Fergal J. Martin, Matthew W. Mitchell, Katherine M. Munson, Moses Njagi Mwaniki, Adam M. Novak, Hugh E. Olsen, Trevor Pesout, David Porubsky, Pjotr Prins, Jonas A. Sibbesen, Jouni Sirén, Chad Tomlinson, Flavia Villani, Mitchell R. Vollger, Lucinda L. Antonacci-Fulton, Gunjan Baid, Carl A. Baker, Anastasiya Belyaeva, Konstantinos Billis, Andrew Carroll, Pi-Chuan Chang, Sarah Cody, Daniel E. Cook, Robert M. Cook-Deegan, Omar E. Cornejo, Mark Diekhans, Peter Ebert, Susan Fairley, Olivier Fedrigo, Adam L. Felsenfeld, Giulio Formenti, Adam Frankish, Yan Gao, Nanibaa' A. Garrison, Carlos Garcia Giron, Richard E. Green, Leanne Haggerty, Kendra Hoekzema, Thibaut Hourlier, Hanlee P. Ji, Eimear E. Kenny, Barbara A. Koenig, Alexey Kolesnikov, Jan O. Korbel, Jennifer Kordosky, Sergey Koren, HoJoon Lee, Alexandra P. Lewis, Hugo Magalhães, Santiago Marco-Sola, Pierre Marijon, Ann McCartney, Jennifer McDaniel, Jacquelyn Mountcastle, Maria Nattestad, Sergey Nurk, Nathan D. Olson, Alice B. Popejoy, Daniela Puiu, Mikko Rautiainen, Allison A. Regier, Arang Rhie, Samuel Sacco, Ashley D. Sanders, Valerie A. Schneider, Baergen I. Schultz, Kishwar Shafin, Michael W. Smith, Heidi J. Sofia, Ahmad N. Abou Tayoun, Françoise Thibaud-Nissen, Francesca Floriana Tricomi, Justin Wagner, Brian Walenz, Jonathan M. D. Wood, Aleksey V. Zimin, Guillaume Bourque, Mark J. P. Chaisson, Paul Flicek, Adam M. Phillippy, Justin M. Zook, Evan E. Eichler, David Haussler, Ting Wang, Erich D. Jarvis, Karen H. Miga, Erik Garrison, Tobias Marschall, Ira M. Hall, Heng Li, and Benedict Paten. 2023. A draft human pangenome reference. *Nature* 617, 7960 (01 May 2023), 312–324. https://doi.org/10.1038/s41586-023-05896-x

[65] 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)*. 1–9. https://doi.org/10.1109/ASAP.2018.8445105

[66] Mao-Jan Lin, Yu-Cheng Li, and Yi-Chang Lu. 2019. Hardware accelerator design for dynamic-programming-based protein sequence alignment with affine gap tracebacks. In *2019 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. IEEE, 1–4.

[67] Joël Lindegger, Damla Senol Cali, Mohammed Alser, Juan Gómez-Luna, Nika Mansouri Ghiasi, and Onur Mutlu. 2023. Scrooge: a fast and memory-frugal genomic sequence aligner for CPUs, GPUs, and ASICs. *Bioinformatics* 39, 5 (03 2023), btad151. https://doi.org/10.1093/bioinformatics/btad151

[68] Daniel Liu and Martin Steinegger. 2023. Block Aligner: an adaptive SIMD-accelerated aligner for sequences and position-specific scoring matrices. *Bioinformatics* 39, 8 (08 2023), btad487. https://doi.org/10.1093/bioinformatics/btad487

[69] Kisaru Liyanage, Hiruna Samarakoon, Sri Parameswaran, and Hasindu Gamaarachchi. 2023. Efficient end-to-end long-read sequence mapping using minimap2-fpga integrated with hardware accelerated chaining. *Scientific Reports* 13, 1 (2023), 20174.

[70] Haiyu Mao, Mohammed Alser, Mohammad Sadrosadati, Can Firtina, Akanksha Baranwal, Damla Senol Cali, Aditya Manglik, Nour Almadhoun Alserr, and Onur Mutlu. 2022. Genpip: In-memory acceleration of genome analysis via tight integration of basecalling and read mapping. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 710–726.

[71] Santiago Marco-Sola, Jordan M Eizenga, Andrea Guarracino, Benedict Paten, Erik Garrison, and Miquel Moreto. 2023. Optimal gap-affine alignment in O (s) space. *Bioinformatics* 39, 2 (2023), btad074.

[72] 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.

[73] Eric Mays, Fred J Damerau, and Robert L Mercer. 1991. Context based spelling correction. *Information Processing & Management* 27, 5 (1991), 517–522.

[74] 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.

[75] Diego AA Morais, João VF Cavalcante, Shênia S Monteiro, Matheus AB Pasquali, and Rodrigo JS Dalmolin. 2022. Medusa: A pipeline for sensitive taxonomic classification and flexible functional annotation of metagenomic shotgun sequences. *Frontiers in Genetics* 13 (2022), 814437.

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

[77] 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.

[78] National Center for Biotechnology Information (NCBI). 2024. GenBank and WGS Statistics. https://www.ncbi.nlm.nih.gov/genbank/statistics/ Accessed: 2025-02-13.

[79] National Center for Biotechnology Information (NCBI). 2024. SRA Database Growth Statistics. https://www.ncbi.nlm.nih.gov/sra/docs/sragrowth/ Accessed: 2025-02-13.

[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] NVIDIA. [n. d.]. Boosting Dynamic Programming Performance Using NVIDIA Hopper GPU DPX Instructions. https://developer.nvidia.com/blog/boosting-dynamic-programming-performance-using-nvidia-hopper-gpu-dpx-instructions/.

[86] William R Pearson and David J Lipman. 1988. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences* 85, 8 (1988), 2444–2448.

[87] René Rahn, Stefan Budach, Pascal Costanza, Marcel Ehrhardt, Jonny Hancox, and Knut Reinert. 2018. Generic accelerated sequence alignment in SeqAn using vectorization and multi-threading. *Bioinformatics* 34, 20 (2018), 3437–3445.

[88] 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 (08 2000), 699–706. https://doi.org/10.1093/bioinformatics/16.8.699

[89] Bertil Schmidt, Felix Kallenborn, Alejandro Chacon, and Christian Hundt. 2024. CUDASW++ 4.0: ultra-fast GPU-based Smith–Waterman protein sequence database search. *BMC bioinformatics* 25, 1 (2024), 342.

[90] 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.

[91] 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.

[92] Po Jui Shih, Hassaan Saadat, Sri Parameswaran, and Hasindu Gamaarachchi. 2023. Efficient real-time selective genome sequencing on resource-constrained devices. *GigaScience* 12 (2023), giad046.

[93] 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.

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

[95] Martin Šošić and Mile Šikić. 2017. Edlib: a C/C++ library for fast, exact sequence alignment using edit distance. *Bioinformatics* 33, 9 (2017), 1394–1395.

[96] 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.

[97] Aaron Stillmaker and Bevan Baas. 2017. Scaling equations for the accurate prediction of CMOS device performance from 180 nm to 7 nm. *Integration* 58 (2017), 74–81.

[98] Hajime Suzuki and Masahiro Kasahara. 2017. Acceleration of nucleotide semi-global alignment with adaptive banded dynamic programming. *BioRxiv* (2017), 130633.

[99] Hajime Suzuki and Masahiro Kasahara. 2018. Introducing difference recurrence relations for faster semi-global alignment of long sequences. *BMC bioinformatics* 19, 1 (2018), 33–47.

[100] 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.

[101] 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.

[102] Esko Ukkonen. 1985. Algorithms for approximate string matching. *Information and Control* 64, 1 (1985), 100–118. https://doi.org/10.1016/S0019-9958(85)80046-2 International Conference on Foundations of Computation Theory.

[103] Esko Ukkonen. 1985. Finding approximate patterns in strings. *Journal of Algorithms* 6, 1 (1985), 132–137.

[104] U.S. Food and Drug Administration. [n. d.]. PrecisionFDA Truth Challenge V2. https://precision.fda.gov/challenges/10.

[105] Robert Vaser, Ivan Sović, Niranjan Nagarajan, and Mile Šikić. 2017. Fast and accurate de novo genome assembly from long uncorrected reads. *Genome research* 27, 5 (2017), 737–746.

[106] Sumit Walia, Cheng Ye, Arkid Bera, Dhruvi Lodhavia, and Yatish Turakhia. 2024. TALCO: tiling genome sequence alignment using convergence of traceback pointers. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 91–107.

[107] 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.

[108] A. Wozniak. 1997. Using video-oriented instructions to speed up sequence comparison. *Bioinformatics* 13, 2 (04 1997), 145–150. https://doi.org/10.1093/bioinformatics/13.2.145

[109] 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.

[110] Alberto Zeni, Giulia Guidi, Marquita Ellis, Nan Ding, Marco D. Santambrogio, Steven Hofmeyr, Aydın Buluç, Leonid Oliker, and Katherine Yelick. 2020. LOGAN: High-Performance GPU-Based X-Drop Long-Read Alignment. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 462–471. https://doi.org/10.1109/IPDPS47924.2020.00055