Sequence alignment acceleration using an FPGA MPSoC

Eric Santigosa

February 9, 2026

Abstract–
Sequence alignment is a critical computational bottleneck in modern genomics, driving the need for specialized hardware acceleration. Architectures like SMX demonstrate significant potential in simulation, but bridging the gap between theoretical logic design and physical deployment remains a major challenge. ASIC manufacturing is prohibitively expensive, so silicon implementations of experimental designs are a rare occurrence. FPGA prototyping offers a promising alternative, but it often relies on non-coherent interfaces and bare-metal deployments that obscure true system-level performance metrics.
In this work, we present the optimization and full-stack integration of the SMX sequence alignment accelerator into the FPGA of a heterogeneous MPSoC. We develop a custom RTL memory bridge that adapts the accelerator's internal logic to a processor's last-level cache coherent port, enabling tightly coupled, zero-copy data sharing between the FPGA and the host CPU. This hardware is orchestrated by a custom software stack built on the Xilinx Runtime (XRT), exposing the accelerator to a standard Linux OS as a seamless library function. Our integrated SMX system achieves up to 92× speedup over SIMD-optimized software.
Keywords- Performance engineering, Sequence alignment, Hardware acceleration, Field Programmable Gate Array (FPGA).

1 INTRODUCTION

SEQUENCE alignment is a fundamental method in bioinformatics used to compare biological sequences (DNA, RNA, or proteins) to identify regions of similarity indicating functional, structural, or evolutionary relationships. By quantifying conservation patterns and mismatches, alignment provides critical insights into gene function and evolutionary divergence [1]. Its importance spans vast areas of research, including phylogenetics [2, 3], drug design [4, 5], and personalized medicine [6, 7], making it a cornerstone computational task in modern life sciences.

Classical approaches are dominated by Dynamic Programming (DP) algorithms like Needleman-Wunsch [8] and Smith-Waterman [9]. While these guarantee optimality, their quadratic time and space complexity ($O(nm)$) make them prohibitively slow for large-scale genomics. To address this, strategies have diverged into heuristics like FASTA [10] and BLAST [11], which trade optimality for speed, and modern exact algorithms like WFA [12], A*PA [13] and QuickEd [14], which aim for sub-quadratic performance by reducing the search space.

Despite algorithmic improvements, the computational cost per DP-element often remains high, reinforcing the need for hardware acceleration [15]. This acceleration spectrum ranges from a generalist approach to more targeted solutions: general-purpose CPUs leverage SIMD extensions for vectorization [16], while GPUs utilize massive core counts for data parallelism [17]. At the far end, Domain Specific Accelerators (DSAs) on FPGAs or ASICs offer maximum efficiency by exploiting custom, hardware-level parallelism specific to the alignment logic [18, 19].

Recent work introduced SMX [20], a heterogeneous architecture designed to counter the over-specialization of typical State-Of-The-Art (SOTA) accelerators. While many accelerators excel only at regular workloads, they struggle with practical, irregular use cases. SMX addresses this by offering a flexible, scalable architecture that balances high computational efficiency with the adaptability required for diverse genomic applications.

To fully exploit the benefits of the SMX architecture, the target implementation is typically an ASIC. As the standard for deployed hardware accelerators, ASICs provide advantages in raw performance, energy efficiency and physical form factor by eliminating the instruction overhead of general-purpose processors and the static power consumption of programmable logic.

While ASICs offer peak performance, their high cost and slow manufacturing make them suboptimal for iterative research. FPGAs, conversely, provide a compelling platform for prototyping. By mapping designs to reconfigurable

• E-mail de contacte: Eric.Santigosa@autonoma.cat
• Menció realitzada: Enginyeria de Computadors
• Treball tutoritzat per: Max Doblas Font (BSC) i Juan Carlos Moure Lopez (DACSO)
• Curs 2025/26

logic, FPGAs enable execution at near-hardware speeds, allowing developers to validate designs, identify bottlenecks, and perform essential software-hardware co-design with no fabrication cost, prior to silicon implementation [21].

However, standard FPGA prototypes often use non-coherent interfaces in bare-metal environments, ignoring critical OS-level factors such as cache coherence and scheduling [22]. This work bridges that gap by integrating SMX as a fully coherent accelerator within a Linux-based Zynq MPSoC. We present a system-level evaluation against a state-of-the-art software aligner and quantify the benefits of tightly coupled hardware acceleration in a realistic operating environment.

2 Objectives and Contributions

This paper presents the first system-level implementation and evaluation of the SMX sequence alignment accelerator, exploring its real-world performance and integration challenges. We implement and extend SMX on a Zynq MPSoC, integrating it as a tightly-coupled, cache-coherent accelerator managed via Xilinx Runtime (XRT) under Ubuntu Linux. We quantify the performance of our implementation and provide a generalizable methodology for integrating complex, state-of-the-art accelerators into heterogeneous systems.

In the following, we summarize the key contributions of this work.

- We present an optimized, efficient RTL integration of SMX within the Zynq MPSoC programmable logic, leveraging the Accelerator Coherency Port (ACP) for low-latency memory access.
- We deliver a complete environment comprising FPGA synthesis scripts and an XRT-based host library to instantiate, configure, and control the SMX accelerator from user-space applications.
- We provide an SMX-accelerated library for the MPSoC that aligns DNA, Amino Acid, and ASCII sequences, supporting diverse cost models such as edit distance, linear gap penalties, and substitution matrices.
- We conduct a detailed performance analysis of SMX at the system level, comparing it with a state-of-the-art software implementation. Additionally, we provide an energy efficiency comparison of the SMX solution against a high-performance x86 CPU running the software baseline.

3 Methodology

This thesis adopted a Spiral Iterative Model as its core research and development framework. This approach was selected because of the complex and unpredictable nature of the project. The limitations, challenges and requirements were not fully known at the outset and were expected to evolve due to the novelty of the integration method. The spiral model provided the necessary flexibility, allowed for continuous risk management, and facilitated a process of progressive refinement based on emergent findings.

Each ”spiral,” or iteration, was split into four distinct phases: 1) Identifying objectives and constraints for the next loop. 2) Analysing potential issues. 3) Implementing the hardware and software for the current iteration. 4) Deploying the build onto the MPSoC, running workloads, and evaluating the results to inform the next spiral.

This four-phase loop allowed us to pivot to new objectives or refine existing ones when we faced unexpected issues, systematically navigating from a minimal example to a fully working system.

4 State of the Art

The challenge of accelerating sequence alignment has been tackled from software and hardware performance optimizations, with the goal of performing the $O(nm)$ computation faster. In the following, we enumerate several common approaches. State-of-the-art solutions, such as the KSW2 library used as the core alignment kernel of Minimap2 [23], often use a combination of both hardware and software strategies.

4.1 Software Optimizations

The optimization of sequence alignment software on general-purpose processors tends to follow a tiered approach that first addresses memory latency and data density, and then maximizes instruction throughput. The foundational technique for processing large sequences is cache blocking (or tiling), which restructures the iteration order of the dynamic programming matrix into small sub-blocks. This ensures that the active working set resides permanently in the fast L1 cache, overcoming the ”memory wall” [24]. To further optimize memory bandwidth, differential encoding is employed to store only the small score differences between adjacent cells (deltas) rather than absolute values. This compression enables the use of narrower data types (e.g., 8-bit or 2-bit), drastically increasing the effective data capacity of the CPU cache.

Once memory access is optimized, bit-parallelism (SWAR) serves as the primary arithmetic optimization, exploiting these compressed differential states to pack and update multiple alignment cells simultaneously within standard 64-bit integer registers [25]. Furthermore, strictly scalar implementations maximize pipeline utilization through branchless logic and linear space buffering; the former uses bitwise predication to replace costly conditional jumps, while the latter restricts memory allocation to the immediate wavefront, thereby preventing operating system paging overhead [26, 16].

4.2 Hardware Optimizations

Hardware acceleration parallelizes the DP matrix calculation, leveraging a spectrum of hardware from general-purpose processors to fully custom accelerators. On general-purpose CPUs, SIMD extensions (e.g., SSE4, AVX2, NEON) are the most common form of acceleration. The seminal work by Farrar [16] introduced a ”striped” or ”intra-sequence” parallelization, where the DP calculation for a single query sequence is vectorized. This approach is

now the foundation for SOTA software libraries like Parasail [27], the SSW Library [28], and the KSW2 library from Minimap2 [23], which are often the baseline for performance comparisons.

Moving to more specialized hardware, Graphics Processing Units (GPUs) offer massive data-parallelism with thousands of cores. Implementations such as CUDASW [29] parallelize the computation of a single DP matrix using the anti-diagonal "wavefront" pattern. While powerful, GPU-based solutions can be limited by data transfer (PCIe) bottlenecks and irregular memory access patterns.

Finally, Domain-Specific Accelerators (DSAs) implemented on FPGAs and ASICs offer the highest performance and power efficiency. FPGAs are particularly well-suited for streaming DP computations, and the classic implementation involves a systolic array [30], where a chain of Processing Elements (PEs) maps directly to the dataflow of the DP matrix.

5 Background

5.1 Sequence Alignment

Sequence alignment seeks to determine the similarities and differences between two given sequences. These differences may denote genetic variations, mutations, evolutionary events, and sequencing errors. More formally, given two input sequences, a reference $R = r_0 r_1 ... r_{m-1}$ and a query $Q = q_0 q_1 ... q_{n-1}$, and a scoring function, the optimal alignment is defined as the sequence of operations (i.e., match, mismatch, insertion, and deletion) that transforms one sequence into the other, maximizing the score function. Sequence alignment is a fundamental operation for genome analysis, pattern recognition, text processing, etc.

This process is typically computed using dynamic programming (DP) and involves two steps: (1) the DP-matrix computation and (2) the alignment traceback.

$$M_{i,j} = \max \begin{cases} M_{i-1,j-1} + \text{score}(r_i, q_j) & \text{(Mismatch)} \\ M_{i-1,j} + I & \text{(Insertion)} \\ M_{i,j-1} + D & \text{(Deletion)} \end{cases} \quad (1)$$

First, the $m \times n$ DP-matrix (Figure 1.a) is computed using Equation 1. In particular, each cell $M_{i,j}$ in the matrix stores the optimal alignment score for the subsequences $R[0..i]$ and $Q[0..j]$. The value of each cell is calculated using a recurrence relation based on its neighbours and the substitution costs, often represented as a matrix (Figure 1.d).

Once the DP-matrix is filled, the optimal alignment score is found in the final cell, $M_{m,n}$. To reconstruct the path that produced this score, the second step, traceback, is performed. This process starts at $M_{m,n}$ and traces backward to the origin $M_{0,0}$, following the path of predecessor cells that contributed the maximum score at each step (Figure 1.b). The resulting path represents the optimal alignment (often encoded as a CIGAR string, Figure 1.c).

The DP-matrix computation is the primary performance bottleneck, with a quadratic $O(mn)$ time and space complexity. However, its computation and memory access patterns are highly regular. Moreover, all cells along a given anti-diagonal are data-independent and can be computed in parallel. Because of this property, the DP-matrix computation is the main target of hardware acceleration. The traceback step, in contrast, is inherently sequential. The decision at each cell (i.e., whether the optimal score came from a match, insertion, or deletion) depends on the result of the previous cell in the path. While its time complexity is only linear, $O(m + n)$, this step is not amenable to the same level of parallelization as the DP-matrix and is commonly executed by a general-purpose CPU.

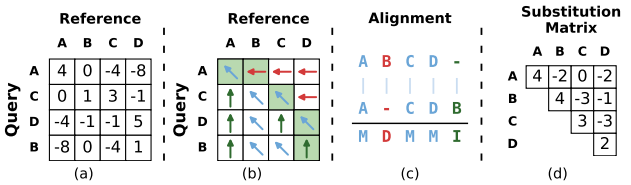

Figure 1: (a) Filled DP-Matrix. (b) Traceback matrix. (c) Final alignment result. (d) Substitution Matrix.

The sequence alignment problem has many variation for different use cases. For example, the *edit model* is commonly used for applications such as comparing ASCII code/text strings or analysing genome DNA/RNA sequences. The edit model assigns a cost of 1 to insertions, deletions and mismatches, and so counts the "number of changes". Other weighted *gap models* employ different values for insertions, deletions, and substitutions 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 to estimate the likelihood of mutations between amino acids in protein alignments, helping to compute biologically significant alignments.

5.2 Field-Programmable Gate Array

A Field-Programmable Gate Array (FPGA) is an integrated circuit that can be reconfigured by a user after manufacturing. Unlike a Central Processing Unit (CPU) that executes a stream of software instructions, or an Application-Specific Integrated Circuit (ASIC) that has its logic permanently fabricated, an FPGA's hardware can be re-programmed in the field to implement any custom digital circuit.

The reconfigurability of an FPGA allows for the implementation of massively parallel Domain-Specific Accelerators (DSAs). By creating a custom digital circuit tailored to a specific problem, developers can achieve orders-of-magnitude higher performance and power efficiency compared to a general-purpose CPU. For a task like sequence alignment, a dataflow architecture, such as a systolic array, can be "unrolled" onto the FPGA fabric, processing data in a deep pipeline that a CPU cannot replicate.

At their core, FPGAs are composed of a large, two-dimensional array of reconfigurable logic blocks, called Configurable Logic Blocks (CLBs). These blocks contain Look-Up Tables (LUTs) and Flip-Flops (FFs). LUTs are small, reconfigurable SRAM-based truth tables that implement any custom combinatorial logic function (e.g., AND, OR, XOR, or complex combinations), while FFs are storage elements that register the output of the LUTs, allowing for the creation of synchronous, pipelined circuits. These

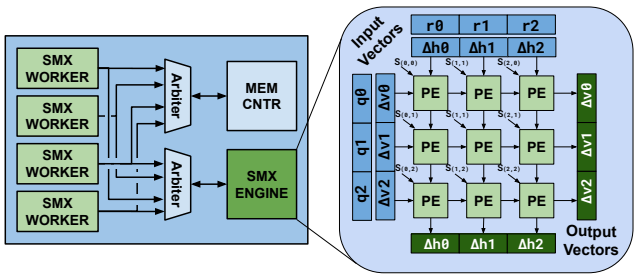

Figure 2: SMX Architecture Diagram. This is the optimal configuration with 4 SMX-Workers per SMX-Engine.

CLBs are surrounded by a flexible routing interconnect that can be programmed to connect any CLB to any other, creating complex data paths. Modern FPGAs also include specialized, hardened blocks such as Block RAMs (BRAMs) for on-chip memory and DSP slices for high-performance arithmetic, which are essential for accelerator design.

Historically, FPGAs were integrated as discrete accelerators connected to a host CPU via external interfaces like PCI Express (PCIe). While effective for massive batched workloads, this discrete model suffers from significant data movement overhead, often negating acceleration gains for fine-grained or latency-sensitive tasks. To overcome this bottleneck, some modern architectures have shifted towards Heterogeneous System-on-Chip (SoC) designs. In this integrated paradigm, the FPGA fabric acts not as a peripheral, but as a tightly coupled peer to the processor. By eliminating external buses, these systems allow for high-bandwidth, low-latency communication and shared memory access, enabling the CPU to offload compute-intensive kernels to the hardware logic while maintaining the flexibility of a standard Operating System for high-level control.

This work leverages a Multi-Processor System-on-Chip (MPSoC), such as the Zynq UltraScale+ device on the KR260 board [31]. This device integrates a high-performance, multi-core ARM CPU subsystem (the Processing System, or PS) with an FPGA (the Programmable Logic, or PL) on the same silicon die. This single-chip solution is the key enabler for our research, as it enables us to prototype a tightly coupled accelerator that shares memory and cache coherence directly with CPUs running a full Linux OS.

6 SMX Acceleration Architecture

SMX [20] is a heterogeneous architecture for sequence alignment acceleration. It combines: (1) SMX-1D, an ISA extension to accelerate irregular and sequential alignment tasks, like traceback; and (2) SMX-2D, a highly optimized coprocessor that accelerates the regular and highly-parallel task of computing the DP-matrix. In this work, because the inclusion of an ISA extension into the physical CPUs is infeasible since their logic is imprinted in non-programmable silicon, we only perform the integration of the SMX-2D accelerator.

SMX-2D (referred to in this work as the SMX accelerator or SMX) is designed as a coprocessor connected to the CPU's Last Level Cache (LLC). The design contains a systolic matrix of SMX Processing Elements (SMX-PE). Each one of these SMX-PEs is responsible for the computation of a single DP-matrix element (DP-element). The CPU can offload the computation of arbitrarily large DP-matrix blocks (DP-blocks) to the accelerator. SMX has Control State Registers (CSR) that the CPU can use to control the accelerator's behaviour. These registers have distinct purposes, such as controlling the execution state, configuring the execution mode, or providing the addresses of the necessary input and output memory structures.

From a high-level host-side view, the minimum configuration the accelerator needs to be provided is a query, a reference, the sequence and element sizes and the scoring scheme. When the SMX accelerator finishes computing the alignment, it will store the resulting calculated DP-matrix in the provided output structures. The accelerator can also be configured to work in score-only mode, when only the final alignment score is needed.

Because storing every element of a massive DP matrix would saturate memory bandwidth, SMX divides the DP-block into smaller tiles (DP-tiles) and saves only their "border" elements (the rightmost column and bottom row) to memory, enabling efficient computation of subsequent tiles and allowing on-demand recomputation of inner DP-elements during traceback. Therefore, SMX can reduce the memory footprint up to 256× compared to the software implementation.

One of the main highlights of SMX is its versatility. The design can handle and adapt to different workloads to extract as much performance as possible in every case. One of the ways it accomplishes this is by adjusting the amount of bits used to encode each DP-element, based on the scoring scheme and element type. Both these variables are configurable, with the scoring scheme allowing either match-mismatch (unit or weighed costs, requiring 1 to $\log_2(\max(cost_{\text{ins}}, cost_{\text{del}}, cost_{\text{mis}}))$ bits per element) or substitution matrix (using matrices like BLOSUM and PAM, which contain 26×26 penalty values ranging from -6 to 15, requiring 6 bits per element) and the element type allowing one of RNA (2 bits), DNA (4 bits), Protein (6 bits) or ASCII (8 bits).

The maximum size associated with the scoring scheme and the element type can be used as the Element Width (EW) for our DP-elements, allowing the hardware to pack $2\times$ to $16\times$ more elements into a single operation than using conventional 32-bit integers. This is possible thanks to the use of differential encoding, which means each cell records the difference relative to its neighbours (Δv and Δh) rather than the absolute score.

6.1 SMX-Engine

The SMX-Engine is composed of a 2D matrix of Processing Elements (SMX-PEs). The engine contains four distinct physical arrays to support the configurable bit-widths (e.g., a 32×32 array for 2-bit operations down to an 8×8 array for 8-bit operations). This configuration determines the Vector Length (`VL`), which represents the number of DP-elements processed in parallel along one dimension. Data flows in a wavefront pattern: inputs $\Delta v'$ and $\Delta h'$ enter from the left and top edges, respectively, and propagate through the grid, allowing the engine to output a computed tile of size `VL`×`VL` every clock cycle.

Due to propagation delays across the large 2D mesh, the array incorporates segmentation registers along the antidiagonals to maintain high frequency. Inside the engine, substitution scores (S') are generated locally; for DNA, this is done via a broadcast mesh of comparators (Match/Mismatch), and for Protein sequences, the engine utilizes a specialized register-based structure to access substitution matrix rows in parallel, bypassing the port limitations of standard SRAMs.

6.2 SMX-Workers

To prevent the high-throughput engine from stalling due to memory latency, the accelerator employs "SMX-Workers." A Worker acts as an orchestrator for a specific DP-block; it handles the fetching of sequence chunks and boundary data from the L2 cache, issues tile tasks to the engine, and writes back the results. The Worker minimizes L2 cache accesses by processing groups of tiles that share the same reference and query cache lines, grouped in "Supertiles." It loads the necessary data once and processes all inner tiles locally using internal SRAM, only writing back to the L2 cache when the supertile computation is complete.

Since the wavefront computation of a matrix has start-up and wind-down phases (where not all PEs are active), a single Worker cannot achieve 100% engine utilization. SMX implements multiple Workers that interleave their requests to the single SMX-Engine. While one Worker waits for memory or resolves a dependency, another can utilize the engine, ensuring maximum computational throughput.

7 SMX MPSoC Implementation

This section details the full-stack implementation of the SMX accelerator on the Xilinx Zynq UltraScale+ MPSoC, as depicted in Figure 3. While Section 6 described the original architecture of the SMX core, this chapter focuses on the solving the challenges of deploying that core as a tightly coupled, OS-managed resource. The integration is achieved through a hardware-software co-design approach, illustrated in Figure 4, which bridges the gap between the accelerator's internal logic and the host operating system. This requires coordinating two distinct domains: the Programmable Logic (PL), which hosts the custom accelerator and protocol adaptation logic, and the Processing System (PS), which executes the user application and manages system resources.

We begin by characterizing the hardware platform and the specific protocol constraints that influenced our design. Subsequently, we detail the microarchitecture of the custom hardware modules, the Control Interface and Memory Bridge. After that, we describe the workflow used to deploy the accelerator. Finally, we describe the accompanying software stack and explain how the driver orchestrates memory mapping and buffer allocation to enable transparent, low-latency acceleration.

7.1 Platform Configuration

The implementation targets the Xilinx Kria KR260 Robotics Starter Kit (Figure 3 c), which is powered by a Zynq UltraScale+ MPSoC. This heterogeneous device partitions its resources into two tightly integrated subsystems. The Processing System (PS) features a quad-core ARM Cortex-A53 Application Processing Unit (APU) running at 1.3 GHz, which we utilize to manage the operating system and execute the non-parallelizable traceback step. Conversely, the Programmable Logic (PL) provides an FPGA fabric with approximately 256k logic cells that hosts the SMX accelerator. While the PL operates at 100 MHz, which is significantly slower than the CPU's frequency, it compensates by exploiting massive spatial parallelism to execute the Dynamic Programming recurrence.

The configuration of this hardware platform can be visualized in Figure 3 b. Communication between the PL and PS subsystems follows the AMBA AXI4 protocol. The PL exposes a 32-bit AXI Lite slave port which our design will implement within its control interface. It will be accessible from the PS' High-Performance (HP) Master. A critical design decision in this work was selecting the interconnect strategy from the PL to memory. We avoid HP ports, which require software to explicitly flush CPU caches to main memory to ensure data visibility. Our design instead utilizes the Accelerator Coherency Port (ACP). The ACP enables the PL to snoop the APU's L2 cache directly, ensuring that the accelerator consumes data immediately after the CPU writes it. This allows for a "zero-copy" data sharing model that significantly reduces the latency of fine-grained alignment tasks, albeit at the cost of stricter transaction constraints (such as fixed burst lengths) that our hardware must actively manage.

7.2 Hardware Microarchitecture and Optimizations

For our SMX system-level deployment, we developed a set of custom hardware modules and microarchitectural extensions (depicted in Figure 3 A) designed to resolve interface mismatches and offload control overhead from the CPU. By further refining the baseline RTL, we ensured the system operates at peak performance.

The interaction with the host processor is mediated by a custom **Control Interface**, which maps the accelerator's configuration registers to the AXI-Lite bus. To facilitate precise performance analysis in this heterogeneous environment, we extended the hardware design with a suite of performance counters. These counters log critical runtime events, such as stall cycles caused by memory starvation versus engine dependencies, providing visibility into system-level bottlenecks that is unavailable in standard black-box IP. The counter logic is distinct from the operational path, ensuring that profiling does not impact alignment throughput.

A specific challenge in the MPSoC integration is the bitwidth mismatch between the accelerator's internal logic and the coherency port. The SMX-Worker is designed to request and write 512-bit DP-Supertile deltas (one cache line) per request, whereas the ACP is strictly limited to a 128-bit data path. To address this, we implemented a custom RTL **Memory Bridge** that acts as a width adapter. It maps each 512-bit logical request from the Worker into exactly one 4-beat ACP burst transaction (4×128 bits), ensuring optimal alignment with the Zynq UltraScale+ cache architecture.

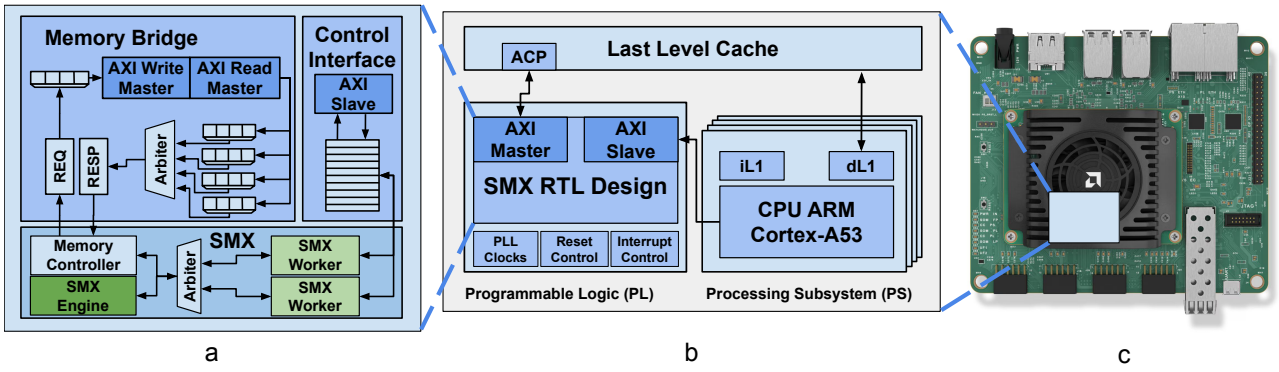

Figure 3: Implementation Overview. (a) SMX Implementation Design (b) Platform Configuration (c) Kria KR260 board

While the accelerator does not require maximizing the theoretical bandwidth of the interconnect, it is highly sensitive to memory access latency. To mitigate this, the Memory Bridge logic is non-blocking, managing up to 16 outstanding read transactions. This allows the accelerator to issue multiple cache line requests speculatively, effectively overlapping the ACP access latency with the computation of current tiles. The bridge is able to handle the redistribution of responses for up to 4 distinct workers by utilizing AXI IDs.

We leveraged this direct memory access to optimize the substitution matrix loading mechanism. In the original design, configuring the scoring matrix required the CPU to perform up to 75 register writes. Our optimized implementation allows the CPU to simply provide a base address, triggering the Memory Bridge to fetch the entire matrix via the ACP, significantly reducing configuration setup time.

Finally, to optimize the score-only alignment mode, we introduced a specialized hardware reduction unit within the SMX-Worker. Previously, obtaining the final score required the CPU to fetch and sum the boundary elements of the last DP-tile. Our implementation integrates a hardware adder that accumulates these values in real-time. This creates a ”fire-and-forget” model where the final alignment score is exposed via a single register read, eliminating the post-processing overhead on the host CPU.

7.3 Hardware Deployment Workflow

Unlike traditional FPGA flows that produce a static bitstream, the MPSoC integration requires a unified hardware-software compilation pipeline. Figure 4 illustrates this co-design flow. The process begins with the hardware synthesis (left branch), where the SMX RTL is packaged into a Vitis kernel (.xo). Simultaneously, the Zynq UltraScale+ hardware definition containing the required clock, interrupt and reset blocks is exported from Vivado as an extensible platform (.xsa). The Vitis linker then integrates these intermediate artifacts, resolving the connectivity between the SMX kernel ports and the platform's exported AXI interfaces.

This linking stage produces a fixed platform (.xsa), which Vitis uses for the packaging step to generate the final binary container (.xclbin). The binary container encapsulates the FPGA bitstream and the metadata required by the runtime. In parallel, the software application (right branch) is cross-compiled for the ARM Cortex-A53 architecture,

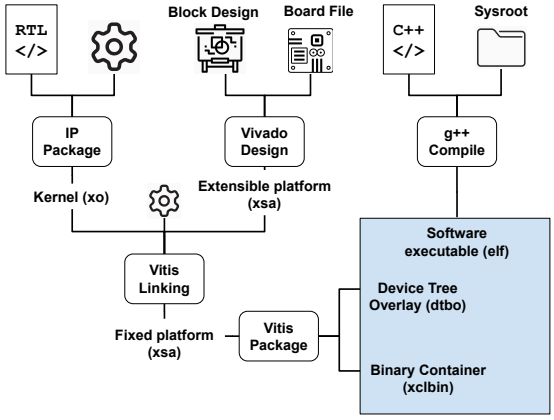

Figure 4: Vitis Integrated Workflow

linking against the KR260 sysroot to produce the host executable (.elf). To enable runtime reconfiguration, the build process also generates a Device Tree Overlay (.dtbo). This overlay is dynamically loaded by the OS to insert the accelerator into the kernel's device tree, allowing the driver to probe and configure the SMX hardware without requiring a system reboot.

7.4 Software Integration and Driver Stack

To expose the SMX accelerator to user-space applications, we have developed a C++ library. Our library is based on the Xilinx Runtime (XRT) software stack. This layered architecture abstracts the low-level hardware interactions, allowing the host application to manage the accelerator through high-level APIs rather than direct register manipulation.

The software stack consists of three primary layers:

User Application Layer: The top-level C++ application utilizes a custom SMX API to manage alignment tasks. This layer handles the preprocessing and traceback of sequence alignments and the allocation of host-aligned memory buffers via the Contiguous Memory Allocator (CMA). Unlike standard non-coherent DMA configurations that require uncached mappings, the utilization of the ACP allows these CMA buffers to be mapped with *Write-Back Cacheable* attributes. This ensures the CPU can operate at full L2 speed without explicit cache flushing, while the

FPGA snoops the latest data directly through the coherency port.

XRT Runtime: The application communicates with the hardware via the XRT Native API. The XRT Shim layer translates these API calls into ioctl commands. It manages the lifecycle of the hardware kernel, handling the loading of the `.xclbin` binary and the mapping of Buffer Objects (BO) into the device's virtual address space.

Kernel Driver (ZOCL): At the lowest level, the Zynq OpenCL (ZOCL) driver manages the physical interface. It handles interrupt servicing and configures the MPSoC's Memory Management Unit (MMU) to ensure the FPGA fabric can access the physical pages assigned to the user application.

8 EVALUATION

In this section, we evaluate the performance, area, and energy efficiency of our SMX accelerator. First, we present throughput results for various sequence lengths and data types, comparing SMX against the state-of-the-art sequence alignment tool. Then, we analyse the execution time distribution to identify bottlenecks in the SMX-accelerated system. Also, we explore the architectural design choices, particularly the multi-worker configuration, and their impact on performance. Finally, we report the area utilization and operating frequency of the SMX design on the FPGA platform along with a discussion of the energy efficiency compared to different CPU architectures.

8.1 Experimental Setup

Hardware Setup: In this work, we deploy our SMX accelerator on a Xilinx Kria KR260 Development Board, which features a Zynq UltraScale+ MPSoC with a 4-core ARM Cortex-A53 CPU running at 1.3GHz and an FPGA fabric with 14,640 Configuration Logic Blocks (CLBs). The software baselines for throughput comparison are executed on a single core of the ARM Cortex-A53 CPU. For energy efficiency comparisons, we also execute the software baseline on a single core of the AMD Ryzen AI 9 HX 370 processor.

Software Setup: The system runs the official Xilinx Ubuntu 24.04 LTS image. For performance and energy comparisons, we select the state-of-the-art SIMD-optimized KSW2 software as the baseline. KSW2 [23] is a widely used library that implements an antidiagonal vectorization strategy. The original KSW2 implementation includes support for x86 SSE vectorization. For ARM experiments, we adapt the code to leverage the ARM NEON vector extension. All software baselines and SMX software are compiled with `-O3` optimizations using g++ 13.3.0 on both ARM and x86 architectures.

MPSoC Toolchain: The SMX accelerator is packaged as an RTL kernel using Vivado 2024.2 and linked to the Vivado extensible platform using Vitis 2024.2. The software running on the ARM CPU is compiled using g++ with the xilinx-zynqmp-common 2024.2 sysroot and the XRT C++ libraries. The area and frequency results for the SMX accelerator are obtained from the post-implementation reports generated by Vivado during the Vitis linking process.

Experimental Datasets: Using the methodology from [12, 14], we generate distinct datasets containing sequences of varying lengths, ranging from 100 to 10K base pairs (bp). These cover typical use cases for standard NGS short reads (e.g., Illumina) as well as Third-Generation long reads (e.g., PacBio HiFi). Additionally, we use a real PacBio HiFi dataset containing 512 reads ranging from 6.6K to 16.3K bp to measure energy consumption. To cover all use cases and demonstrate SMX's versatility, we align sequences composed of DNA (with and without gap penalties), proteins (both with gap costs and BLOSUM matrices), and ASCII characters using edit distance.

Power Evaluation: To report the power consumption of the different systems, we read the processor's current power draw from its registers every 0.1 seconds. On the Kria KR260 board, we read the specific `hwmon2/power1_input` performance counter, which represents the entire board's power consumption. For the ARM software execution measurements, the entire Programmable Logic (PL) is turned off to prevent passive FPGA power consumption. For the x86 core, we obtain the CPU package's power consumption using the `turbostat` Linux tool.

8.2 SMX Throughput Results

Figure 5 presents the throughput achieved by our SMX implementation and the software baseline. As shown in the figure, SMX achieves a speed-up ranging from from 0.6× to 5× for sequences of length 100 bp and 4× to 92× for sequences of length 10000 bp over KSW. The speed-up is particularly higher for longer sequences, because of the architectural advantage of the accelerator when facing the quadratic complexity of sequence alignment.

If we focus on SMX accelerated versions, we can identify clear trends. In the first place, throughput of computed elements goes up when we increase sequence length, because of the accelerator's pipeline being fully saturated, which amortizes the initial data setup and transfer overhead. For very short sequences, added CPU latencies per alignment outweigh the benefits obtained from massive parallelization.

Figure 5 shows a drop in alignments/s when moving from DNA (2-bit/4-bit) to Protein (6-bit) or ASCII (8-bit). This performance variation is architectural. As detailed in Section 6.1, the SMX-Engine reconfigures its physical array based on element width. DNA-edit alignments (2 bits) leverage a larger Vector Length of 32 (VL), processing up to 1024 elements per cycle. In contrast, ASCII alignments (8 bits) reduce the VL to 8, naturally lowering the maximum theoretical throughput.

Protein alignment is a special case, since it is often computationally expensive in software due to memory accesses to the substitution matrix (BLOSUM/PAM). SMX has a dedicated hardware structure for this purpose, so it maintains high performance even for Protein alignments. Unlike software, the SMX-Engine utilizes a specialized register-based structure to access substitution scores in parallel, eliminating the memory bottleneck associated with weighted scoring.

Furthermore, we must distinguish between the alignments with and without traceback. When performing score-only (no traceback) alignments, SMX memory writes are reduced by 16×, which decreases the execution time. Fur-

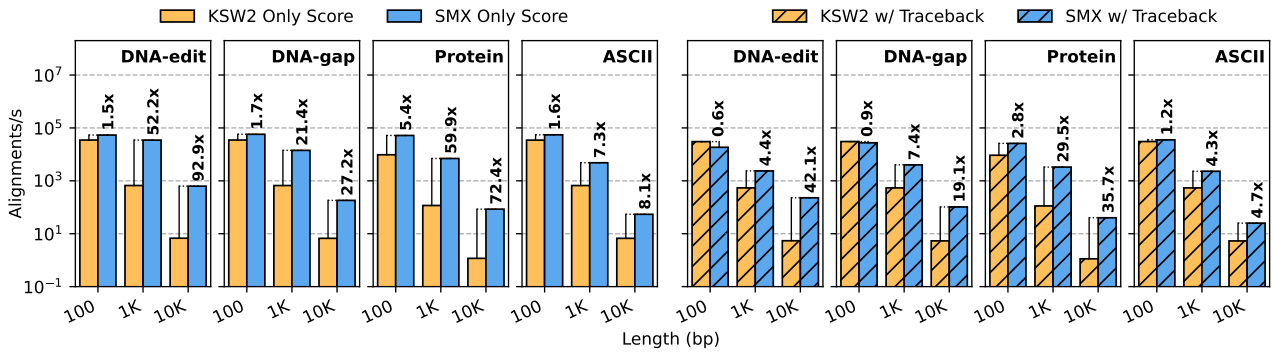

Figure 5: Throughput of alignments per second in logarithmic scale for each element size subplot using synthetic datasets of lengths 100, 1000 and 10000 base pairs (bp). The numbers above the bars are the speed-ups of SMX with two Workers in reference to the software ARM KSW2 version.

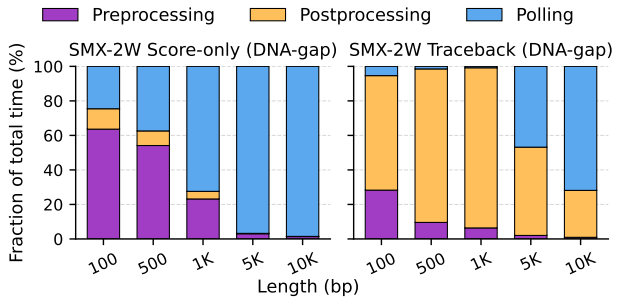

Figure 6: Distribution of execution time measured by the CPU for traceback and score-only modes for alignment sizes ranging from 100 to 10K.

thermore, the traceback step introduces a significant overhead to the CPU. This is evidenced as a approximate 2× speed-up of score-only executions against their traceback counterpart. A critical observation is the disparity in operating frequencies. The software baseline benefits from the ARM CPU's 1.3 GHz clock speed, whereas our SMX accelerator operates at only 100 MHz. Despite this 13× clock frequency disadvantage, SMX generally demonstrates superior performance. This highlights the architectural efficiency of the SMX design: by exploiting the fine-grained parallelism of the DP-matrix computation via its Engine, it performs orders of magnitude more useful work per clock cycle than the CPU's SIMD units.

8.3 Execution Time Distribution

In Figure 6, Preprocessing (purple) refers to the time needed to pack input data and configure the Workers, while Postprocessing (orange) corresponds to the computation the last tile and the addition of its score to the one provided by the counter (score-only) or to the recomputation of all tiles in the traceback path (traceback). Polling (blue) is the time the CPU is idle waiting for the accelerator to compute the DP-matrix.

If we look at the time distribution of score-only executions (left half of Figure 6), the system is latency-bound for short sequence lengths. The Preprocessing overhead outweighs the execution time of the accelerator. This indicates that for extremely short reads the performance ceiling is set not by the accelerator's compute capability, but by the CPU's overhead in managing the offload. Even with Zero-Copy (ACP), the software still has to pack the data into the buffer before the hardware sees it. From this, we can conclude the use of SMX for score-only alignments starts making the most sense when aligning sequences of length 1Kbp or larger.

Figure 6 (right) shows the Postprocessing bar dominating execution time. As the SMX accelerator reduces the matrix filling time to near-zero, the non-accelerated traceback phase becomes the dominant system bottleneck. This confirms that while SMX solves the quadratic ($O(nm)$) problem, the linear ($O(m+n)$) traceback remains a CPU-bound constraint. This added step is the reason why on Figure 5 the versions that perform traceback exhibit lower gains relative to the baseline than the ones that don't. When performing traceback, the performance benefits outweigh the added overhead when aligning sequences of length 5Kbp or more.

8.4 SMX Utilization Analysis

The maximum number of DP-Elements one Engine is able to compute each cycle is 1024, 256, 100 and 64 for element sizes 2, 4, 6 and 8. Nonetheless, because of memory fetches and dependencies between DP-Tiles, the Engine is left idle for periodic portions of time. This characteristic allows us to increase performance without having to replicate the whole architecture, which would be impossible in this platform due to resource constraints. By adding a second Worker per Engine, we could ideally double the performance while only increasing the area corresponding to the additional Worker design, without having to instantiate another entire accelerator.

Figure 7 shows SMX with two Workers (blue dots) reaching up to 85% peak GCUPS, while SMX-1W's maximum is lower (slightly more than half of SMX-2W's). The jump in GCUPS between the 1-Worker and 2-Worker configurations validates the multi-worker architecture described in Section 6.2. By interleaving two Workers, the system effectively hides these latencies, keeping the systolic array active and pushing utilization closer to the theoretical maximum for long sequences.

We also observe a stagnation of GCUPS increase in proportionality to sequence length at certain points, depend-

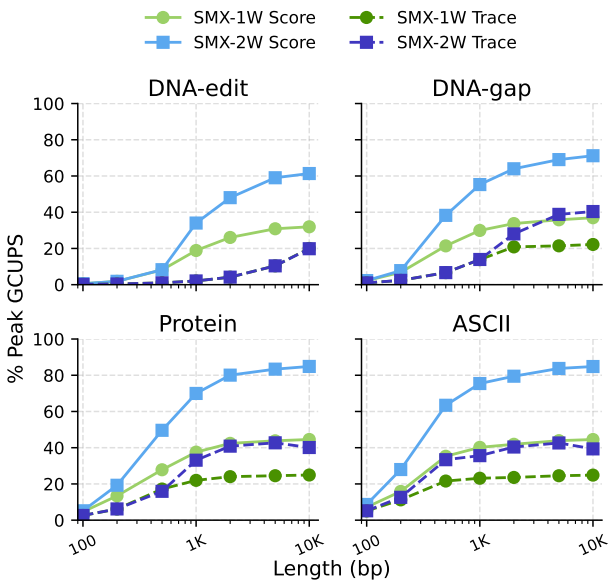

Figure 7: Performance of one and two Worker deployments of SMX in GCUPS (Giga Cells Updated Per Second) relative to the maximum.

Table 1: Post-Implementation Resource Utilization on Zynq UltraScale+ XCK26 (1 SMX-Engine, 2 SMX-Workers).

Resource	Used	Available	Utilization (%)
CLB	**13,548**	14,640	**92.54%**
LUT	73,378	117,120	62.65%
LUTRAM	1,183	57,600	2.05%
FF	42,129	234,240	17.99%
BRAM	5	144	3.13%
DSP	32	1,248	2.56%

ing on mode and element width. This implies we would most likely obtain further performance gains from including a third Worker per Engine, but this was not possible for reasons we will discuss on the following subsection.

8.5 FPGA Utilization and Clock Frequency

Resource utilization metrics were extracted from the Vivado post-implementation reports generated during the Vitis linking process. Table 1 summarizes the utilization of the Zynq UltraScale+ FPGA resources. The final design, configured with one SMX-Engine and two SMX-Workers, utilizes a substantial portion of the available Programmable Logic (PL) resources. More specifically, it uses 92.54% of the available Configuration Logic Blocks (CLBs). According to the Vivado report, each SMX-Worker utilizes around 4K CLBs (27% of the available) and the SMX-Engine occupies around the same amount, while the Memory Bridge occupies 2.5K (17%) and the Control Interface, 2.3K (15%). The sum of the amounts of CLBs of each submodule surpasses that of the global design because a single CLB can be used by multiple modules if Vitis linking manages to share them.

The design meets timing constraints at a clock frequency of 100 MHz. The critical path is located within the SMX-Engine, specifically spanning the combinational logic between the stages of the systolic array. While the frequency could theoretically be increased by deeper pipelining or optimizing data paths, such modifications are constrained by the available resources on the KR260's modest FPGA fabric. As indicated by the high CLB utilization, introducing additional pipeline registers would exacerbate routing congestion, making timing closure increasingly difficult. Even if the 100MHz clock speed is a limiting factor, the accelerator delivers superior performance, which highlights the massive architectural advantage of SMX over general purpose CPUs.

8.6 Energy Efficiency Results

Energy efficiency is a critical metric for evaluating the practicality of computational solutions, especially in large-scale bioinformatics pipelines where operational costs are driven by power consumption, and in portable diagnostic devices where battery life is critical. While raw throughput determines the speed of analysis, the energy cost per alignment dictates the feasibility of scaling the solution to larger datasets or smaller devices. To provide a comprehensive comparison, we evaluate the SMX accelerator against two opposing ends of the general-purpose computing spectrum: a low-power embedded ARM Cortex-A53 (representing edge processing) and a high-performance x86 AMD Ryzen AI 9 HX 370 (representing modern workstations). We utilize the real-world PacBio HiFi dataset, consisting of 512 sequences ranging from 6.6K to 16.3K base pairs. To ensure stable average power measurements, the workload is executed ten times in a continuous loop. We specifically configure the accelerator for DNA alignment with gap penalties in score-only mode, as this computationally dense task best isolates the efficiency of SMX compared to standard SIMD approaches.

Figure 8 (left) illustrates the raw power draw over time. The high-performance x86 baseline (grey) exhibits the highest average power (25.16 W) but completes the workload quickly (5.9 s). The ARM baseline (orange) operates at a lower power envelope (3.89 W) similar to the FPGA, but increases execution time significantly up to 119.8 seconds, rendering it impractical for high-throughput workloads. SMX (blue) combines the best of both worlds: it almost matches the low power profile of the embedded ARM processor (4.34 W) while reducing the execution time of the high-performance x86 CPU (4.9 s).

Figure 8 (right) normalizes these metrics to quantify efficiency. By offloading the quadratic matrix computation to specialized hardware, SMX achieves a dramatic reduction in the energy cost per operation. Specifically, SMX proves to be 7.0× more energy-efficient than the x86 KSW2 baseline and 21.9× more efficient than the ARM KSW2 baseline. This confirms that the SMX architecture not only accelerates time-to-solution but does so with a fraction of the energy budget required by general-purpose processors.

9 Conclusions

In this work, we presented the full-stack integration of the SMX sequence alignment accelerator into the FPGA of a

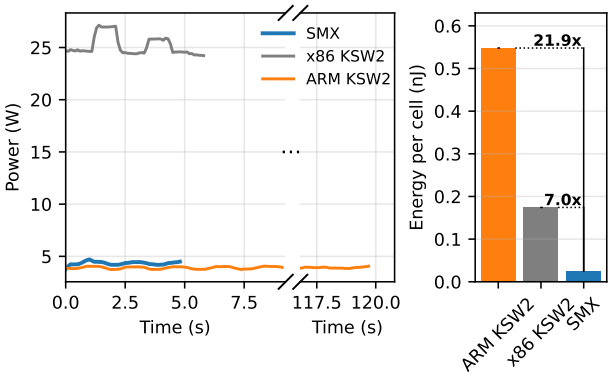

Figure 8: Power (W) and energy (nJ) comparison for DNA-gap alignment of SMX against software baselines running on both ARM (Cortex-A53) and x86 (Ryzen AI 9) architectures. We align a PacBio dataset of 512 sequences ranging from 6.6K to 16.3K bp ten times in a row.

heterogeneous MPSoC. We developed a custom RTL memory bridge that enables tightly coupled, zero-copy data sharing between the FPGA and the host CPU via the Accelerator Coherency Port (ACP). This hardware is orchestrated by a custom software stack built on the Xilinx Runtime (XRT), exposing the accelerator as a seamless library function to the operating system. Our system-level evaluation demonstrates that the integrated SMX system achieves up to $92\times$ speed-up over SIMD-optimized software and proves to be $7.0\times$ more energy-efficient than x86 baselines and $21.9\times$ more efficient than ARM baselines.

In **future work**, we will implement additional algorithmic approaches, such as banded or windowed, making use of the SMX accelerated library, also exploring multi-threaded options. Another interesting improvement would be to scale the design to systems with larger FPGAs such as the Xilinx Versal series boards, into which we could fit more SMX-Workers and SMX-Engines in a system with faster CPUs. We would also like to further automate and generalize this integration process to accelerate the iterative RTL design process of other accelerators.

Acknowledgements

The authors of this work would like to thank Santiago Marco-Sola for his guidance and support throughout the completion of this project. We are thankful to AMD for providing us the hardware platform without which this project wouldn't have been possible.

References

[1] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison, *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge university press, 1998.

[2] J. Felsenstein, "Evolutionary trees from dna sequences: a maximum likelihood approach," *Journal of molecular evolution*, vol. 17, no. 6, pp. 368–376, 1981.

[3] N. Saitou and M. Nei, "The neighbor-joining method: a new method for reconstructing phylogenetic trees.," *Molecular biology and evolution*, vol. 4, no. 4, pp. 406–425, 1987.

[4] A. Šali and T. L. Blundell, "Comparative protein modelling by satisfaction of spatial restraints," *Journal of molecular biology*, vol. 234, no. 3, pp. 779–815, 1993.

[5] D. B. Kitchen, H. Decornez, J. R. Furr, and J. Bajorath, "Docking and scoring in virtual screening for drug discovery: methods and applications," *Nature reviews Drug discovery*, vol. 3, no. 11, pp. 935–949, 2004.

[6] H. Li and R. Durbin, "Fast and accurate short read alignment with burrows–wheeler transform," *bioinformatics*, vol. 25, no. 14, pp. 1754–1760, 2009.

[7] M. A. Hamburg and F. S. Collins, "The path to personalized medicine," *New England Journal of Medicine*, vol. 363, no. 4, pp. 301–304, 2010.

[8] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of Molecular Biology*, vol. 48, no. 3, pp. 443–453, 1970.

[9] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147, no. 1, pp. 195–197, 1981.

[10] W. R. Pearson and D. J. Lipman, "Improved tools for biological sequence comparison.," *Proceedings of the National Academy of Sciences*, vol. 85, no. 8, pp. 2444–2448, 1988.

[11] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of molecular biology*, vol. 215, no. 3, pp. 403–410, 1990.

[12] S. Marco-Sola, J. C. Moure, M. Moreto, and A. Espinosa, "Fast gap-affine pairwise alignment using the wavefront algorithm," *Bioinformatics*, vol. 37, no. 4, pp. 456–463, 2021.

[13] R. Groot Koerkamp and P. Ivanov, "Exact global alignment using a* with chaining seed heuristic and match pruning," *Bioinformatics*, vol. 40, no. 3, p. btae032, 2024.

[14] M. Doblas, O. Lostes-Cazorla, Q. Aguado-Puig, C. Iñiguez, M. Moreto, and S. Marco-Sola, "Quicked: high-performance exact sequence alignment based on bound-and-align," *Bioinformatics*, vol. 41, no. 3, p. btaf112, 2025.

[15] T. Robinson, J. Harkin, and P. Shukla, "Hardware acceleration of genomics data analysis: challenges and opportunities," *Bioinformatics*, vol. 37, no. 13, pp. 1785–1795, 2021.

[16] M. Farrar, "Striped smith–waterman speeds database searches six times over other simd implementations," *Bioinformatics*, vol. 23, no. 2, pp. 156–161, 2007.

[17] Y. Liu, D. L. Maskell, and B. Schmidt, "Cudasw++: optimizing smith-waterman sequence database searches for cuda-enabled graphics processing units," *BMC research notes*, vol. 2, no. 1, p. 73, 2009.

[18] C. W. Yu, K. Kwong, K.-H. Lee, and P. H. W. Leong, "A smith-waterman systolic cell," in *International Conference on Field Programmable Logic and Applications*, pp. 375–384, Springer, 2003.

[19] Y. Turakhia, G. Bejerano, and W. J. Dally, "Darwin: A genomics co-processor provides up to 15,000 x acceleration on long read assembly," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 199–213, 2018.

[20] M. Doblas, P. J. Shih, O. Lostes-Cazorla, M. Moretó, C. F. Batten, and S. Marco-Sola, "Smx: Heterogeneous architecture for universal sequence alignment acceleration," in *Proceedings of the 58th Annual IEEE/ACM International Symposium on Microarchitecture*, 2025.

[21] S. D. Brown, R. J. Francis, J. Rose, and Z. G. Vranesic, *Field-programmable gate arrays*, vol. 180. Springer Science & Business Media, 1992.

[22] D. Chiou, D. Sunwoo, J. Kim, N. A. Patil, W. Reinhart, D. E. Johnson, J. Keefe, and H. Angepat, "Fpga-accelerated simulation technologies (fast): Fast, full-system, cycle-accurate simulators," in *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, pp. 249–261, IEEE, 2007.

[23] H. Li, "Minimap2: pairwise alignment for nucleotide sequences," *Bioinformatics*, vol. 34, no. 18, pp. 3094–3100, 2018.

[24] M. D. Lam, E. E. Rothberg, and M. E. Wolf, "The cache performance and optimizations of blocked algorithms," *ACM SIGOPS Operating Systems Review*, vol. 25, no. Special Issue, pp. 63–74, 1991.

[25] G. Myers, "A fast bit-vector algorithm for approximate string matching based on dynamic programming," *Journal of the ACM (JACM)*, vol. 46, no. 3, pp. 395–415, 1999.

[26] E. W. Myers and W. Miller, "Optimal alignments in linear space," *Bioinformatics*, vol. 4, no. 1, pp. 11–17, 1988.

[27] J. Daily, "Parasail: Simd c library for global, semi-global, and local pairwise sequence alignments," *BMC bioinformatics*, vol. 17, no. 1, p. 81, 2016.

[28] M. Zhao, W.-P. Lee, E. P. Garrison, and G. T. Marth, "Ssw library: an simd smith-waterman c/c++ library for use in genomic applications," *PloS one*, vol. 8, no. 12, p. e82138, 2013.

[29] Y. Liu, A. Wirawan, and B. Schmidt, "Cudasw++ 3.0: accelerating smith-waterman protein database search by coupling cpu and gpu simd instructions," *BMC bioinformatics*, vol. 14, no. 1, p. 117, 2013.

[30] Y.-T. Chen, J. Cong, J. Lei, and P. Wei, "A novel high-throughput acceleration engine for read alignment," in *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*, pp. 199–202, IEEE, 2015.

[31] Z. UltraScale, "Mpsoc technical reference manual," *UG1085*, 2018.