2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)

TALCO: Tiling Genome Sequence Alignment using Convergence of Traceback Pointers

Sumit Walia, Cheng Ye, Arkid Bera, Dhruvi Lodhavia and Yatish Turakhia
Department of Electrical and Computer Engineering
University of California San Diego
{swalia, chye, arbera, dlodhavia, yturakhia}@ucsd.edu

***Abstract*— Pairwise sequence alignment is one of the most fundamental and computationally intensive steps in genome analysis. With the improving costs and throughput of third-generation sequencing technologies and the growing availability of whole-genome datasets, longer alignments are becoming more common in the field of bioinformatics. However, the high memory demands of long alignments create significant obstacles to hardware acceleration. Banding techniques allow recovering high-quality alignments with lower memory, but they also require more memory for long alignments than what is typically available on-chip in hardware accelerators. Recently, tiling-based hardware accelerators have made remarkable strides in accelerating sequence alignment, achieving three to four orders of magnitude improvement in alignment throughput over software tools without any restrictions on alignment length. However, it is crucial to note that existing tiling heuristics can cause the alignment quality to degrade, which is a critical concern for the wider adoption of accelerators in the field of bioinformatics. To address this issue, this paper describes TALCO – a novel method for tiling long sequence alignments, that, similar to prior tiling techniques, maintains a constant memory footprint during the acceleration step independent of alignment length. However, unlike previous techniques, TALCO also ensures *optimal* alignments under banding constraints. TALCO does this by leveraging the convergence of traceback paths beyond a tile to a single point on the boundary of that tile – a strategy that *generalizes* well to a broad set of sequence alignment algorithms. We demonstrate the advantages of TALCO by applying it to two different and widely-used banded sequence alignment algorithms, X-Drop and WFA-Adapt. To the best of our knowledge, this is the first time that a tiling technique is being applied to a non-classical algorithm for sequence alignment, such as WFA-Adapt. The TALCO tiling strategy is beneficial to both software and hardware. When implemented in software, the TALCO strategy reduces the memory requirements for X-Drop and WFA-Adapt algorithms by up to 39× and 57×, respectively, and when implemented as ASIC accelerator, it provides up to 1,900× and 2,000× improvement in alignment throughput/watt over CPU baselines implementing the same algorithms. Compared to state-of-the-art GPU and ASIC baselines implementing tiling heuristics, TALCO provides up to 50× and 1.1× improvement in alignment throughput, respectively, while also maintaining a higher alignment quality. Code availability: https://github.com/TurakhiaLab/TALCO.**

I. Introduction

Pairwise genome sequence alignment is a fundamental building block of many genomic analysis pipelines [1]–[4] and has been studied extensively over the last several decades [5]–[10]. With improved cost, accuracy, and throughput of long-read genome sequencing technologies, coupled with the widespread availability of complete whole-genome assemblies for a large number of species and individual organisms, the ability to align long genomic sequences has gained significant importance in recent years [11]–[13]. In particular, long-read sequencing has led to major breakthroughs in recent history [14]. In genomic medicine, it helped achieve the fastest genetic diagnosis of newborn patients [15]. It has been used to better characterize structural variations and complex regions in the human genome [16]. Long alignments are also essential in comparative genomics to gain new insights into the evolution of gene families and other genomic features by comparing whole genomes of different species [17], [18].

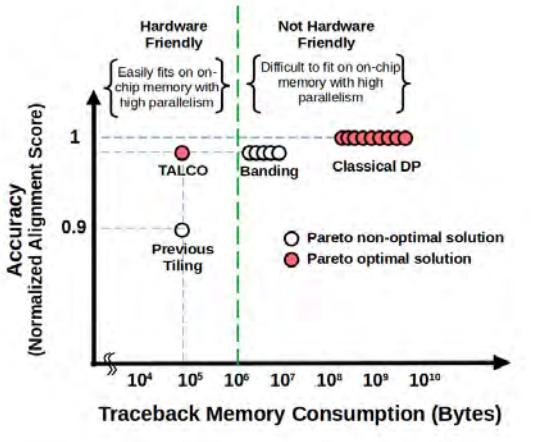

Fig. 1: Accuracy versus memory trade-offs of different alignment techniques highlights how TALCO is Pareto optimal. TALCO achieves the same accuracy as banding techniques while matching the memory requirements of previous tiling techniques.

In sequence alignment, a scoring system is generally used for evaluating and optimizing the quality of different alignments. Classical dynamic programming algorithms [19], [20] yield *optimal* alignments under the most widely-accepted scoring systems (Section II) but these algorithms are impractical for long alignments due to high memory and computational demands. Therefore, widely-used bioinformatic software packages typically utilize various banding strategies for restricting the alignment search space (Section II). This results in a minor drop in alignment scores relative to full dynamic programming but helps to compute long alignments under practical runtime and memory constraints (Figure 1). However, the memory

requirement of these banded algorithms increases linearly with sequence length, which poses significant challenges for hardware acceleration of long alignments. On one hand, the memory requirement is too large for performing long alignments within the limited on-chip memory available on hardware accelerators, and on the other hand, off-chip memory does not offer sufficient bandwidth to achieve meaningful acceleration [21]. This challenge was recently circumvented in the Darwin co-processor [7] through a greedy tiling strategy that allowed alignment subproblems to fit within the on-chip memory, thereby greatly speeding up the compute-intensive portions of the alignment step. This strategy was later incorporated into several other genomic accelerators [8]–[10], [22]–[25]. Although, with appropriate parameters, this strategy can deliver optimal alignment results with a high degree of certainty; however, it cannot *guarantee* the alignments to be optimal or equivalent to their software counterpart which does not employ the same tiling approach (Figure 1), which is a critical concern in the field of bioinformatics and imposes a barrier to the wider adoption of hardware accelerators in real-world applications of critical significance, such as the genomic diagnosis of patients [26]. Since sequence alignment typically accounts for 30%-90% of the total execution time in genomic workloads [27]–[29], based on Amdahl's law [30], there is diminishing return to accelerate this step further over the three to four orders of magnitude speedup already achievable with tiling-based accelerators. Instead, for wider adoption, future hardware accelerators should commensurately focus on improving and *guaranteeing* alignment quality.

To that end, in this paper, we introduce *TALCO* – a novel strategy for tiling alignments using the convergence of traceback pointers. TALCO is a highly generalizable strategy that can be used to perform tiling on a wide range of banded alignment algorithms while ensuring alignment result equivalence. Alignment banding strategies are commonly employed in bioinformatic software for long alignments and have been widely accepted by the bioinformatics community [5], [31]–[34]. This makes TALCO highly useful for the hardware acceleration of existing algorithms for long genomic alignments. The paper makes the following contributions:

1) We introduce a novel tiling strategy, TALCO, that guarantees optimality under banding constraints, i.e., it can be applied to any banded sequence alignment algorithm while guaranteeing that the alignment scores are not degraded due to tiling. In that sense, TALCO is complementary to existing accelerators [7]–[10], [22]–[25] as it can be used to substitute their tiling heuristics while assuring alignment quality. TALCO is based on our unique insight that traceback pointers in alignments have weak long-range dependencies, resulting in the convergence of multiple alignment paths.
2) To demonstrate generalizability, we apply TALCO to widely-used banded sequence alignment algorithms, X-Drop and WFA-Adapt, which are based on very different approaches. We call the modified algorithms TALCO-XDrop and TALCO-WFAA, respectively. To the best of our knowledge, TALCO-WFAA is the first accelerator based on the WFA-Adapt algorithm capable of performing arbitrary long sequence alignments.
3) We demonstrate the benefits of TALCO through software implementation of TALCO-XDrop and TALCO-WFAA that achieve up to 39× and 57× improvement in memory footprint, respectively, for long alignments compared to software baselines. We will open-source our software implementation on publication. We also demonstrate that the convergence of traceback paths that TALCO leverages in these algorithms happens fast, requiring only 3-15% for redundant computation of cells in the overlapping regions of tiles in TALCO for alignment error rates ranging from 1-30%.
4) We present a hardware accelerator design for TALCO-XDrop and TALCO-WFAA which can perform arbitrary long alignments with a small on-chip memory requirement. We performed ASIC analysis of TALCO-XDrop and TALCO-WFAA using FreePDK 45nm process technology. TALCO-XDrop (TALCO-WFAA) requires 58.3mm^2 (30.0mm^2) area and 19.0W (32.9W) of power. Overall, we found that TALCO-XDrop (TALCO-WFAA) ASIC achieves up to 1,900× (2,000×) improvement in alignment throughput/watt over software baselines implementing the same algorithm. TALCO also improves the alignment throughput over state-of-the-art GPU and ASIC baselines that implement tiling heuristics by over 50× and 1.1×, respectively. We have released our RTL code and OpenROAD scripts for ASIC analysis, accessible from GitHub (https://github.com/TurakhiaLab/TALCO).
5) We also synthesized TALCO-XDrop and TALCO-WFAA for FPGAs available on the Amazon EC2 FPGA instances (f1.2xlarge), which achieved operating frequencies of 119MHz and 166MHz, and throughput within 0.15× and 0.29× of ASIC implementations, respectively.

The rest of the paper is organized as follows. Section II provides relevant background on genome sequence alignment, banding techniques, and existing tiling techniques for long genome sequence alignment. We then explain TALCO and its application on two widely adopted sequence alignment algorithms in Section III. Section IV describes the hardware design of TALCO-XDrop and TALCO-WFAA. The experimental methodology is described in Section V. We present the results in Section VI. Section VII describes the related work and Section VIII concludes the paper.

II. Background

Sequence Alignment: Given two input sequences, a query sequence $Q = q_1, q_2, .., q_n$, and a reference sequence $R = r_1, r_2, .., r_m$, the problem of pairwise sequence alignment is to assign gaps (denoted by '$-$') in R and Q to produce a valid alignment that maximizes the alignment score [7]. In a valid sequence alignment, each character in R and Q is aligned with a corresponding character or a gap in the other sequence (Figure 2b,d). Alignments can be scored in many different ways with the typical objective of maximizing the number of

matching characters in the alignment while minimizing the number of mismatches and gaps that need to be inserted in each sequence to achieve the alignment. One particular instance of the scoring strategy involves minimizing the Levenshtein distance [35], in which each mismatch or gap in the alignment increments the distance by one. Gotoh's affine gap [36] strategy assigns different scores for the opening and extension of gaps, providing a more accurate model for how gaps appear in biological sequences.

Needleman-Wunsch and Smith-Waterman algorithms: These are classical approaches to optimally solve the pairwise sequence alignment problem and are based on dynamic programming (DP) algorithms. The Needleman-Wunsch algorithm [20] solves the *global* sequence alignment problem, as defined above, while allowing the flexibility to reward each combination of matching or mismatching characters in the alphabet differently in the alignment score. The Smith-Waterman algorithm [19] is designed to find *local* sequence alignments, i.e., substrings of the input sequences that align with the maximum score. Both algorithms compute a traceback matrix of size $m \times n$, where each cell of the matrix points to one of its adjacent cells (up, left, or diagonal) indicating the direction to arrive at that cell with the maximum score (Figure 2a). The optimal alignment(s) between the two sequences is determined by following a path(s) of traceback pointers in this matrix. Both algorithms have a time and space complexity of $\mathcal{O}(mn)$. Affine gap scoring increases the memory requirement for storing traceback pointers in these algorithms by a constant factor.

WFA algorithm: In 1986, Eugene Myers proposed the $\mathcal{O}(nd)$ algorithm that produces optimal global alignments based on Levenshtein distance [37]. The algorithm works by computing optimal partial alignments of increasing Levenshtein distance until it finds a global alignment between the two input sequences. The algorithm requires $\mathcal{O}(nd)$ time and $\mathcal{O}(d^2)$ memory, where n is the length of the shorter sequence and d is the optimal Levenshtein distance between the sequences. The algorithm uses a triangular-shaped traceback matrix (Figure 2c-d), whose columns correspond to the diagonals of the DP matrix and rows correspond to the alignment score (i.e., the number of mismatches and gaps in the partial alignment up to that row). Each cell stores a traceback pointer with three possible configurations, pointing to the same, adjacent-left, or adjacent-right cell in one row above that cell, using which the optimal alignment can be reconstructed. WFA [38] is a recent extension of the $\mathcal{O}(nd)$ algorithm to support affine gaps and generalized distance measures. Here too, supporting affine-gap penalties result in a constant-fold increase in the memory requirement. When the differences between the two sequences are small, Myers' $\mathcal{O}(nd)$ and WFA algorithms are very fast and memory-efficient in comparison to classical DP approaches. However, the performance can also be worse when the two sequences are long (and hence, have more differences) or dissimilar, or when scoring parameters span a large range of values [39]. BiWFA [40] is a recent extension of WFA that trades runtime for memory efficiency, but it still requires up to hundreds of megabytes for long and noisy alignments.

Banding techniques: Banding techniques improve the speed and memory requirements of the alignment algorithms by heuristically pruning the search space. X-Drop [41] has been a popular banding technique for classical DP algorithms since its incorporation into the BLAST [34] algorithm and virtually all long-read and whole-genome aligners use some form of this banding technique [5], [33]. Briefly, the X-Drop algorithm dynamically prunes out cells at the edges of each anti-diagonal as it computes them if they fall below the best score seen so far by more than a user-defined parameter, X, as shown in Figure 2e. The WFA software library also has a different variation of this heuristic, WFA-Adapt (WFAA) [38], which drops the outer columns of the triangular matrix which are lagging much behind the furthest in terms of the number of bases covered in the alignment.

Tiling techniques: Though banding techniques reduce the memory requirement of alignment algorithms, it still usually grows linearly to the lengths of the sequences being aligned. This limits the parallelism as well as the maximum alignment length that can be supported in custom hardware relying on the on-chip memory to achieve massive acceleration. To make alignment more amenable to hardware acceleration, a tiling heuristic, called GACT, was introduced in Darwin [7] and later adapted in other accelerators [8]–[10], [22]–[25]. Briefly, this technique extends an alignment using a series of overlapping tiles, with tile size and overlap determined by fixed parameters, T and O, as shown in Figure 2f. It required only $\mathcal{O}(T^2)$ memory, corresponding to a single tile of the alignment and independent of the lengths of sequences being aligned, to be maintained on-chip (the traceback path was stored off-chip in $\mathcal{O}(n)$ memory), allowing arbitrary long alignments to be accelerated in hardware with high speedup. In Darwin-WGA [8], the authors introduced GACT-X, which incorporated X-Drop banding in GACT. Even though this can give good practical results, the GACT-X heuristic is not guaranteed to provide equivalent alignments to X-Drop and the performance is sensitive to properties of the input sequences, and the choice of T and O. So far, tiling has not been explored for the WFA algorithm.

III. TALCO Algorithm Description

Motivation: Figure 3 provides motivation for the TALCO algorithm. The figure shows the traceback pointer matrix for two sequences, aligned using the Needleman-Wunsch algorithm with X-Drop banding. Traceback pointers reachable from a starting wavefront (i.e., an anti-diagonal in the matrix) marked with a dotted line are highlighted in red. As the figure shows, alignment paths starting from different cells on the wavefront merge at a single cell in the matrix, highlighted with a blue circle, and then share a common path to the origin (i.e., the top-left cell). Such convergence of traceback pointers from nearby cells (which motivates our definition of a "frontier" below) is a prevalent characteristic observable in many alignments and closely resembles the "rank convergence" property of dynamic programming algorithms found in a previous paper [42]. In

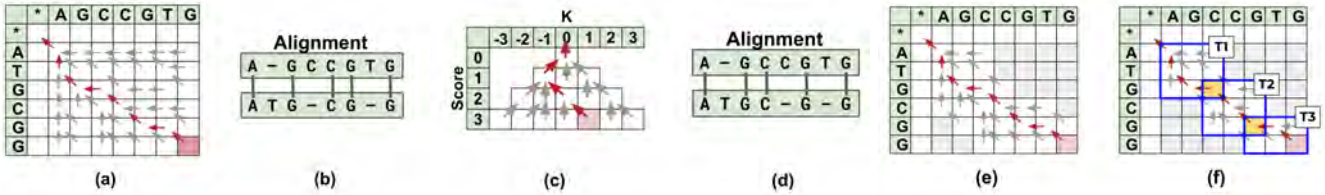

Fig. 2: Traceback matrices and alignments obtained from Needleman-Wunsch and WFA alignment algorithms, for example, input sequences R='AGCCGTG' and Q='ATGCGG'. (a) Traceback matrix for the Needleman-Wunsch algorithm with scoring parameters: {`match=2`, `mismatch=-1`, `gap=-1`}. Each cell stores a single traceback pointer pointing up, left, or diagonal, with the traceback path starting from the pink cell highlighted in red. (b) Alignment corresponding to the traceback path in (a). (c) Traceback matrix for the WFA algorithm with scoring parameters: {`mismatch=1`, `gap=1`}, with the traceback path starting from the pink cell highlighted in red. (d) Alignment corresponding to the traceback path in (c). (e) Traceback matrix for the X-Drop algorithm (adaptive banding) with scoring parameters same as (a) and {`X=3`}. Pruned cells are highlighted in gray, and the traceback path starting from the pink cell is highlighted in red. (f) Illustration of the tiling heuristic (GACT) with parameters: {`T=3`, `O=1`}. Overlapping cells between two consecutive tiles are highlighted in yellow, and the traceback path starting from the pink cell is highlighted in red.

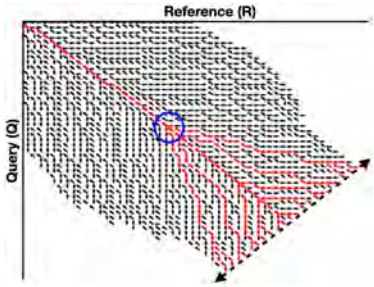

Fig. 3: Illustration of the convergence of traceback pointers using an example traceback matrix for Needleman-Wunsch algorithm with X-Drop banding.

that paper, the authors postulated that rank convergence is frequently observed because the score dependence between two cells in the DP matrix weakens as their distance increases, which we observe is also the case with traceback pointers. However, we note here that while TALCO takes advantage of the traceback convergence property to improve the speed of tiling, it is not a necessary condition for the algorithm to work or provide optimal results (under banding constraints).

Frontiers and markers: The TALCO algorithm defines two terms, frontier and marker, as follows. The i^{th} **frontier**, f_i, is defined as a set of cells in the traceback matrix such that the scores and traceback pointers for cells in f_i depend only on the previous frontier, f_{i-1}. Hence, if $j < k$, f_j will be computed before f_k in the alignment process. There could be many ways to define a frontier in an alignment algorithm. When all frontiers are combined, they cover the entire traceback matrix and adjacent frontiers can have overlapping cells. For example, in the X-Drop algorithm, the j^{th} frontier can be defined as the set of cells in two adjacent wavefronts, W_j and W_{j+1}, as shown in Figure 4b. Since a traceback pointer in W_{j+2} can only point to W_{j+1} (in case of up or left pointers) or W_j (in case of diagonal pointers) wavefronts, any traceback path from f_k is guaranteed to pass through f_j, if $j < k$. For the WFAA algorithm, the frontier definition depends on the choice of scoring parameters. When Levenshtein distance scoring is used, where every mismatch and gap has the same penalty of one, a frontier can be defined as a single row in the triangular traceback matrix. This is because, each row in the matrix corresponds to the alignment score (number of mismatches and gaps) in the partial alignment up to that row, and any alignment with a score of $s+1$ or higher would contain a partial alignment of score s. With affine gap scoring, for e.g., with a penalty of 1 for extending gaps and an extra penalty of 1 for opening gaps, the frontier needs two consecutive rows of the traceback matrix, as alignment from row s must pass through row $s-1$ or $s-2$. Therefore, for the WFAA algorithm with the aforementioned scoring parameters, the j^{th} frontier can be defined as the set of cells in two adjacent wavefronts, W_j and W_{j+1}, as shown in Figure 5b. A **marker** is a special frontier (f_M) in the TALCO algorithm that is used to separate the two phases of the TALCO algorithm.

TALCO Algorithm: TALCO takes advantage of the property that if optimal traceback paths starting from every cell on a frontier f_Y converge to a single cell C_{tb}, then since optimal traceback paths starting from later frontiers would pass through some cell on f_Y, they would also pass through C_{tb}. Algorithm 1 provides the pseudocode for the TALCO algorithm which exploits this insight to tile an arbitrary alignment algorithm, *Align*, which has a scoring function called *AlignScore* and a traceback function called *AlignTraceback*.

TALCO, within each tile, uses *AlignScore* to compute the scores (S_{curr}) and traceback pointers (Ptr_j) for each cell in the current frontier (f_j). It does this using: a) the scores from the previous frontier (S_{prev}), and b) the corresponding bases in R and Q (line 9). *AlignScore* also returns the coordinates of the last cell, C_{last}, which depends on whether it is a global or local alignment algorithm. For example, for a global alignment function, such as WFA-Adapt, C_{last} will correspond to the last cell (m,n) in the dynamic programming matrix, and for a local alignment function, such as X-Drop, C_{last} will correspond to the maximum scoring cell found so far. At the end of each tile (line 37), the current traceback path is extended by up to M frontiers, where M is fixed, using a) the *AlignTraceback*

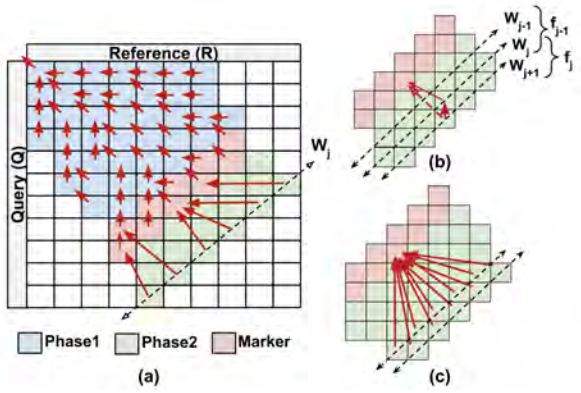

Fig. 4: An illustration of TALCO-XDrop (TALCO applied to the X-Drop algorithm). (a) Traceback pointer matrix in the TALCO-XDrop algorithm with cells colored by the phases in which they were computed (cells outside the X-Drop bands are in white) (b) An illustration of traceback pointer redirection in TALCO-XDrop. The cell on wavefront W_{j+1} is redirected to the marker cell on its traceback path by the cell on wavefront W_j that it would otherwise point to. (c) At convergence, all cells on a frontier (two consecutive wavefronts) would point to the same cell on the marker.

Algorithm 1 Tiling Alignment using the TALCO Algorithm

1: $i \leftarrow 0$
2: $tb \leftarrow []$ ▷ Initialize empty traceback path
3: $S_{marker} \leftarrow AlignInit()$
4: **while** *True* **do** ▷ Multiple Tiles, outer loop
5: $j \leftarrow i$
6: $S_{prev} \leftarrow S_{marker}$
7: $TB_{Tile} \leftarrow$ 2-D matrix with M rows
8: **while** *True* **do** ▷ Within Each Tile
9: $(S_{curr}, Ptr_{f_j}, C_{last}) \leftarrow AlignScore(S_{prev}, f_j, R_{f_j}, Q_{f_j}, S_{prev})$
10: **if** $j - i = M - 1$ **then** ▷ Reached Marker
11: $ConvPtr_{prev} \leftarrow$ A unique id for each cell in S_{curr}
12: $S_{marker} \leftarrow S_{curr}$
13: **end if**
14: **if** $j - i < M$ **then** ▷ Phase 1
15: $TB_{Tile}[j-i] \leftarrow Ptr_{f_j}$
16: **if** f_j has reached the end of the alignment **then**
17: $C_{tb} \leftarrow C_{last}$
18: **break**
19: **end if**
20: **else** ▷ Phase 2
21: $ConvPtr_{curr} \leftarrow redirect(Ptr_j, ConvPtr_{prev})$
22: **if** C_{last} was updated in the current tile **then**
23: $ConvPtr_{last} \leftarrow ConvPtr_{curr}[C_{last}]$
24: **end if**
25: **if** all pointers in $ConvPtr_{curr}$ point to a marker cell C_x **then**
26: $C_{tb} \leftarrow C_x$
27: **break**
28: **else if** f_j has reached the end of the alignment **then**
29: $C_{tb} \leftarrow ConvPtr_{last}$
30: **break**
31: **end if**
32: $ConvPtr_{prev} \leftarrow ConvPtr_{curr}$
33: **end if**
34: $j \leftarrow j + 1$
35: $S_{prev} \leftarrow S_{curr}$
36: **end while**
37: $tb.append(AlignTraceback(TB_{Tile}, C_{tb}))$
38: $i \leftarrow i + M$
39: **if** f_i has reached the end of the alignment **then**
40: **break**
41: **end if**
42: **end while**
43: **return** tb ▷ Return traceback path

function on TB_{Tile}, b) the traceback matrix for that tile, and c) C_{tb}, the starting cell for the traceback.

Each tile in TALCO has two phases. In phase 1, TALCO stores traceback pointers for each cell in the current frontier, Ptr_{f_j}, as a row in TB_{Tile} matrix (line 15). It does this for a total of M frontiers, after which it reaches the marker in the current tile and begins phase 2 (line 20).

In phase 2, TALCO computes convergence pointers, $ConvPtr_{curr}$, for each cell in the current frontier. A convergence pointer indicates the cell on the marker that the traceback path from the corresponding cell on the frontier would lead to. This is done recursively for each cell on the current frontier using a redirect function (line 21), that stores the same pointer of the cell on the previous frontier that the current cell traceback pointer points to. Figures 4 and 5 show how the redirection of pointers works for X-Drop and WFAA algorithms. Convergence pointers for only two frontiers, current and previous, need to be maintained in this phase. If all convergence pointers in the current frontier point to the same marker cell (we referred to this frontier earlier as f_Y), it can be used to determine the starting cell for traceback in the current tile, C_{tb}, and terminate phase 2 (lines 25). Figures 4 and 5 illustrate this for X-Drop and WFAA algorithms. This method of using the convergence of traceback pointers instead of a greedy approach to determine the starting cell C_{tb} for traceback in a tile is what differentiates TALCO from previous tiling approaches.

Memory requirement: If B_{max} is the maximum number of cells in a frontier, then at most $M \times B_{max}$ pointers and B_{max} scores in S_{prev} and S_{curr} will need to be maintained during phase 1 of TALCO (Algorithm 1). In phase 2, B_{max} pointers will be maintained for frontiers $ConvPtr_{prev}$ and $ConvPtr_{curr}$, with each pointer requiring $log_2(B_{max})$ bits for pointing to a unique cell on the marker. Hence, the maximum memory required for a single tile in TALCO is $\mathcal{O}(M \times B_{max} + log_2(B_{max}) \times B_{max})$. For a local extension algorithm, additional $log_2(B_{max})$ memory bits are required to store the score and convergence pointer of the maximum scoring cell. Therefore, the memory requirement for each tile in TALCO depends only on M and B_{max}, irrespective of the lengths of input sequences. This also allows a memory-constrained accelerator to fall back to the host when a frontier size exceeds the maximum size it can handle to guarantee equivalent alignment results with and without tiling, a feature that is missing in the previous tiling approaches.

Proof of equivalence: TALCO algorithm (Algorithm 1) is guaranteed to produce alignments identical to *Align*, when *Align* takes full input sequences and does not perform tiling.

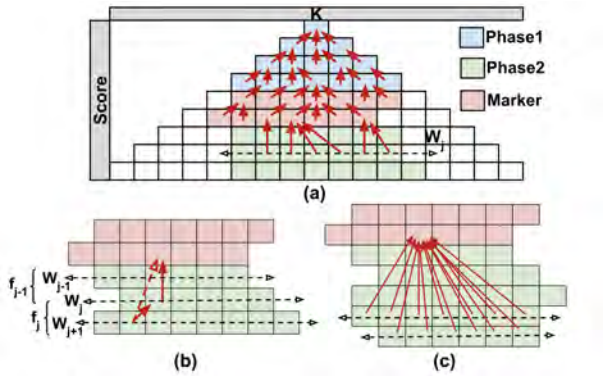

Fig. 5: An illustration of TALCO-WFAA (TALCO applied to the WFA-Adapt algorithm). (a) Traceback pointer matrix in the TALCO-WFAA algorithm with cells colored by the phases in which they were computed (cells outside the WFAA bands are in white). (b) An illustration of traceback pointer redirection in TALCO-WFAA. The cell on wavefront W_{j+1} is redirected to the marker cell on its traceback path by the cell on wavefront W_j that it would otherwise point to. (c) At convergence, all cells on a frontier (two consecutive rows) would point to the same cell on the marker.

We only provide intuitive proof for this here due to space constraints. Since TALCO uses the marker scores from the current tile to correctly initialize the next tile (line 6), the scores and traceback pointers in TALCO and *Align* will always match for each cell. The partial alignment produced by TALCO in the current tile (line 37) is also guaranteed to be present in *Align* without tiling. This is because the TALCO algorithm guarantees that *Align* traceback also passes through C_{tb} in each tile. For example, consider the case in which all convergence pointers at the current frontier, f_j, point to a single cell, C_x, on the marker (line 25). Hence, if traceback *Align* starts from a cell on a frontier f_k, which occurs after f_j, the traceback is guaranteed to include C_x, which is the value assigned to C_{tb} in line 26. This is because since $k > j$, traceback from every cell on f_k passes through some cell f_j, which passes through C_x. Remaining parts of the Algorithm set the value of C_{tb} correctly for cases in which convergence does not take place before the end of the alignment is reached (lines 28-29) or when *Align* algorithm is local and the maximum scoring cell starts before the converging frontier (lines 22-23).

IV. TALCO Accelerator Design

Figure 6 provides an overview of accelerator design for TALCO-*Align* (TALCO applied to tile an algorithm, *Align*). The accelerator is implemented as a co-processor. Host CPU issues alignment tasks along with the addresses of sequence pairs to be aligned to parallel TALCO-*Align* units and receives the output. Host CPU also performs alignments that fail because of frontier lengths exceeding the hardware constraints. The overhead for failed alignments corresponds to the alignment speed of the software. Next, we describe the TALCO accelerator hardware design for tiling X-Drop and WFA-Adapt algorithms.

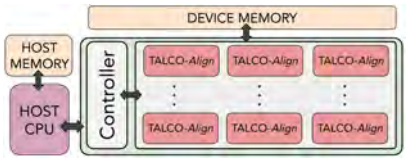

Fig. 6: Overview of the interconnection between the host CPU and TALCO-*Align* co-processor, where *Align* denotes the sequence alignment algorithm that TALCO is applied to. In this paper, we discuss *Align* for X-Drop and WFA-Adapt.

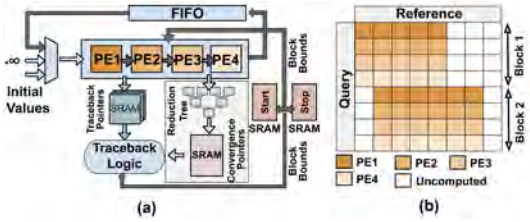

Fig. 7: TALCO-XDrop hardware design. (a) Systolic architecture of TALCO-XDrop array with $N_{PE}=4$. (b) Query blocking applied to an example dynamic programming (DP) matrix whose cells are colored corresponding to the PE computing them (uncomputed cells outside the X-Drop bands are in white).

TALCO-XDrop Accelerator Design: The TALCO-XDrop accelerator closely resembles GACT-X [8] but requires additional elements to incorporate the second phase of the TALCO algorithm shown within the gray box in Figure 7a. The TALCO-XDrop accelerator aligns two sequences, a reference (R) and a query (Q), based on the X-Drop algorithm with TALCO tiling. Input sequences are ASCII encoded but stored using 3 bits per base-pair (bp) in SRAM for an extended DNA alphabet $\sigma \in \{A, C, G, T, N\}$, where N is any ambiguous character. Tiling parameters M, the number of wavefronts from the beginning of a tile to the marker, and B_{max}, the maximum number of cells in a wavefront, are fixed at design time. If any wavefront exceeds B_{max} cells, the accelerator falls back to the CPU with an invalid flag in the corresponding output to perform the alignment. Alignment scoring parameters can be passed to hardware as input, and include the substitution matrix (W), gap open penalty (o), and gap extend penalty (e).

As shown in Figure 7, the TALCO-XDrop accelerator calculates the scores of the DP matrix using a systolic array of processing elements (PEs), exploiting the available wavefront parallelism along a stripe of N_{PE} rows, similar to GACT-X [8]. Figure 7 shows a TALCO-XDrop array with $N_{PE} = 4$. During phase one, each PE computes the scores according to the Needleman-Wunsch [20] scoring equations with an affine gap [36] penalty (Eqs. (1) to (3)). The final score is calculated using Eq. (3) which determines the direction pointer. Each pointer has 4 bits: 2 bits to encode Eq. (3) and 1 bit each for Eq. (1) and Eq. (2). Further, a running maximum score (H_{max}) is maintained in a systolic fashion. A stripe calculation terminates when scores of all cells along a column drop below ($H_{max} - X$), where X is an input parameter of the X-Drop algorithm as shown as the uncomputed region in Figure 7. H and D values for the last row of a stripe are stored in a FIFO,

as shown in Figure 7 since they are required for computing the next stripe. Depending upon whether the value in the last row of the previous stripe was calculated or not, values are read from FIFO, or initialized to $-\infty$. Each PE has a dedicated SRAM bank to store traceback pointers and additional SRAMs are used to store the start and end positions of each stripe, as they vary based on the X value, and are during the traceback.

$$I_{i,j} = max\{H_{i,j-1} - o, I_{i,j-1} - e\} \quad (1)$$
$$D_{i,j} = max\{H_{i-1,j} - o, D_{i-1,j} - e\} \quad (2)$$
$$H_{i,j} = max\{I_{i,j}, D_{i,j}, H_{i-1,j-1} + W(R_i, Q_j)\} \quad (3)$$

Traceback pointers are only stored during phase 1 of TALCO (Algorithm 1). When the TALCO algorithm moves to phase 2 (Algorithm 1), each PE uses pointer redirection to calculate the cell on the marker to which its traceback path would lead, as described in Section III and shown in Figure 5b. A single register H'_{max} keeps track of the maximum score seen so far. A reduction tree is used to determine if all N_{PE} pointers computed by N_{PE} PEs in each cycle for the subset of the wavefront in the current stripe have converged to the same cell. If they have converged, the index of the converged cell on the marker is stored in a separate SRAM for convergence pointers (Figure 7) addressed by the current wavefront index, otherwise, -1 is stored. Once pointers for all subsets of a frontier are obtained, appended values determine convergence. If all the values are the same, convergence is detected, otherwise the algorithm continues.

Once the convergence is detected on a frontier, score calculation is performed till $H'_{max} > H_{max}$. If the criteria match, traceback logic starts from the converged point, otherwise, the cell with H_{max} serves as the starting point. The traceback logic traverses each step depending on the pointers and start/stop values stored in the SRAM. The traceback path of the current tile is sent to the software to reconstruct the alignment.

TALCO-WFAA Accelerator Design: Similar to TALCO-XDrop, TALCO-WFAA Array hardware design accelerates the alignment of two sequences, a reference (R) and a query (Q) using the TALCO-WFAA algorithm explained in Section III. Input sequences are again ASCII encoded and stored using 3 bits in SRAM for an extended DNA alphabet $\sigma \in$\{A, C, G, T, N\}. TALCO-WFAA accelerator computes the alignment by iteratively performing *extend*, *reduce*, and *compute* steps described in WFA-Adapt [38]. Here too, tiling parameters M, the number of wavefronts from the beginning of a tile to the marker, and B_{max}, the maximum number of cells in a wavefront, are fixed at design time. If any row exceeds B_{max} cells, the accelerator falls back to the CPU with an invalid flag in the corresponding output to perform the alignment. In addition, WFA-Adapt alignment parameters are decided at design time as they determine the traceback pointer size and cell dependencies [43]. In our case, we set mismatch (x), gap open (o), and gap extend (e) penalties to 1, 2, and 1, respectively, so that the frontier consists of two rows in the traceback matrix, as described in Section III.

Figure 8 shows an array of the TALCO-WFAA accelerator which exploits parallelism using N_{ext} extend and N_{comp} compute modules. Due to WFA-Adapt's irregular access pattern, we dedicate a pair of SRAM to store query and reference sequences for every *extend* and *compute* block. ❶ During the first phase of TALCO-WFAA, each *extend* module receives a cell offset and a K position in the triangular matrix. On the start signal, *extend* compares bases of R and Q from positions r_i and q_i respectively, calculated using the cell offset and K, until a mismatch is found, and returns the new offset. To exploit parallelism, multiple consecutive bases of R and Q are compared simultaneously and a reduction tree is used to compute the offset of the earliest mismatch. Once a new offset for all the cells, within k_{max} and k_{min}, corresponding to a score in the triangular matrix is computed, *reduce* module is sent a start signal. *Reduce* logic performs pruning of cells at the boundaries of each new row whose new offsets are left far behind based on the WFA-Adapt heuristic [38]. ❷ Another reduction tree is used to determine distant cells and generate k'_{max} and k'_{min}, stored in dedicated SRAMs as shown in Figure 8. ❸ Four arrays of registers, one for each H_{s-1}, I_{s-1}, D_{s-1}, and H_{s-2} rows in the triangular matrix are used to store cell offsets of a frontier. ❹ Next, *compute* module finds the farthest-reaching cell offset for a cell based on the previously computed frontier using Eqs. (4) to (7) of the WFA algorithm [38]. ❺ *Compute* further calculates the traceback pointers and stores them in SRAM and the iteration continues.

$$I_{s,K} = max\{M_{s-o,K-1} + 1, I_{s-e,K-1} + 1\} \quad (4)$$
$$D_{s,K} = max\{M_{s-o,K+1}, D_{s-e,K+1}\} \quad (5)$$
$$X_{s,K} = max\{I_{s,K}, D_{s,K}, M_{s-x,K} + 1\} \quad (6)$$
$$H_{s,K} = X_{s,K} + LCP(Q[X_{s,K} - K, n-1], R[X_{s,K}, m-1] \quad (7)$$

In the second phase, *extend* and *reduce* modules perform the conventional tasks however, *compute* module, instead of traceback pointers, convergence pointers are determined and stored in registers H_{s-1}, I_{s-1}, D_{s-1}, and H_{s-2}. No further traceback pointer SRAM memory utilization occurs. ❻ A reduction tree is used to detect convergence in a frontier. ❼ If converged, traceback logic generates the compact CIGAR one character at a time using the pointers and k_{max}, and k_{min} values stored in the SRAM. If the end of sequences is reached before convergence, traceback starts from (m,n) where m and n are the lengths of reference and query sequences, respectively. The result is generated as a compact CIGAR and later unpacked into full CIGAR using CPU threads.

TALCO-WFAA accelerator design takes some inspiration from WFA-FPGA design [43]. Apart from the extra logic to incorporate TALCO's phase 2, TALCO-WFAA and WFA-FPGA have two additional differences in their implementation. First, WFA-FPGA accesses SRAM to calculate the next wavefront offsets from the previous, while TALCO-WFAA uses dedicated registers (in ❸) to reduce memory accesses that favor hardware acceleration (this works better on ASICs, but WFA-FPGA design is more FPGA-friendly). Furthermore,

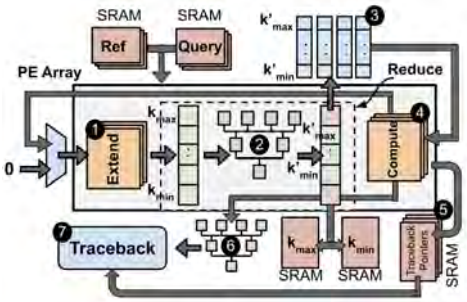

Fig. 8: TALCO-WFAA hardware design

WFA-FPGA has parallel *extend* units but a single *compute* unit is used, however, TALCO-WFAA exploits parallelism with both *extend* (in ❶) and *compute* (in ❹) units.

V. Experimental Methodology

Dataset: To generate sequence pairs for evaluating alignment throughput and accuracy at different alignment lengths and error rates, we started with the human genome assembly, GRCh38, only using the 1–22, X, and Y chromosomes. Using this as the reference, we simulated 32 sets of long reads using PBSIM2 [44] with lengths {10, 20, 50, and 100} Kbp and error rates {1%, 5%, 15%, and 30%}, resembling error profiles of Pacific Biosciences (PacBio) and Oxford Nanopore Technology (ONT) (Table I). Each set included 20,000 reads. PBSIM2 provided the subset of the reference genome aligning with each read, and the corresponding alignment, as output. Default settings of the continuous long read (CLR) [45] model were used in PBSIM2 to generate PacBio reads and tuned to model ONT reads [46].

Read Type	PacBio				ONT			
Substitution error	0.06%	0.28%	0.83%	1.65%	0.39%	1.97%	5.91%	11.82%
Insertion error	0.46%	2.28%	6.83%	13.65%	0.25%	1.21%	3.64%	7.27%
Deletion error	0.48%	2.45%	7.35%	14.70%	0.36%	1.82%	5.45%	10.91%
Total error	**1%**	**5%**	**15%**	**30%**	**1%**	**5%**	**15%**	**30%**

TABLE I: Error profile of the read sets

Baseline Methods: We used Libgaba [47], WFA-Adapt [38], BiWFA [40], Edlib [48], and Scrooge [10] as our software (CPU) baselines. Specifically, we compared the alignment throughput, memory footprint, and power consumption of TALCO-XDrop with Libgaba (semi-global alignment with X-Drop termination), and TALCO-WFAA with the WFA-Adapt algorithm in WFA2-lib, since Libgaba (WFA-Adapt) algorithm is equivalent to TALCO-XDrop (TALCO-WFAA), whereas other baselines are included for alignment throughput comparison only. We evaluate BiWFA `v2.3` with default settings and affine gap scoring scheme, and Edlib `v1.2.7` with default settings.

All CPU analyses were performed on a 64-core Intel Xeon Silver 4216 processor, running at 2.1 GHz with 384 GB DDR4 RAM clocked at 3.2 GHz. We demonstrate the benefits of TALCO using CPU implementations of TALCO-XDrop and TALCO-WFAA, implemented in `C++` with multithreading using OpenMP [49] and compiled using `g++-10.3`. For a fair comparison, all baselines were configured to obtain the maximum throughput without affecting the alignment score obtained from the recommended default settings. Scoring and other input parameters for TALCO-XDrop and TALCO-WFAA were tuned to match the baseline tools. We implemented a software version of TALCO-XDrop from scratch in `C++`, and for TALCO-WFAA, we modified the unidirectional align function of the WFA library (adopted in WFA-Adapt) to use the TALCO tiling strategy. We did this by modifying the *compute* function to forward the convergence pointers and check for convergence, while the *reduce* and *extend* function invocations remained the same. We measured each tool's processor and DRAM power utilization using BenchExec [50]. The peak memory utilization of each process was tracked using the `VmPeak` utility in Linux.

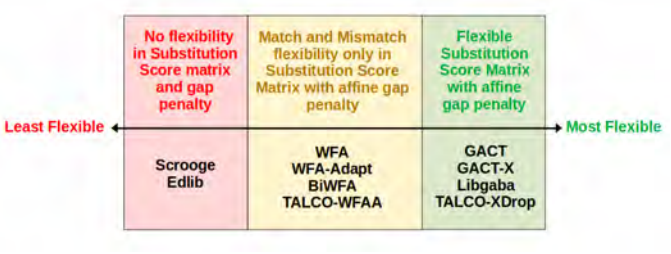

Fig. 9: Illustration of flexibility in the alignment scoring function of all evaluated tools, including TALCO. Tools with greater scoring flexibility tend to exhibit lower speed and memory efficiency but solve a more generalized formulation of the alignment problem with broader bioinformatic utility.

We further compared the alignment throughput of TALCO accelerators against the state-of-the-art GPU and ASIC aligners based on the code and RTL availability. We used the GPU implementations of the GACT [51] and Scrooge [10] algorithms for GPU baselines. We performed all GPU analyses on NVIDIA RTX A6000 GPU [52] and used NVIDIA Nsight [53] to collect the kernel runtime. For the ASIC baseline, we used the GACT-X RTL code available from Darwin-WGA [8]. GACT-X implements a tiling heuristic on the X-Drop algorithm, making it suitable for a direct comparison with TALCO-XDrop. We synthesized and executed GACT-X as described below to estimate its performance.

Figure 9 highlights the relative flexibility of different methods in their scoring function, as it tends to impact their relative speed and memory performance.

ASIC Area, Power, and Throughput Analysis: We used SystemVerilog to implement the RTL design of TALCO-XDrop and TALCO-WFAA accelerators described in Section IV. We used OpenROAD [54], with OpenRAM [55] as the memory compiler, to perform ASIC analysis of our designs and GACT-X through place-and-route on a FreePDK 45nm process technology [56] at the worst-case process-voltage-temperature (PVT) corner. Each of our accelerators was provisioned with the required number of DDR4-2400 channels, and DRAM-

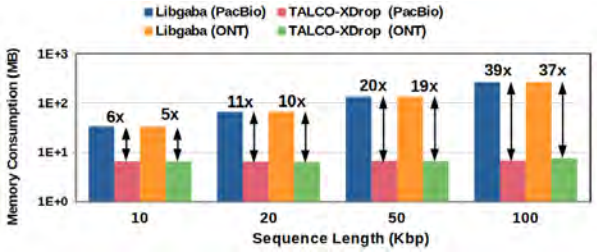

Fig. 10: Memory footprint of a single-thread CPU execution of Libgaba and TALCO-XDrop for aligning PacBio and ONT reads of different lengths with 15% error rate.

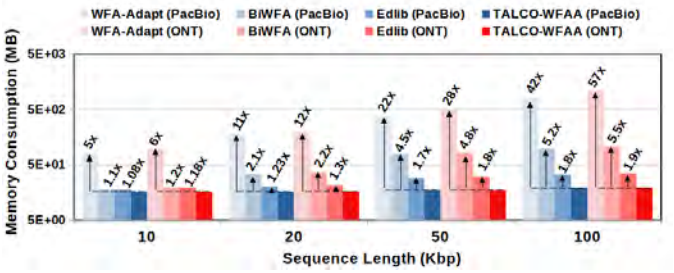

Fig. 11: Memory footprint of a single-thread CPU execution of WFA-Adapt, BiWFA, and TALCO-WFAA for PacBio and ONT reads of different lengths with 15% error rate.

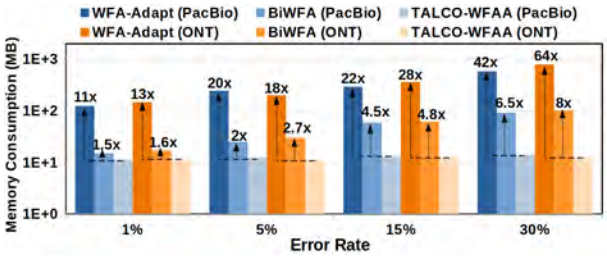

Fig. 12: Memory footprint of a single-thread CPU execution of WFA-Adapt, BiWFA, and TALCO-WFAA for 50Kbp PacBio and ONT reads at error rates.

Power (version 4) [57] was used to estimate the DRAM cycles and power. ASIC alignment throughput for TALCO-XDrop, TALCO-WFAA, and GACT-X was estimated using the maximum ASIC frequency reported from OpenROAD analysis and cycle counts required to align each pair of sequences on individual arrays derived from FPGA implementations of the accelerators (see below).

FPGA Analysis: To evaluate the achievable performance of TALCO-XDrop and TALCO-WFAA accelerators on FPGAs, we synthesized them on Xilinx Virtex UltraScale Plus `xcvu9p-flgb2104-2-i` FPGA [58] alongside the AWS shell [59] for deployment on the Amazon EC2 FPGA instances (f1.2xlarge). We were able to map 32 arrays of TALCO-XDrop, each with 16 PEs, and 16 arrays of TALCO-WFAA, each with 16 *extend* and 16 *compute* modules. The operating clock frequency was 115MHz for TALCO-XDrop and 166MHz for TALCO-WFAA. We also synthesized a single array of GACT-X with 32 PEs on Xilinx Virtex UltraScale Plus `xcvu9p-flgb2104-2-i` FPGA [58] with an operating frequency of 150MHz to derive the cycle counts to align pairs of sequences.

VI. Results and Discussion

Software Memory Footprint with TALCO: The TALCO tiling strategy can be used to improve the memory footprint of software aligners without affecting the alignment results. To quantify the benefits, we compared the memory footprints of software implementations of TALCO-XDrop and TALCO-WFAA with Libgaba and WFA-Adapt, respectively, for aligning reads of different lengths. Figures 10 and 11 show these results for the single-threaded execution of these algorithms. Since baseline algorithms use banding but do not incorporate tiling, their memory footprint increases linearly with read length. For Libgaba (WFAA), the memory footprint increased by 7× (14.5×), from 33 MB (57 MB) to 260 MB (830 MB) as PacBio read lengths were increased from 10 Kbp to 100 Kbp, whereas the memory footprint of TALCO-XDrop (TALCO-WFAA) increased by only 1.1× (1.8×), from 6.2 MB (11 MB) to 6.54 MB (19.7 MB). A small increase in the memory footprint of the TALCO algorithms can be attributed to the memory needed to store sequences and their alignment in memory.

Because the memory footprint of the WFA-Adapt is also dependent on the rate of differences between the sequences being aligned, we further evaluate TALCO-WFAA and WFA-Adapt for aligning sequences of fixed length at different error rates in Figure 12. As expected, WFA-Adapt had a roughly 15× higher memory footprint when the error rate of reads was increased from 1% to 30%, whereas TALCO-WFAA had a lower, and roughly constant memory footprint, providing 10–30× and 12–64× improvement for PacBio and ONT reads, respectively (Figure 12). We also found that while BiWFA indeed improves the memory footprint of WFA and WFA-Adapt, its footprint is also large (100MB at 30% error rate) and increases linearly with the number of differences (and therefore, with both alignment length and error rate) in the sequences being aligned, as shown in Figure 11 and 12. Therefore, TALCO-WFAA provided up to 5.3× improvement in memory footprint over BiWFA as read lengths were increased from 10 Kbp to 100 Kbp (Figure 11). Furthermore, the memory footprint advantage of TALCO-WFAA over BiWFA in the software also increases with error rates, from 2× to 8× as the error rate increases from 1% to 30% (Figure 12). Over Edlib, TALCO-WFAA provides 1.08-1.8× and 1.18-1.9× improvement in memory footprint for PacBio and ONT reads, respectively, with memory savings increasing with read lengths (Figure 11). Edlib's memory performance is attributable to its more restrictive scoring policy (based on Levenshtein distance, Figure 9) that has limited bioinformatic utility. Moreover, TALCO can be used to tile Edlib's algorithm as well (see Section VII).

Overall, TALCO is effective in reducing the memory footprint of existing algorithms, and the benefit increases with the increase in read lengths. This saving in memory footprint per thread from TALCO can also be exploited in software for

higher thread-level parallelism.

Speed of Convergence of Traceback Pointers: Because frontiers in phase 2 of TALCO incur redundant computations since they do not contribute to the partial alignment of the tile and are recomputed in the subsequent tile(s) (resulting in overlapping tiles). To evaluate the speed of convergence of traceback paths, we measured the efficiency of TALCO-XDrop and TALCO-WFAA as the average percentage of frontiers that were computed in a tile in phase 1 of TALCO for 50Kbp reads at different error rates. It is desired that a quick convergence occurs for higher efficiency as it indicates less redundant computation in phase 2. We observed that traceback pointers indeed converged fast, with average efficiencies of 95–97% at low error rates (1-5%) and over 85% in the worst case. This is an improvement over previous tiling approaches since they conservatively set tile overlap for all alignments, typically resulting in 75% or lower efficiency [7], [8], [10].

Accelerator	Component	Configuration	Area (mm^2)	Power (W)
TALCO-XDrop	Logic	32x(32PE)	4.60	12.8
	SRAM	32x(32x4KB + 1KB)	53.7	2.44
	DDR4-2400	4x32GB	-	3.79
	Total		**58.3**	**19.0**
TALCO-WFAA	Logic	16x(16x{Extend,Compute})	17.2	28.1
	SRAM	16x(32KB + 16x2KB + 1KB)	12.8	1.10
	DDR4-2400	4x32GB	-	3.79
	Total		**30.0**	**32.9**

TABLE II: ASIC analysis of TALCO-XDrop and TALCO-WFAA accelerator providing area and power breakdown of the individual components. The critical path delay of TALCO-XDrop and TALCO-WFAA was found to be 2.5ns and 1.8ns, respectively.

ASIC Area, Power, and Frequency Analysis: Table II provides the area and power breakdown for each component of the TALCO-XDrop and TALCO-WFAA accelerators based on our RTL implementations.

For the TALCO-XDrop accelerator, we allowed storing `512`KB wavefronts (equivalent to `512` frontiers) in a single tile in each TALCO-XDrop array as it attained the maximum throughput/watt for all the read types in our experiments. Each array was provisioned with `128`KB of memory for tiling. An additional `1`KB memory was provisioned per array for storing convergence pointers. We synthesized 32 TALCO-XDrop arrays, each with 32 PEs, at 400MHz maximum operating frequency, requiring an area of 58.3mm^2 and consuming 19W of power, including the SRAM and DRAM components, as shown in Table II.

Similarly, we synthesized 16 TALCO-WFAA arrays for ASIC analysis and found a total area and power requirement of 30mm^2 and 32.9W, respectively, for a maximum operating frequency of 556MHz (Table II). We configured the maximum wavefront length = `128` (corresponding to 128 frontiers) for TALCO-WFAA, since we found in our experiments that all wavefronts have $k_{max} - k_{min} < 90$ for all the read types, for chosen scoring parameters.

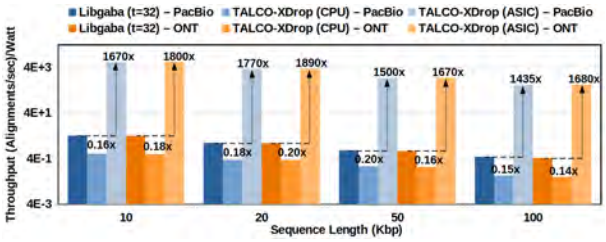

Fig. 13: Comparison of throughput/watt for ASIC and software implementation of TALCO-XDrop compared to the Libgaba algorithm executing on 32 CPU threads as the baseline.

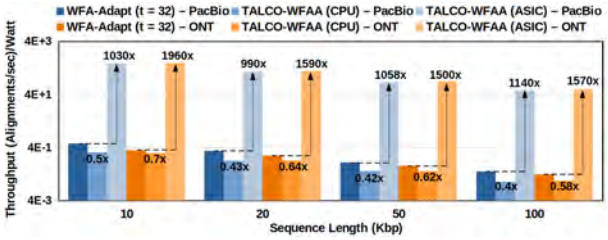

Fig. 14: Comparison of throughput/watt for ASIC and software implementation of TALCO-WFAA compared to the WFA-Adapt algorithm executing on 32 CPU threads as the baseline.

ASIC Performance Comparison with Baselines: Figure 13 compares the alignment throughput per watt of TALCO-XDrop ASIC and software implementation for PacBio and ONT datasets at different sequence lengths with the Libgaba algorithm executing on 32 CPU threads used as the baseline. We set `X=100` since it provided the best alignment scores for Libgaba on all reads in our dataset. TALCO-XDrop was configured to the same X-drop value to produce alignments with matching scores. As Figure 13 shows, TALCO-XDrop ASIC (software) achieved roughly up to 1,900× (0.18×) and 1,800× (0.20×) improvement in throughput per watt over Libgaba for PacBio and ONT reads, respectively.

Similarly, Figure 14 compares the alignment throughput per watt of TALCO-WFAA ASIC and software implementation for PacBio and ONT datasets at different sequence lengths with WFA-Adapt algorithm executing on 32 CPU threads used as the baseline. We used $\texttt{WF}_{\texttt{diff}} = 50$ setting for WFA-Adapt as it produced optimal alignment scores for all reads in our dataset. TALCO-WFAA was tuned for the same $\texttt{W}_{\texttt{diff}}$ value and scoring scheme to produce alignments with matching scores. WFA-Adapt's remaining settings were configured as recommended by the WFA authors [38]. As Figure 14 shows, TALCO-WFAA ASIC (software) achieved roughly up to 1200× (0.4×) and 2000× (0.58×) improvement over WFA-Adapt for PacBio and ONT reads, respectively.

Figure 16 compares the alignment throughput of TALCO-XDrop and TALCO-WFAA ASIC implementation with *all* baselines for PacBio and ONT datasets at different sequence lengths with an error rate of 15%. These results should be viewed in the context of Figure 9 since different tools offer different flexibility in their alignment scoring function, which tends to impact their relative speed and memory performance.

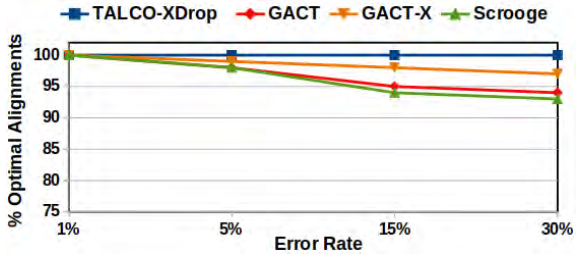

Fig. 15: Percentage of alignments with optimal scores produced by TALCO-XDrop and previous tiling approaches, GACT, GACT-X, and Scrooge, for aligning 50 Kbp PacBio and ONT reads at different error rates. We used linear gap penalties for this experiment to ensure fairness to Scrooge as it does not support affine gap penalties.

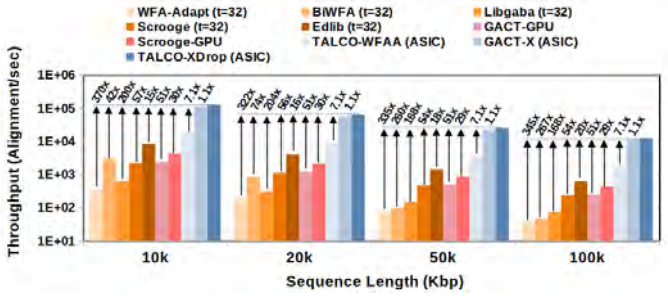

Fig. 16: Comparison of throughput (alignments/sec) for ASIC implementation of TALCO-XDrop and TALCO-WFAA compared to the CPU, GPU, and ASIC implementation of baseline tools for PacBio and ONT datasets with an error rate of 15%.

For a fair comparison, all baselines were configured to obtain the maximum throughput without affecting the alignment score obtained from the recommended default settings. For example, after this tuning, we set GACT (Scrooge) parameters {T,O}={320,120} ({W,O}={64,33}). We note that, unlike TALCO, heuristic tiling approaches (GACT, GACT-X, and Scrooge) did not produce optimal alignment scores in many cases (Figure 15).

As Figure 16 shows, TALCO-XDrop accelerator achieved the highest alignment throughput among all methods, providing 180×, 50×, and 1.1× improvement over CPU, GPU, and ASIC baselines, respectively. Interestingly, TALCO-XDrop provided 1.1× speedup even over GACT-X, which implements a tiling heuristic of X-Drop, while producing higher quality alignments. This can be partially explained by the fact that nearly 37% of cells in GACT-X are redundantly computed in the overlapping regions of the tile, whereas TALCO has lesser redundancy on average as it leverages the fast convergence of pointers.

TALCO-WFAA accelerator also achieved over 65× and 2.2× improvement over CPU and GPU baselines, respectively, though it was found to be 7× (6.3×) slower than TALCO-XDrop (GACT-X) ASIC for this dataset. However, when the error rates are lower, TALCO-WFAA accelerator can provide speedup over TALCO-XDrop (GACT-X). For example, for the read dataset with 1% error rate, we found the TALCO-WFAA accelerator to be 1.1× (1.2×) faster than TALCO-XDrop (GACT-X).

FPGA performance: We could synthesize 32 (16) arrays of TALCO-XDrop (TALCO-WFAA) each with 16 PEs (16 *compute* and 16 *extend* modules) along with AWS shell using Vitis Unified Software Platform on Xilinx Virtex UltraScale Plus (`xcvu9p-flgb2104-2-i`) FPGA, clocked at 119 MHz (166 MHz). This implementation of TALCO-XDrop (TALCO-WFAA) provided a peak throughput of roughly 20,000 (6,000) alignments per second, 6.7× (3.4×) lower than ASIC for ONT and PacBio 10Kbp reads with 15% error rate. In other words, for long alignments, even a single FPGA implementation using TALCO achieves 1-2 orders and 1 order of magnitude improvement in alignment throughput over the software baseline executing on 32 CPU threads and tiling-based GPU baselines, respectively.

VII. Related Work

Algorithms: Dynamic programming (DP) is an immensely popular algorithmic paradigm in the field of bioinformatics. TALCO is a tiling strategy that can be easily applied to the many variants of the classical dynamic programming algorithms for specific pairwise alignment that has been developed for different bioinformatic applications over the years. For example, minimap2 [5], a popular long read aligner, uses a variation of X-drop banding strategy with a two-piece affine gap penalty for scoring alignments, which requires 5 DP matrices, instead of 3 in Gotoh. This alignment can be tiled using TALCO by requiring convergence of traceback pointers on all five matrices instead of just three in TALCO-XDrop presented in this paper.

It is also possible to extend TALCO to some exact pairwise alignment formulations that use banding to improve the average runtime of the search. For example, Ukkonen's algorithm [60] is guaranteed to behave equivalent to a classical dynamic programming algorithm without any banding, but it internally only considers cells within an exponentially increasing band around the main diagonal, resulting in $\mathcal{O}(ns)$ average runtime, where s is the Levenshtein distance between the sequences. This algorithm was further improved in Myers' BitVector algorithm [61], which provided a constant factor improvement in runtime by exploiting bit-level parallelism, and in Edlib [48], which further added traceback support and new alignment options. These algorithms provide large improvements relative to the classical algorithms when the two sequences being aligned are similar, but have a restrictive scoring scheme based on Levenshtein distance.

Some other non-classical algorithms have also recently gained traction for sequence alignment. These include A*PA [62] and Astarix [63], which employ a form of A* algorithm for finding exact global alignments. While these algorithms are not based on Dynamic Programming, they store traceback information, apply banding, and have a frontier (vertices of the priority queue) that is not significantly different from WFA. TALCO can therefore be used to optimally tile these methods. Bitap [64], [65] is another algorithm that uses bit-level parallelism to find the exact alignment of two sequences that have k or fewer differences, where the differences are expressed in terms of Levenshtein distance. GenASM [9]

extended the Bitap algorithm to support traceback and applied a tiling strategy similar to the GACT algorithm to compute long alignments. Similar to GACT, GenASM cannot guarantee optimality even under banding. TALCO can be applied to GenASM to tile alignments while guaranteeing optimal results under banding constraints.

Dynamic Programming has also found applications in bioinformatics far beyond pairwise sequence alignment. These include multiple sequence alignment [66], [67], remote homology search [68], phylogenetics [69], base-calling [70], variant calling [71], and RNA secondary structure prediction [72]. Several of these algorithms can also be tiled with TALCO. For example, CLUSTAL [66] and MUSCLE [67], among the most cited bioinformatic tools, extend classical DP algorithms with position-specific scoring to align multiple sequences. HMMER [68] is a remote homology search tool that uses the Viterbi algorithm on a Hidden Markov Model trained on a multiple alignment of input sequences. Many emerging applications, such as pangenomics [73], [74], utilize dynamic programming for sequence-to-graph [74], [75], and partial-order alignments [76]. The underlying computation structure of many of these algorithms is quite similar to the classical DP algorithms, with both matrix fill and traceback steps to find the optimal alignment, making them amenable to tiling based on the TALCO strategy.

Accelerators: There has been a recent surge in the development of hardware accelerators for genomic workloads utilizing GPUs [10], [24], [77]–[81], FPGAs/ASICs [7]–[10], [22], [43], [82]–[100], and in-memory processing [101]–[111]. In particular, pairwise sequence alignment using dynamic programming has been a key focus in many accelerators [7]–[10], [24], [43], [77]–[79], [83]–[85], [87]–[99]. There has also been significant emphasis on accelerating it in commercial products, such as the FPGA accelerator developed by Illumina and the inclusion of DPX instructions in NVIDIA's latest GPUs [112], [113]. Commercial ASIC accelerators could also be warranted in the future due to the exponential growth of genomic data [114]. Hardware accelerators have traditionally encountered difficulties in maintaining the traceback pointers due to the fact that the memory requirements for traceback in most alignment algorithms and heuristics usually scale linearly or quadratically with the lengths of the sequences. Consequently, it is nearly impossible to store the traceback state for performing long alignments within the limited on-chip memory available on hardware accelerators. Additionally, even though off-chip memory has a higher capacity, it does not offer enough bandwidth to achieve significant acceleration, creating a further hurdle. To circumvent this issue, previous accelerators have employed one of the following four strategies. Firstly, some accelerators only deal with alignment scoring, omitting the traceback functionality, which severely limits their applicability, such as for pre-alignment filtering. Examples of such accelerators include LOGAN [77] and BioSEAL [105]. Secondly, several accelerators have restricted the length of alignments they handle. Accelerators such as GSWABE [78], GenAx [83], SeedEx [84], and ASAP [85] serve as examples of this category. Thirdly, some accelerators, such as FPGA-SW [87], use off-chip memory for storing traceback pointers in case of long alignments, which limits their achievable throughput. Finally, some recent accelerators have applied a heuristic tiling approach for handling alignments of arbitrary lengths, first proposed in the GACT algorithm of the Darwin co-processor [7] and was later adapted in other accelerators, such as GACT-X [8], GenASM [9], Scrooge [10], SeGraM [22], ABSW [23], Darwin-GPU [24] and RAPIDx [25].

One downside of the previous alignment tiling methods has been that they cannot guarantee optimality, which could degrade the quality of alignments in some cases and make it unattainable to preserve parity with non-tiling software methods. For example, authors in [115] tried to accelerate minimap2 on FPGAs by adopting the GACT-X algorithm but observed equivalent alignment in less than 95% of cases. TALCO is a novel tiling approach that can guarantee *optimal* alignments under banding constraints and be applied orthogonally to each of the aforementioned categories of accelerators (CPU, GPU, FPGA, and ASIC). This development could broaden the applicability and lead to wider adoption of hardware accelerators in the bioinformatics community.

VIII. Conclusion and Future Work

In this paper, we presented TALCO, a generalizable tiling strategy for banded sequence alignment algorithms. Unlike previous greedy tiling strategies, which cannot guarantee the quality of results and impose a barrier to the wider adoption of hardware accelerators in real-world applications, TALCO ensures that the alignment quality is not compromised due to tiling. This allows arbitrarily long alignments to be accelerated in hardware accelerators that have a fixed amount of on-chip memory to store the traceback state and achieve orders of magnitude gain in throughput and energy efficiency. TALCO leverages the convergence of traceback paths, which, as we demonstrate, appears to be a general property in both classical and non-classical algorithms for sequence alignment. When implemented in software, tiling with TALCO reduces the peak memory footprint of alignment algorithms by 1-2 orders of magnitude for long reads. We expect TALCO to be beneficial to a broad range of alignment algorithms and accelerators. In the future, we plan to apply TALCO to accelerate and improve the memory footprint of algorithms spanning a wide variety of bioinformatic applications, such as multiple-sequence alignment, sequence-to-graph alignment, and base-calling.

IX. Acknowledgments

We thank Saeed Maleki for the helpful discussions. Research reported in this publication was supported by an Amazon Research Award (Fall 2022 CFP) and funding from the Hellman Fellowship and the U.S. Centers for Disease Control and Prevention through the Office of Advanced Molecular Detection (contract 75D30123C17463).

REFERENCES

[1] Zhenyu Li, Yanxiang Chen, Desheng Mu, Jianying Yuan, Yujian Shi, Hao Zhang, Jun Gan, Nan Li, Xuesong Hu, Binghang Liu, Bicheng Yang, and Wei Fan. Comparison of the two major classes of assembly algorithms: overlap–layout–consensus and de-bruijn-graph. *Briefings in functional genomics*, 11(1):25–37, 2012.

[2] Margaret A Hamburg and Francis S Collins. The path to personalized medicine. *New England Journal of Medicine*, 363(4):301–304, 2010.

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

[4] M Stratton. A comprehensive catalogue of somatic mutations from a human cancer genome. 2010.

[5] Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100, 05 2018.

[6] Sergey Koren, Brian P Walenz, Konstantin Berlin, Jason R Miller, Nicholas H Bergman, and Adam M Phillippy. Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. *Genome research*, 27(5):722–736, 2017.

[7] Yatish Turakhia, Gill Bejerano, and William J Dally. Darwin: a genomics coprocessor. *IEEE Micro*, 39(3):29–37, 2019.

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

[9] Damla Senol Cali, Gurpreet S. Kalsi, Zülal Bingöl, Can Firtina, Lavanya Subramanian, Jeremie S. Kim, Rachata Ausavarungnirun, Mohammed Alser, Juan Gomez-Luna, Amirali Boroumand, Anant Norion, Allison Scibisz, Sreenivas Subramoneyon, Can Alkan, Saugata Ghose, and Onur Mutlu. 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)*, pages 951–966, 2020.

[10] Joël Lindegger, Damla Senol Cali, Mohammed Alser, Juan Gómez-Luna, Nika Mansouri Ghiasi, and Onur Mutlu. Scrooge: A Fast and Memory-Frugal Genomic Sequence Aligner for CPUs, GPUs, and ASICs. *Bioinformatics*, 03 2023. btad151.

[11] Yoshitaka Sakamoto, Sarun Sereewattanawoot, and Ayako Suzuki. A new era of long-read sequencing for cancer genomics. *Journal of human genetics*, 65(1):3–10, 2020.

[12] David Gordon, John Huddleston, Mark JP Chaisson, Christopher M Hill, Zev N Kronenberg, Katherine M Munson, Maika Malig, Archana Raja, Ian Fiddes, LaDeana W Hillier, Christopher Dunn, Carl Baker, Joel Armstrong, Mark Diekhans, Benedict Paten, Jay Shendure, Richard K. Wilson, David Haussler, Chen-Shan Chin, and Evan E. Eichler. Long-read sequence assembly of the gorilla genome. *Science*, 352(6281):aae0344, 2016.

[13] Eric E Schadt, Steve Turner, and Andrew Kasarskis. A window into third-generation sequencing. *Human molecular genetics*, 19(R2):R227–R240, 2010.

[14] Vivien Marx. Method of the year: long-read sequencing. *Nature Methods*, 20(1):6–11, Jan 2023.

[15] John E Gorzynski, Sneha D Goenka, Kishwar Shafin, Tanner D Jensen, Dianna G Fisk, Megan E Grove, Elizabeth Spiteri, Trevor Pesout, Jean Monlong, and Gunjan Baid. Ultrarapid nanopore genome sequencing in a critical care setting. *New England Journal of Medicine*, 386(7):700–702, 2022.

[16] Doruk Beyter, Helga Ingimundardottir, Asmundur Oddsson, Hannes P Eggertsson, Eythor Bjornsson, Hakon Jonsson, Bjarni A Atlason, Snaedis Kristmundsdottir, Svenja Mehringer, Marteinn T Hardarson, Sigurjon A. Gudjonsson, Droplaug N. Magnusdottir, Aslaug Jonasdottir, Adalbjorg Jonasdottir, Ragnar P. Kristjansson, Sverrir T. Sverrisson, Guillaume Holley, Gunnar Palsson, Olafur A. Stefansson, Gudmundur Eyjolfsson, Isleifur Olafsson, Olof Sigurdardottir, Bjarni Torfason, Gisli Masson, Agnar Helgason, Unnur Thorsteinsdottir, Hilma Holm, Daniel F. Gudbjartsson, Patrick Sulem, Olafur T. Magnusson, Bjarni V. Halldorsson, and Kari. Stefansson. Long-read sequencing of 3,622 icelanders provides insight into the role of structural variants in human diseases and other traits. *Nature Genetics*, 53(6):779–786, 2021.

[17] Cory Y McLean, Philip L Reno, Alex A Pollen, Abraham I Bassan, Terence D Capellini, Catherine Guenther, Vahan B Indjeian, Xinhong Lim, Douglas B Menke, Bruce T Schaar, Aaron M. Wenger, Gill Bejerano, and David M. Kingsley. Human-specific loss of regulatory DNA and the evolution of human-specific traits. *Nature*, 471(7337):216–219, 2011.

[18] Chimpanzee Sequencing and Analysis Consortium Waterson Robert H. waterston@ gs. washington. edu Lander Eric S. lander@ broad. mit. edu Wilson Richard K. rwilson@ watson. wustl. edu. Initial sequence of the chimpanzee genome and comparison with the human genome. *Nature*, 437(7055):69–87, 2005.

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

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

[21] Tony Robinson, Jim Harkin, and Priyank Shukla. Hardware acceleration of genomics data analysis: challenges and opportunities. *Bioinformatics*, 37(13):1785–1795, 2021.

[22] Damla Senol Cali, Konstantinos Kanellopoulos, Joël Lindegger, Zülal Bingöl, Gurpreet S. Kalsi, Ziyi Zuo, Can Firtina, Meryem Banu Cavlak, Jeremie Kim, Nika Mansouri Ghiasi, Gagandeep Singh, Juan Gómez-Luna, Nour Almadhoun Alserr, Mohammed Alser, Sreenivas Subramoney, Can Alkan, Saugata Ghose, and Onur Mutlu. 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*, ISCA '22, page 638–655, New York, NY, USA, 2022. Association for Computing Machinery.

[23] Yi-Lun Liao, Yu-Cheng Li, Nae-Chyun Chen, and Yi-Chang Lu. 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)*, pages 1–9, 2018.

[24] Nauman Ahmed, Tong Dong Qiu, Koen Bertels, and Zaid Al-Ars. GPU acceleration of darwin read overlapper for de novo assembly of long DNA reads. *BMC bioinformatics*, 21(13):1–17, 2020.

[25] Weihong Xu, Saransh Gupta, Niema Moshiri, and Tajana Rosing. RAPIDx: High-performance ReRAM processing in-memory accelerator for sequence alignment. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.

[26] Geraldine A Van der Auwera, Mauricio O Carneiro, Christopher Hartl, Ryan Poplin, Guillermo Del Angel, Ami Levy-Moonshine, Tadeusz Jordan, Khalid Shakir, David Roazen, Joel Thibault, Eric Banks, Kiran V. Garimella, David Altshuler, Stacey Gabriel, and Mark A. DePristo. From FastQ data to high-confidence variant calls: the genome analysis toolkit best practices pipeline. *Current protocols in bioinformatics*, 43(1):11–10, 2013.

[27] Arun Subramaniyan, Yufeng Gu, Timothy Dunn, Somnath Paul, Md Vasimuddin, Sanchit Misra, David Blaauw, Satish Narayanasamy, and Reetuparna Das. Genomicsbench: A benchmark suite for genomics. In *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 1–12, 2021.

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

[29] Mohammed Alser, Hasan Hassan, Hongyi Xin, Oğuz Ergin, Onur Mutlu, and Can Alkan. GateKeeper: a new hardware architecture for accelerating pre-alignment in DNA short read mapping. *Bioinformatics*, 33(21):3355–3363, 05 2017.

[30] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), page 483–485, New York, NY, USA, 1967. Association for Computing Machinery.

[31] Martin C Frith, Michiaki Hamada, and Paul Horton. Parameters for accurate genome alignment. *BMC bioinformatics*, 11:1–14, 2010.

[32] Zheng Zhang, Piotr Berman, Thomas Wiehe, and Webb Miller. Post-processing long pairwise alignments . *Bioinformatics*, 15(12):1012–1019, 12 1999.

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

[34] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers,

and David J Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

[35] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710. Soviet Union, 1966.

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

[37] Eugene W Myers. An o (nd) difference algorithm and its variations. *Algorithmica*, 1(1-4):251–266, 1986.

[38] Santiago Marco-Sola, Juan Carlos Moure, Miquel Moreto, and Antonio Espinosa. Fast gap-affine pairwise alignment using the wavefront algorithm. *Bioinformatics*, 37(4):456–463, 09 2020.

[39] Jordan M. Eizenga and Benedict Paten. Improving the time and space complexity of the wfa algorithm and generalizing its scoring. *bioRxiv*, 2022.

[40] Santiago Marco-Sola, Jordan M Eizenga, Andrea Guarracino, Benedict Paten, Erik Garrison, and Miquel Moreto. Optimal gap-affine alignment in O(s) space. *Bioinformatics*, 39(2):btad074, 02 2023.

[41] Zheng Zhang, Scott Schwartz, Lukas Wagner, and Webb Miller. A greedy algorithm for aligning dna sequences. *Journal of Computational biology*, 7(1-2):203–214, 2000.

[42] Saeed Maleki, Madanlal Musuvathi, and Todd Mytkowicz. Parallelizing dynamic programming through rank convergence. *ACM SIGPLAN Notices*, 49(8):219–232, 2014.

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

[44] Yukiteru Ono, Kiyoshi Asai, and Michiaki Hamada. Pbsim2: a simulator for long-read sequencers with a novel generative model of quality scores. *Bioinformatics*, 37(5):589–595, 2021.

[45] Anthony Rhoads and Kin Fai Au. Pacbio sequencing and its applications. *Genomics, Proteomics Bioinformatics*, 13(5):278–289, 2015. SI: Metagenomics of Marine Environments.

[46] Ivan Sović, Mile Šikić, Andreas Wilm, Shannon Nicole Fenlon, Swaine Chen, and Niranjan Nagarajan. Fast and sensitive mapping of nanopore sequencing reads with GraphMap. *Nature communications*, 7(1):11307, 2016.

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

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

[49] Rohit Chandra, Leo Dagum, Ramesh Menon, David Kohr, Dror Maydan, and Jeff McDonald. *Parallel programming in OpenMP*. Morgan kaufmann, 2001.

[50] Philipp Wendler and Dirk Beyer. sosy-lab/benchexec: Release 3.16, February 2023.

[51] Nauman Ahmed, Tong Dong Qiu, Koen Bertels, and Zaid Al-Ars. Gpu acceleration of darwin read overlapper for de novo assembly of long dna reads. *BMC bioinformatics*, 21(13):1–17, 2020.

[52] NVIDIA. Nvidia rtx a6000 datasheet, 2020.

[53] NVIDIA. Nsight compute metrics guide, 2020.

[54] Tutu Ajayi, Vidya A. Chhabria, Mateus Fogaça, Soheil Hashemi, Abdelrahman Hosny, Andrew B. Kahng, Minsoo Kim, Jeongsup Lee, Uday Mallappa, Marina Neseem, Geraldo Pradipta, Sherief Reda, Mehdi Saligane, Sachin S. Sapatnekar, Carl Sechen, Mohamed Shalan, William Swartz, Lutong Wang, Zhehong Wang, Mingyu Woo, and Bangqi Xu. Toward an open-source digital flow: First learnings from the openroad project. In *Proceedings of the 56th Annual Design Automation Conference 2019*, DAC '19, New York, NY, USA, 2019. Association for Computing Machinery.

[55] Matthew R. Guthaus, James E. Stine, Samira Ataei, Brian Chen, Bin Wu, and Mehedi Sarwar. Openram: An open-source memory compiler. In *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–6, 2016.

[56] James E. Stine, Ivan Castellanos, Michael Wood, Jeff Henson, Fred Love, W. Rhett Davis, Paul D. Franzon, Michael Bucher, Sunil Basavarajaiah, Julie Oh, and Ravi Jenkal. Freepdk: An open-source variation-aware design kit. In *2007 IEEE International Conference on Microelectronic Systems Education (MSE'07)*, pages 173–174, 2007.

[57] Yonghui Li Sven Goossens Matthias Jung Omar Naji Benny Akesson Norbert Wehn Karthik Chandrasekar, Christian Weis and Kees Goossens. Drampower: Open-source dram power energy estimation tool.

[58] Ultrascale architecture and product data sheet: Overview, Nov 2022.

[59] Amazon Web Service. Overview of aws ec2 fpga development kit. https://github.com/aws/aws-fpga.

[60] Esko Ukkonen. Algorithms for approximate string matching. *Information and Control*, 64(1):100–118, 1985. International Conference on Foundations of Computation Theory.

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

[62] Ragnar Groot Koerkamp and Pesho Ivanov. Exact global alignment using a* with seed heuristic and match pruning. *bioRxiv*, page 92, 2022.

[63] Pesho Ivanov, Benjamin Bichsel, Harun Mustafa, André Kahles, Gunnar Rätsch, and Martin Vechev. AStarix: Fast and optimal sequence-to-graph alignment. In Russell Schwartz, editor, *Research in Computational Molecular Biology*, pages 104–119, Cham, 2020. Springer International Publishing.

[64] Ricardo Baeza-Yates and Gaston H. Gonnet. A new approach to text searching. *Commun. ACM*, 35(10):74–82, oct 1992.

[65] Sun Wu and Udi Manber. Fast text searching: Allowing errors. *Commun. ACM*, 35(10):83–91, oct 1992.

[66] Fabian Sievers, Andreas Wilm, David Dineen, Toby J Gibson, Kevin Karplus, Weizhong Li, Rodrigo Lopez, Hamish McWilliam, Michael Remmert, Johannes Söding, Julie D Thompson, and Desmond G Higgins. Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega. *Molecular Systems Biology*, 7(1):539, 2011.

[67] Robert C Edgar. MUSCLE: a multiple sequence alignment method with reduced time and space complexity. *BMC bioinformatics*, 5(1):1–19, 2004.

[68] S R Eddy. Profile hidden Markov models. *Bioinformatics*, 14(9):755–763, 01 1998.

[69] David Sankoff. Minimal mutation trees of sequences. *SIAM Journal on Applied Mathematics*, 28(1):35–42, 1975.

[70] Jared T Simpson, Rachael E Workman, PC Zuzarte, Matei David, LJ Dursi, and Winston Timp. Detecting dna cytosine methylation using nanopore sequencing. *Nature methods*, 14(4):407–410, 2017.

[71] Rasmus Nielsen, Thorfinn Korneliussen, Anders Albrechtsen, Yingrui Li, and Jun Wang. Snp calling, genotype calling, and sample allele frequency estimation from new-generation sequencing data. 2012.

[72] Tatsuya Akutsu. Dynamic programming algorithms for rna secondary structure prediction with pseudoknots. *Discrete Applied Mathematics*, 104(1-3):45–62, 2000.

[73] Nicholas Noll, Marco Molari, Liam P Shaw, and Richard A Neher. Pangraph: scalable bacterial pan-genome graph construction. *bioRxiv*, pages 2022–02, 2022.

[74] Glenn Hickey, David Heller, Jean Monlong, Jonas A Sibbesen, Jouni Sirén, Jordan Eizenga, Eric T Dawson, Erik Garrison, Adam M Novak, and Benedict Paten. Genotyping structural variants in pangenome graphs using the vg toolkit. *Genome biology*, 21:1–17, 2020.

[75] Heng Li, Xiaowen Feng, and Chong Chu. The design and construction of reference pangenome graphs with minigraph. *Genome biology*, 21:1–19, 2020.

[76] Yan Gao, Yongzhuang Liu, Yanmei Ma, Bo Liu, Yadong Wang, and Yi Xing. abPOA: an SIMD-based C library for fast partial order alignment using adaptive band. *Bioinformatics*, 37(15):2209–2211, 2021.

[77] Alberto Zeni, Giulia Guidi, Marquita Ellis, Nan Ding, Marco D. Santambrogio, Steven Hofmeyr, Aydın Buluç, Leonid Oliker, and Katherine Yelick. LOGAN: High-performance GPU-Based X-Drop long-read alignment. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 462–471, 2020.

[78] Yongchao Liu and Bertil Schmidt. GSWABE: Faster gpu-accelerated sequence alignment with optimal alignment retrieval for short dna sequences. *Concurr. Comput.: Pract. Exper.*, 27(4):958–972, mar 2015.

[79] Nauman Ahmed, Jonathan Lévy, Shanshan Ren, Hamid Mushtaq, Koen Bertels, and Zaid Al-Ars. Gasal2: a gpu accelerated sequence alignment library for high-throughput ngs data. *BMC bioinformatics*, 20:1–20, 2019.

[80] Sneha D. Goenka, Yatish Turakhia, Benedict Paten, and Mark Horowitz. Segalign: A scalable gpu-based whole genome aligner. In *SC20:*

International Conference for High Performance Computing, Networking, Storage and Analysis, pages 1–13, 2020.

[81] Jacopo Pantaleoni and Nuno Subtil. Nvbio.

[82] Mohammed Alser, Taha Shahroodi, Juan Gómez-Luna, Can Alkan, and Onur Mutlu. SneakySnake: a fast and accurate universal genome pre-alignment filter for CPUs, GPUs and FPGAs. *Bioinformatics*, 36(22-23):5282–5290, 12 2020.

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

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

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

[86] Arun Subramaniyan, Jack Wadden, Kush Goliya, Nathan Ozog, Xiao Wu, Satish Narayanasamy, David Blaauw, and Reetuparna Das. Accelerated seeding for genome sequence alignment with enumerated radix trees. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 388–401, 2021.

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

[88] Khaled Benkrid, Ying Liu, and AbdSamad Benkrid. A highly parameterized and efficient FPGA-Based skeleton for pairwise biological sequence alignment. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 17(4):561–570, 2009.

[89] Konstantina Koliogeorgi, Nils Voss, Sotiria Fytraki, Sotirios Xydis, Georgi Gaydadjiev, and Dimitrios Soudris. Dataflow acceleration of Smith-Waterman with traceback for high throughput next generation sequencing. In *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, pages 74–80, 2019.

[90] Jeff Allred, Jack Coyne, William Lynch, Vincent Natoli, Joseph Grecco, and Joel Morrissette. Smith-Waterman implementation on a FSB-FPGA module using the intel accelerator abstraction layer. In *2009 IEEE International Symposium on Parallel Distributed Processing*, pages 1–4, 2009.

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

[92] J. M. Marmolejo-Tejada, V. Trujillo-Olaya, C. P. Rentería-Mejía, and J. Velasco-Medina. Hardware implementation of the Smith-Waterman algorithm using a systolic architecture. In *2014 IEEE 5th Latin American Symposium on Circuits and Systems*, pages 1–4, 2014.

[93] Ho-Cheung Ng, Shuanglong Liu, Izaak Coleman, Ringo S.W. Chu, Man-Chung Yue, and Wayne Luk. Acceleration of short read alignment with runtime reconfiguration. In *2020 International Conference on Field-Programmable Technology (ICFPT)*, pages 256–262, 2020.

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

[95] Lorenzo Di Tucci, Kenneth O'Brien, Michaela Blott, and Marco D. Santambrogio. Architectural optimizations for high performance and energy efficient Smith-Waterman implementation on FPGAs using OpenCL. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 716–721, 2017.

[96] D. S. Nurdin, M. N. Isa, and S. H. Goh. DNA sequence alignment: A review of hardware accelerators and a new core architecture. In *2016 3rd International Conference on Electronic Design (ICED)*, pages 264–268, 2016.

[97] K. Puttegowda, W. Worek, N. Pappas, A. Dandapani, P. Athanas, and A. Dickerman. A run-time reconfigurable system for gene-sequence searching. In *16th International Conference on VLSI Design, 2003. Proceedings.*, pages 561–566, 2003.

[98] Jing-Ping Wu, Yi-Chien Lin, Ying-Wei Wu, Shih-Wei Hsieh, Ching-Hsuan Tai, and Yi-Chang Lu. A memory-efficient accelerator for dna sequence alignment with two-piece affine gap tracebacks. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4, 2021.

[99] Ernst Joachim Houtgast, Vlad-Mihai Sima, Koen Bertels, and Zaid Al-Ars. An FPGA-based systolic array to accelerate the BWA-MEM genomic mapping algorithm. In *2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, pages 221–227, 2015.

[100] Tae Jun Ham, David Bruns-Smith, Brendan Sweeney, Yejin Lee, Seong Hoon Seo, U Gyeong Song, Young H Oh, Krste Asanovic, Jae W Lee, and Lisa Wu Wills. Genesis: A hardware acceleration framework for genomic data analysis. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 254–267. IEEE, 2020.

[101] Robert Hanhan, Esteban Garzón, Zuher Jahshan, Adam Teman, Marco Lanuzza, and Leonid Yavits. Edam: Edit distance tolerant approximate matching content addressable memory. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ISCA '22, page 495–507, New York, NY, USA, 2022. Association for Computing Machinery.

[102] Anirban Nag, C. N. Ramachandra, Rajeev Balasubramonian, Ryan Stutsman, Edouard Giacomin, Hari Kambalasubramanyam, and Pierre-Emmanuel Gaillardon. GenCache: Leveraging in-cache operators for efficient sequence alignment. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '52, page 334–346, New York, NY, USA, 2019. Association for Computing Machinery.

[103] H. Mao, M. Alser, M. Sadrosadati, C. Firtina, A. Baranwal, D. Cali, A. Manglik, N. Alserr, and O. Mutlu. 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)*, pages 710–726, Los Alamitos, CA, USA, oct 2022. IEEE Computer Society.

[104] Wenqin Huangfu, Shuangchen Li, Xing Hu, and Yuan Xie. RADAR: A 3D-ReRAM based dna alignment accelerator architecture. In *Proceedings of the 55th Annual Design Automation Conference*, DAC '18, New York, NY, USA, 2018. Association for Computing Machinery.

[105] Roman Kaplan, Leonid Yavits, and Ran Ginosasr. Bioseal: In-memory biological sequence alignment accelerator for large-scale genomic data. In *Proceedings of the 13th ACM International Systems and Storage Conference*, SYSTOR '20, page 36–48, New York, NY, USA, 2020. Association for Computing Machinery.

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

[107] Farzaneh Zokaee, Hamid R. Zarandi, and Lei Jiang. AligneR: A process-in-memory architecture for short read alignment in ReRAMs. *IEEE Computer Architecture Letters*, 17(2):237–240, 2018.

[108] Roman Kaplan, Leonid Yavits, Ran Ginosar, and Uri Weiser. A resistive CAM processing-in-storage architecture for DNA sequence alignment. *IEEE Micro*, 37(4):20–28, 2017.

[109] Wenqin Huangfu, Shuangchen Li, Xing Hu, and Yuan Xie. RADAR: A 3D-ReRAM based dna alignment accelerator architecture. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6, 2018.

[110] Zamshed I. Chowdhury, Masoud Zabihi, S. Karen Khatamifard, Zhengyang Zhao, Salonik Resch, Meisam Razaviyayn, Jian-Ping Wang, Sachin S. Sapatnekar, and Ulya R. Karpuzcu. A DNA read alignment accelerator based on computational RAM. *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, 6(1):80–88, 2020.

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

[112] Pieter Van Rooyen, Robert J McMillen, and Michael Ruehle. Bioinformatics systems, apparatuses, and methods executed on an integrated circuit processing platform, June 13 2017. US Patent 9,679,104.

[113] Anne C Elster and Tor A Haugdahl. Nvidia hopper gpu and grace cpu highlights. *Computing in Science & Engineering*, 24(2):95–100, 2022.
[114] William J Dally, Yatish Turakhia, and Song Han. Domain-specific hardware accelerators. *Communications of the ACM*, 63(7):48–57, 2020.
[115] Carolina Teng, Renan Weege Achjian, Jiang Chau Wang, and Fernando Josepetti Fonseca. Adapting the GACT-X aligner to accelerate Minimap2 in an FPGA cloud instance. *Applied Sciences*, 13(7), 2023.

Artifact Appendix

A. Abstract

Here we briefly describe how to reproduce the main results of this paper. The instructions cover: 1) the installation of dependencies, 2) the process for downloading the dataset, 3) steps for building TALCO-XDrop, TALCO-WFAA, and all baseline components, and 4) methods for conducting software and ASIC analysis. The source code and instructions can be accessed from GitHub (https://github.com/TurakhiaLab/TALCO) or the Zenodo archive (https://zenodo.org/records/10306077).

B. Artifact check-list (meta-information)

- **Algorithm:** Sequence Alignment Algorithms: Needleman-Wunsch, X-Drop, WFA-Adapt, Edlib, and GACT-X
- **Program:** `Docker, C++/C`
- **Compilation:** `g++-11.4, gcc-11.4`
- **Dataset:** Simulated reads from PBSIM2, resembling error profiles of Pacific Biosciences (PacBio) and Oxford Nanopore Technology (ONT)
- **Hardware:** Intel CPU and NVIDIA GPU
- **Metrics:** Memory Usage, Alignment Throughput/Watt, and Alignment Throughput
- **Output:** Experiments produce outputs in the console or log files.
- **Experiments:** a) Memory Usage for TALCO-XDrop, TALCO-WFAA (software), and CPU baselines, b) ASIC analysis for TALCO-XDrop and TALCO-WFAA, c) CPU/GPU/ASIC baselines for throughput analysis
- **Memory space requirement:** 20GB
- **Disk space requirement:** 50GB
- **Time needed to complete the experiments:** 10 hours
- **Publicly available:** Yes
- **Code licenses:** MIT
- **Data licenses:** MIT
- **Archive DOI:** 10.5281/zenodo.10306077

C. Description

1) **How to access**: The codebase can be accessed from GitHub (https://github.com/TurakhiaLab/TALCO) or the Zenodo archive (https://zenodo.org/records/10306077).
2) **Hardware Dependencies**:
   - Intel CPU and NVIDIA GPU
   - 20GB memory and 50GB storage
3) **Software Dependencies**:
   - `Linux OS`
   - `gcc >= 10.5.0`
   - `cmake >= 3.16.3`
   - `CUDA >= 10.0`
   - `Python >= 3.8.0`
   - `Python3-pip`

D. Installation

Use the following commands to clone the TALCO repository, install the necessary tools, download the dataset, and build TALCO-XDrop, TALCO-WFAA, and software baselines.

```
$Host: git clone --recursive
    https://github.com/TurakhiaLab/TALCO.git
$Host: cd TALCO/software/scripts
$Host: sudo ./install_dependencies.sh
$Host: source setup_dataset.sh
$Host: source build_baseline.sh
$Host: source build_TALCO.sh
```

E. Experiment workflow

The results presented in this paper can be reproduced using the following instructions:

1) Generate the memory footprint of single-threaded execution of TALCO-XDrop (software), TALCO-WFAA (software), and baselines (Libgaba, WFA-Adapt, and BiWFA) using the following command.

```
$Host: ./analysis.sh mem
```

2) Alignment throughput of all software baselines, executed on 32 CPU threads, can be generated using the following command.

```
$Host: ./analysis.sh thp
```

3) Alignment throughput/watt of all the software baselines, executed on 32 CPU threads, can be generated using the following command. Please note that this analysis cannot be performed on AWS/VM/Docker as Benchexec does not work with them and is also described at https://github.com/TurakhiaLab/TALCO/tree/main/software.

```
$Host: ./analysis.sh thp/w
```

ASIC Analysis

1) We provide a pre-built docker image (`swalia14/talco:latest`) with all necessary tools installed in it for ASIC evaluation. Use the following command to download and run the docker image.

```
$Host: docker run -it
    swalia14/talco:latest
```

2) ASIC analysis of our designs (TALCO-XDrop and TALCO-WFAA) is performed using the following commands (inside the docker container):

```
$Docker: cd /TALCO/hardware/scripts
# Area, Power, and Critical path delay of
    designs
$Docker: source ASIC_analysis.sh
    [XDrop/WFAA]
# SRAM area and power
$Docker: source SRAM_analysis.sh
# DRAM power and cycle count
```

```
$Docker: source DRAM_analysis.sh
```

F. Evaluation and Expected results

- **Memory Footprint**: The goal of this experiment is to demonstrate that the TALCO tiling strategy improves the memory footprint of software aligners without affecting the alignment results. To quantify the benefits, Software Analysis Step 1 reproduces the results in Figure 10 and 11.
- **ASIC Analysis**: We perform ASIC analysis of our designs using OpenROAD with OpenRAM, and use DRAMPower for DRAM analysis. Table II is reproduced in ASIC Analysis Step 2.
- **Throughput**: This experiment compares the alignment throughput of TALCO-XDrop and TALCO-WFAA ASIC implementation with all baselines. Figure 16 is reproduced in Software Analysis Step 2 and ASIC Analysis Step 2.
- **Throughput/Watt**: This experiment compares the alignment throughput per watt of TALCO-XDrop (TALCO-WFAA) with the Libgaba (WFA-Adapt) algorithm executing on 32 CPU threads used as the baseline. ASIC Analysis Step 2 and Software Analysis Step 3 reproduce the results in Figure 13 and 14.

G. Notes

1) We recommend using the docker image we provided to reproduce the ASIC results presented in the paper.
2) We performed the CPU and GPU analysis on a 64-core Intel Xeon Silver 4216 processor and NVIDIA RTX A6000 GPU, respectively, and the results may vary with other processors.

H. Methodology

Submission, reviewing, and badging methodology:

- https://www.acm.org/publications/policies/artifact-review-and-badging-current
- http://cTuning.org/ae/submission-20201122.html
- http://cTuning.org/ae/reviewing-20201122.html