Dataflow acceleration of Smith-Waterman with Traceback for high throughput Next Generation Sequencing

K.Koliogeorgi*, N.Voss†, S.Fytraki†, S.Xydis*, G.Gaydadjiev†, D.Soudris*
**Microprocessors and Digital Systems Laboratory, ECE , NTUA, Greece*,†*Maxeler Technologies, UK*
*{konstantina, sxydis, dsoudris}@microlab.ntua.gr,†{nvoss, sfytraki, georgi}@maxeler.com

***Abstract*—Smith-Waterman algorithm is widely adopted by most popular DNA sequence aligners. The inherent algorithm computational intensity and the vast amount of NGS input data it operates on, create a bottleneck in genomic analysis flows for short-read alignment. FPGA architectures have been extensively leveraged to alleviate the problem, each one adopting a different approach. In existing solutions, effective co-design of the NGS short-read alignment still remains an open issue, mainly due to narrow view on real integration aspects, such as system wide communication and accelerator call overheads. In this paper, we propose a dataflow architecture for Smith-Waterman Matrix-fill and Traceback alignment stages, to perform short-read alignment on NGS data. The architectural decision of moving both stages on chip extinguishes the communication overhead, and coupled with radical software restructuring, allows for efficient integration into widely-used Bowtie2 aligner. This approach delivers $\times$18 speedup over the respective Bowtie2 standalone components, while our co-designed Bowtie2 demonstrates a 35% boost in performance.**

***Index Terms*—Next Generation Sequencing, Reconfigurable Acceleration, Dataflow Computing, Bowtie2, Smith Waterman, Traceback**

I. Introduction

The development of next-generation sequencing (NGS) technologies has dramatically changed the landscape of human genetics research [1]. Advances in the field of DNA and RNA sequencing have led to effective genome mapping and have paved the way to personalized genomic medicine [2]. NGS platforms have now the capacity to generate billions of short fragments of DNA in a matter of hours. These small pieces of DNA, called reads, are the input to various types of genomic analysis such as variant calling [3] and differential gene expression [4]. The first step in any genomic analysis pipeline however is short read alignment, which entails finding a specific location on the reference human genome where a short read is best mapped. The vast amount of sequencing data and the excessive time requirements for this step to execute, have put considerable strain on the computing systems used for genome analysis. Since the throughput of NGS technologies does not cease its exponential growth [5], there is an ever-present need for identifying bottlenecks and proposing accelerated solutions for popular aligner tools.

Several aligners [6], [7] have been developed that rely on a seed-and-extend model for aligning the short reads. According to this model, in the seeding step, each short read is further fragmented in short pieces, called seeds, that align exactly on the reference genome. In the seed-extension step each seed is extended so that the whole short read aligns with the reference, allowing mismatches. Most of the state-of-the-art aligners implement variations of the Smith-Waterman [8] string matching algorithm to perform the seed-extension step. Smith-Waterman is a dynamic programming algorithm that operates in two stages; the *matrix-fill* stage fills a two-dimensional similarity matrix with score values. Starting at a predefined matrix cell, the *traceback* stage traverses the matrix backwards until it constructs a valid alignment path.

A profiling based on a 10 million short-read input dataset of length 100bp, that was collected as part of EU healthcare project AEGLE [9], indicates that Smith-Waterman dominates the execution time of Bowtie2 aligner [6] by a percentage of 60%. However, the histogram in Fig.1 showcases that this time is actually shared among independent Smith-Waterman tasks distributed across all reads. Each read alignment can invoke from one up to 270 Smith-Waterman *matrix-fill* tasks, each one followed by a *traceback* task. All individual *matrix-fill* tasks add up to the 56% of total execution time and *traceback* 4%. An initial naive approach would target this stage to employ hardware acceleration and tackle the alignment bottleneck. A straightforward integration of an accelerated *matrix-fill* phase of Smith-Waterman though [10], [11], [12] would introduce a huge overhead, due to both the immense amount of the accelerator calls and the transferring of the matrices to the CPU for the traceback stage. In fact, taking into account the time overhead provisioned by each call to the accelerator and the accelerator-CPU transfer time for each matrix, the overall execution time of the aligner can actually be increased. A challenge as such has also been noted by [13] regarding JVM-FPGA communication overhead.

Most existing works either propose standalone matrix-fill Smith-Waterman acceleration ignoring the traceback stage [10]–[12] and thus the communication overhead in a real system, or provide an end-to-end hardware implementation of both seed and extend phases [14], [15]. The latter architectural decision introduces immense memory requirements, to support storing the human genome on chip. Moreover, such systems constitute new tools that introduce a learning curve for biologists and come at the expense of safe-to-use results and advanced visualization analyses provided by well-known and defacto sequencing frameworks such as Bowtie2 [6], BWA [7]. There are only a few software/hardware co-design acceleration works for short-read alignment [16], [17]. Therefore, effective co-design of the NGS short-read alignment still remains an open issue, mainly due to narrow view on real integration

This work has been partially funded by EU Horizon 2020 program under grant agreement No 825061 EVOLVE (https://www.evolve-h2020.eu/)

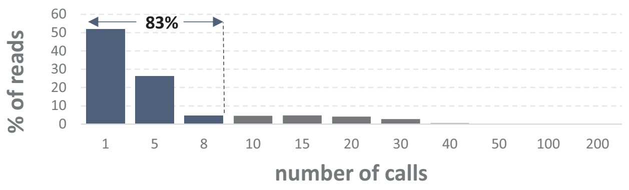

Fig. 1: Distribution of Matrix-Fill Function Calls Across Reads in Bowtie2.

provided by existing solutions. Co-design of NGS alignment exposes several challenges that can be only highlighted by holistically and carefully profiling a sequencer and modeling the behavior of a co-designed version.

In this paper, we design, implement and explore a novel high performance reconfigurable accelerator for Smith-Waterman matrix-fill and traceback stages, that enables integration into a real system. The contribution of this paper can be briefly summarized as follows: (i) through profiling and modeling Bowtie2 we identify the communication overhead that a traditional co-design approach introduces, and propose an architecture that alleviates this problem by moving more computation on chip and restructuring the software code to minimize accelerator calls. (ii) we provide a dataflow implementation of matrix-fill and traceback components, that operate in a complete pipelined manner and can process an unbound number of streaming short reads to cope with the exponential growth in NGS data (iii) we exploit the inherent data dependencies within a single alignment task and we leverage task-level parallelism through an interleaving execution pattern that maximizes the throughput via high utilization of the underline FPGA resources. Experimental results show that the standalone accelerator is $\times 18$ faster than Bowtie2 SIMD implementation. (iv) we present a Bowtie2 code restructuring that implements an aggregation-batching strategy, arranges data in an interleaved pattern and feeds the dataflow accelerator in high-throughput streaming fashion. To the best of our knowledge, this is the first work that integrates a hardware accelerator into the original Bowtie2 rather than building an equivalent aligner from scratch. An integration of the accelerator with the restructured Bowtie2 source code manages to deliver 35% performance gain.

The remainder of the paper is organized as follows. A short literature review on accelerating Smith-Waterman in SectionII precedes the theoretical background of Smith-Waterman (Section III) and the architecture description in Section IV. Section V provides preliminary results and insights on integrating the accelerator with Bowtie2. Section VI provides a performance evaluation of the architecture and integration. Lastly, Section VII concludes the paper.

II. Related Work

A very extended survey, that can also serve as a guide of available implementations for genetic sequence alignment based on Smith-Waterman can be found in [18]. Most Smith-Waterman accelerators [19], [10], [11], [12] implement the first phase of computing the similarity matrix based on a wavefront approach. Their architecture consists of a pipeline of PEs that forms a systolic array and is mapped to the computation of a matrix anti-diagonal per time step. The authors in [12] provide a very detailed architecture as such, that implements a multistage-PE design, and optimize each stage in terms of resources utilization and delay. Similarly in [20], the authors propose a reconfigurable accelerator that implements a modified equation to improve mapping efficiency of a single PE, and a special floor plan to cut down the interface components routing delay. In this paper, we aim to extend these designs by implementing the complete Smith-Waterman algorithm along with the Traceback procedure. In our final high-throughput real system, on-chip traceback diminishes matrix transfer overhead cost and thus enables efficient integration, without adding extra latency thanks to the pipelined scheduling of consecutive tasks.

There are only a few works of accelerated sequence alignment based on Smith-Waterman with Traceback. The authors in [21] propose a space efficient, global sequence alignment architecture that accelerates both the forward scan and traceback for variable reference lengths. The traceback procedure is implemented in software and the host initiates a full traceback by request. The authors in [22] similarly employ a pipeline of PEs to calculate the similarity matrix and traceback. In order to address memory issues that arise when saving the similarity matrix for traceback, they propose to identify the read-database alignments with the highest maximum match score and recalculate the matrices for traceback in software. Although these works provide the traceback functionality under certain circumstances, they are not designed to cope with a high throughput input rate of short reads generated by an NGS platform. In this work, we attempt to provide an accelerator that continuously accepts new short-reads for alignment and complies with the trends set by latest NGS platforms and state-of-the-art sequencers. Recently, Darwin [23] has proposed an end-to-end hardware acceleration for 3rd generation sequencing, implementing accelerators for both seed extract and extend phases and highlighting the importance of including the traceback step on chip so as not to undermine the benefits of hardware acceleration. However our two solutions are not directly comparable, since 3rd generation [24] sequencing highly differs algorithmically.

III. Theoretical Background

A. *Smith-Waterman Algorithm*

Smith-Waterman [8] (SmW) is a dynamic programming algorithm for performing local sequence alignment and determining similar regions between two nucleic sequences. The algorithm mainly consists of two phases: (i) filling a similarity matrix and (ii) tracing back the similarity matrix to find the optimal alignment between the two sequences. Let Q, $|Q| = n$ be the read sequence that aligns against reference sequence S, $|S| = m$. Q and S constitute a read-reference pair and alignment task. SmW performs alignment by filling similarity matrix H according to Eq.1. In this equation, q and r stand for gap extend and gap open penalties respectively, while sc stands for the substitution matrix that assigns each pair of bases a score for match or mismatch.

$$\begin{aligned} E_{i,j} &= \max\{E_{i-1,j}, H_{i-1,j} - q\} - r \\ F_{i,j} &= \max\{F_{i,j-1}, H_{i,j-1} - q\} - r \\ H_{i,j} &= \max\{H_{i-1,j-1} + sc[Q[i], S[j]], E_{i,j}, F_{i,j}, 0\} \end{aligned} \tag{1}$$

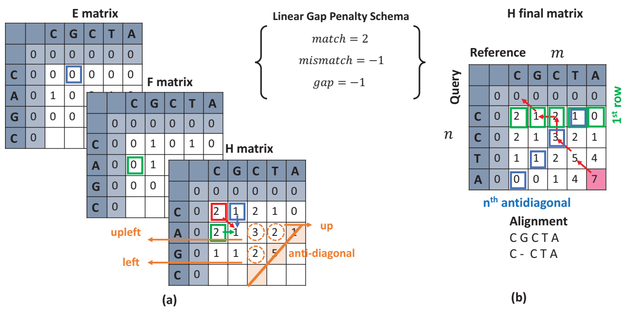

Fig. 2: Matrix Fill Dependencies and Traceback example for simplified Smith-Waterman with linear gap penalty scheme.

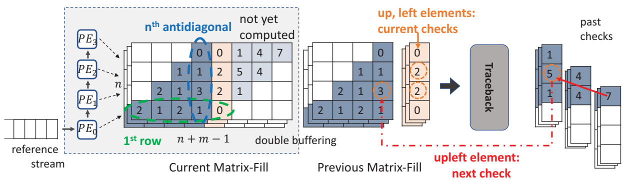

Fig. 3: Flow of data from Matrix Fill to Traceback phase.

SmW then identifies the highest score in H matrix. Starting at this position, a traceback function traverses the matrices backwards until it reaches a zero-score element and thus acquires the optimal alignment path.

IV. Dataflow Smith-Waterman & Traceback Engine

The proposed implementation of SmW is based on the algorithmic version utilized by Bowtie2. Bowtie2 implements a variation of SW, that applies a heuristic in order to overcome data dependencies and boost performance through SSE2 instructions. We deliver a pure SmW implementation but adhere to the scoring scheme, alphabet $\{A, C, G, T, N\}$ and initialization conditions utilized in Bowtie2. The proposed architecture includes two dataflow modules, each one implementing a phase of SmW, the *Matrix-Fill* and *Traceback* components.

A. Matrix Fill

Matrix-fill kernel receives as input the read-reference pairs to be aligned and calculates the matrices H, F, E and the starting point of the traceback procedure. A control input stream indicates the beginning of each new read-reference pair. The substitution matrix that defines the match and mismatch penalties is implemented as a Read-Only Memory on chip. The $Traceback$ kernel needs to traverse the data generated by Matrix-Fill in reverse order, starting from the aforementioned position. For that reason, the matrices are stored on on-chip memory for subsequent use from Traceback.

Fig.2 illustrates an alignment example utilizing a simplified linear gap penalty scheme. Fig.2(a) presents an intermediate snapshot of the calculations whereas Fig.2(b) the final results. Each cell in the matrix H requires the values from the up cells of matrices E, F, H (indicated by blue boxes), the $left$ cells of E, F, H (green boxes) and the $upleft$ cell (red box) of H. Therefore all elements in a single anti-diagonal can be computed in parallel and are dependent only on values from the previous two anti-diagonals. The proposed architecture exploits this property to extract parallelism and thus fills H,F,E matrices per anti-diagonal in n + m -1 steps.

The architecture is built on an *array of PEs*, equal to the length of the read sequence. Each PE holds a single base character of the read sequence, and is in charge of computing its respective row in the H matrix. The PEs operate in parallel and compute an antidiagonal of the original matrix per time step. Due to this anti-diagonal scheduling of computations, the score matrices are filled in a skewed pattern as depicted in Fig.3, in the matrix inside the grey frame. A single column-vector of this matrix corresponds to one antidiagonal of the original matrix layout of Fig.2(b). Fig.3 illustrates an instance of this skewed memory during computation. At this timestamp, n antidiagonals are already computed and the $n+1^{\text{th}}$ antidiagonal is currently being evaluated. The antidiagonals/columns on the right side of the $n+1^{\text{th}}$ are not yet computed. The reference sequence is streamed through the array of PEs one base at a time. Within m steps all reference bases stream through one PE and a row of the similarity matrix is computed. For example, Fig.3 highlights with green indications the computation of the 1st row of H matrix of Fig.2(b) during the m^{th} time step. Similarly, the n^{th} antidiagonal of H matrix Fig.2(b) is calculated at n^{th} time step. When all input streams pass through all PEs (after $n+m-1$ steps), all values of similarity matrices are computed. The last PE, that generates the values for the last row, runs additional logic to locate the cell with the maximum score in the final row. This operation is completely overlapped with the matrices computation and thus does not add extra clock cycles.

B. Traceback

Traceback module reconstructs the alignment path by traversing the H matrix in reverse order, as depicted in Fig.2(b). Matrix fill streams the position of the maximum score to the Traceback module and buffers out skewed E, F, H matrices in reverse order. Fig.3 demonstrates how Traceback receives one column, and thus an anti-diagonal, of each skewed matrix per time step. At this time instance, Traceback has already resolved the first two backwards steps of Fig.2(b) and evaluates if the next step is in the $n+1^{\text{th}}$ antidiagonal that is currently streamed in. The alignment path computation begins when the anti-diagonal containing the starting point arrives. Depending on the next backwards step, the next cell could either be the up, $left$ neighbor of any of H,F,E matrices, or the $upleft$ element of matrix H. In this exampple, Traceback expects the upleft cell of the current cell and thus has to wait for the next anti-diagonal to arrive. As the traceback progresses, it computes the alignment score, the number of gaps or substitutions and the starting and ending point of the alignment. These values are streamed to the host along with a list of the edits, that include the type of any encountered mismatches, their position on the read reference and the different base encountered.

C. Architecture Optimizations

1) Interleaving Data Scheme: The computation of a cell value from each PE requires a chain of multiplexers according to Eq.1, and thus introduces a latency L between the computation of consecutive anti-diagonals. To avoid idle

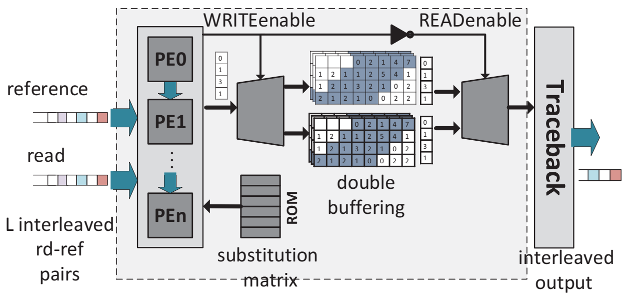

Fig. 4: Architecture of Dataflow Engines on Chip.

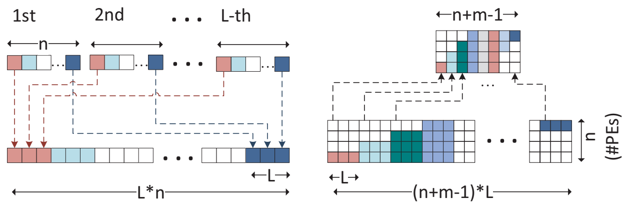

Fig. 5: Interleaving of L read sequences. Interleaving of input streams causes a skewed memory pattern for storing score matrices in BRAM.

TABLE I: Adverse effects of Matrix-Fill Integration.

Matrix Fill Single Task			
	Hardware (sec)		Software (sec)
	Integrated	Computation only	Bowtie2
pre-process	0.000292		
HW run	0.042973		
write-back	0.007764		
Total	0.051029	0.00004	0.000549

clock cycles, the bases of subsequent input read-reference pairs are interleaved in a round robin manner. Fig.5 illustrates the interleaving scheme for L read sequences of L read-reference pairs. The same reordering applies for the reference sequences. This way, L anti-diagonals are computed per L clock cycles instead of just one per L cycles. Each one of the L anti-diagonals corresponds to a similarity matrix of a different read-reference pair. The interleaving also affects the layout of the skewed matrices. Fig.5 demonstrates that between two consecutive anti-diagonals of a single matrix, $L-1$ anti-diagonals from different matrices but of the same order intervene. Note that the skewed layout is preserved.

2) Double Buffering Technique: All values in matrices E, F, H are needed before Traceback starts execution. Therefore, Traceback module has to wait for Matrix-Fill module to complete writing the matrices. Similarly, Matrix-Fill has to wait for Traceback to read the values from the matrices before aligning the next batch of L read-reference pairs and writing over the data. To avoid halting Matrix-Fill's operation while streaming data to Traceback, the *double buffering* technique is employed. Matrix-Fill and Traceback write and read data, respectively, alternating between two on-chip memories. A ”swap” enable signal is controlled by the Matrix-Fill module and decides which data to stream to Traceback. In Fig.3, while Matrix-Fill fills the matrices for the current read-reference pair, the matrices of the previous pair are streamed in a last in- first out fashion to Traceback. Therefore, the two modules are carefully synced and orchestrated to operate in a pipelined manner and continuously align any number of incoming read-reference pairs, organized in batches of L interleaved pairs. The above description is pictured in the overall on-chip architecture in Fig.4.

V. Accelerator Integration with Bowtie2 Aligner

A. Bowtie2 Alignment Algorithm

Bowtie2 [6] is based on the seed-and-extend model for aligning short reads. In the *seed phase*, each read is fragmented into smaller sequences called seeds, and then each seed is aligned against the human genome in an ungapped fashion, creating a set of candidate positions for full-alignment of the initial read. A pre-built BWT-based index of the reference genome is utilized to efficiently search through the immense human genome for seed matches. In the *seed extend phase*, each seed is extended into full alignment by performing SIMD-accelerated dynamic programming, i.e. Smith-Waterman. Once the matrices are filled in, Bowtie2 traverses the final row and locates the cell with the maximum score, that is also greater that a predefined threshold, and initiates a traceback procedure.

During the traceback, a list of edit operations are stored when mismatches (gaps,substitutions) occur. Along with information about the position of the alignment, an alignment result is formed. This is repeated for a different number of seeds per each short read, until all valid alignments are found or an upper limit of tries is reached. All the found alignment results are then sorted based on the alignment score and the best alignment is reported on the *output* in SAM format [25]. Note that a single read alignment is in fact a chain of seed-extend alignments and neither the order nor the number of examined seeds can be known a priori.

B. Proposed Co-designed Bowtie2

1) Straightforward Integration Implications: Bowtie2 repeats the above algorithm for all reads in an input FASTQ format file [26]. Each read is first fragmented into seeds and mapped to exact matches in the genome utilizing the FM-index. The seeds are prioritized in descending order based on their probability to deliver an exact alignment upon extension. A designated function then iterates over the prioritized seeds and initiates Matrix-Fill tasks followed by Traceback, until the read alignment is resolved. This algorithmic structure spawns an unbound number of alignment tasks per read, each one depending on the previous seed-extension alignment task.

A decision to integrate a Matrix-Fill accelerator into Bowtie2 introduces two major types of overhead, as shown in Table I. The first one is attributed to the transfer cost of writing the matrices required by Traceback back to the CPU. The second one is due to the accelerator invoking cost. For a single matrix fill task these overheads translate to $\times 100$ slowdown, despite the speedup acquired from the pure matrix-fill computation.

2) Alleviating Communications and Transfer Overhead: In order to eliminate these overheads, we make two critical

TABLE II: Architecture Utilization for Varying Read-Reference Lengths. Interleaving factor equals to 32 in all cases.

Configuration		Utilization		
Read Length	Reference Length	BRAM18	DSP	Logic
50	110	11.90%	0.10%	9.7%
100	160	25.25%	0.10%	15.33%
150	210	43.03%	0.10%	20.97%
200	260	61.30%	0.10%	26.66%

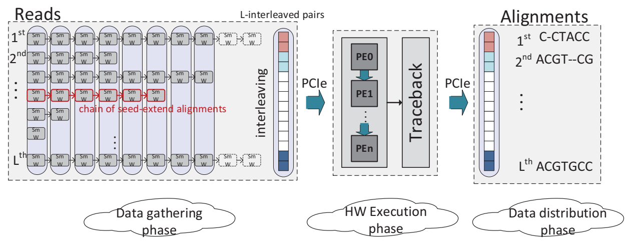

Fig. 6: Restructured Bowtie2 Three-phase Algorithm. For simplicity, only a single batch of L reads is illustrated.

architectural decisions. The first one involves moving the Traceback computation to the HW and targets alleviating the transfer cost of matrices. The second one employs major software restructuring to constraint the number of acceleration calls and thus avoid the calling overhead.

The proposed code restructuring splits the above iterative algorithm in three separate phases; a data-gathering phase, a hardware execution phase and lastly a data-distribution phase. In the first phase, we iterate over the input reads and perform seed extraction and prioritization. During this search phase, we formulate the input streams of the accelerator. Each seed location produces a pair of read-reference sequences to align. In order to seamlessly invoke the accelerator and exploit all the parallelism it provides, the construction of the input streams is compliant with the data interleaving technique explained in Section IV. Thus, for every L read-reference pairs, the read and reference sequences are interleaved as shown in Fig.5. Note that each of L pairs, corresponds to a seed-extension task of a different read.

Once the input to the accelerator is constructed, the accelerator is invoked for the execution phase, that includes the matrix-fill and traceback modules running on hardware. During the data-distribution phase, a loop iterates over the output streams to assess the result of the alignments and distribute the data to data structures read by I/O Bowtie2 functions. For each seed alignment pair, the list of edit operations is traversed in order to construct a valid alignment result and insert it in a list of all alignments for the given read. A subsequent function, checks the data structure that holds the alignments per read in order to report to the output the alignment in SAM format. Compliance between the accelerator output and the structures traversed for buffering the output is critical for seamless integration.

C. Architecture Adaptiveness

Each phase in the three-phase restructured Bowtie2, runs for a predefined number of reads N before passing the results to the next phase. Data generated during the gathering phase are required in the data-distribution phase for output reporting. Therefore, the number of reads N for which the three stages execute directly defines the amount of data stored. The three-stage algorithm is executed iteratively in batches of N until the input reads are exhausted. The accelerators can receive high throughput input streams of any length, thus N also defines the total number of acceleration calls. The proposed architecture allows for the value of N to be configured at run-time. Depending on the size of the input dataset and the memory specifications of the platform, there is an optimal N for which the total accelerator call overhead is minimized. A small optimization study for our utilized platform is presented in Section VI.

There is also an inherent irregularity among alignment of different reads in Bowtie2, as an aligner driver function decides on the order and number of seed-extension alignments examined within each single read alignment. The proposed architecture takes into account this variability and allows for the number of seeds examined per read to be configured depending on the input dataset requirements. The quality of the input reads defines the required number of examined seeds per read so that the alignment accuracy is preserved.

Fig.6 illustrates the three-phase restructured Bowtie2, for $N = L$ for simplicity. In fact, the input to one iteration of the three stages is of size $N = factor * L$. The three-phases are repeated for any number of batches of N. Fig.6 also depicts the interleaving schema across different reads and the constraint on seed-extension alignments per read examined.

VI. Experimental Results

A. Experimental Setup

The kernels were implemented using the dataflow computing model developed by Maxeler Technologies, targeting Maxeler's MAX5C DFE (dataflow engine) [27] with Xilinx VU9P Ultrascale FPGAs. The DFE consists of a large capacity arithmetic chip, x8 PCIe Gen 2 connectivity and it has 38 MB of on-chip SRAM fast memory. MaxCompiler version 2018.1 and Vivado 2017.4 were used to synthesize the design and the achieved frequency is 200MHz. Table II includes parameters values and utilization results of the resulted architecture. As shown, the proposed accelerator has been carefully designed and implemented delivering very low FPGA resources consumption, thus either allowing its potential deployment to smaller FPGA devices, or enabling more aggressive data parallel implementations through multiple accelerator instantiations in a single MAX5C FPGA device. Without loss of generality, in the rest of the section, we perform all the experimental evaluation of the proposed design considering the case of a single accelerator intantiation for configuration no.2.

The read-reference pairs for alignment were acquired by real patient data generated for the purposes of AEGLE EU funded project [9] considering the Chronic Lymphotic Leukemia case.

B. Architecture Evaluation

The proposed design is evaluated in terms of performance, scalability and accuracy.

Input buffer size sensitivity: The two modules receive high throughput input streams of any length. The only limitation is that the buffer size containing the read-reference pairs is a

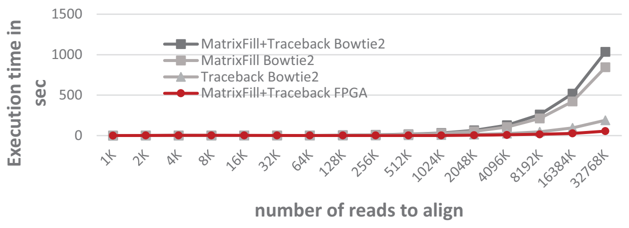

Fig. 7: Alignment and Traceback Latency for different implementations.

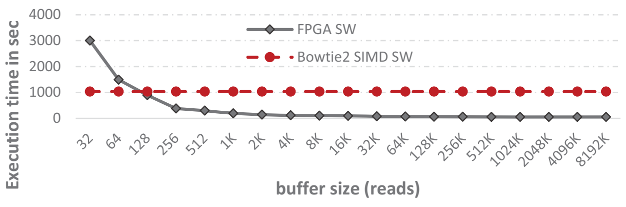

Fig. 8: Impact of input buffer size on accelerator-call overhead and thus execution latency.

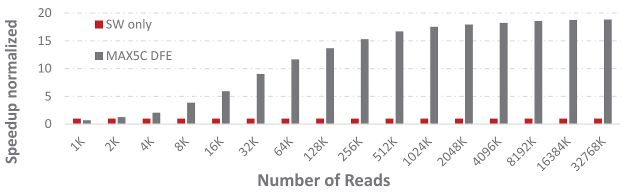

Fig. 9: Speedup of Proposed Accelerator over Bowtie2 SIMD Smith-Waterman.

multiple of interleaving factor L. An exploration is performed to identify the optimal range of values for N, that defines the amount of intermediate data stored and the buffer size of the input streams sent to the accelerator. For that purpose, 30 million reads are aligned in total, but each time the accelerator is invoked a different number of times depending on the buffer size. The results are presented in Fig.8.

Fig.8 indicates that for too small buffer size the execution severely slows down and can be worse that the software execution utilized by Bowtie2. On the other hand, a large buffer size does not seem to influence the performance for length greater than 64K reads. Thus, depending on the input dataset size, there is a value for N for which the acceleration call overhead is negligible.

Accuracy sustainability: Biomedical applications introduce strict accuracy constraints. In order to ensure the correctness of our design, we performed validation of all components and intermediate results. Given a read-reference candidate pair to align, our design is guaranteed to deliver exactly the same results as Bowtie2. We have only loosened the constraints regarding the number of seed extensions examined when aligning a read. After extensive profiling performed on a 60-million-read realistic dataset from CLL malignancies, we found that i) 83% of the reads are aligned within examination of the first 8 seed-extension candidate pairs, ii) another 16% requires checking 9 to 35 pairs and iii) only a mere 1% examines up to 270 (Fig.1).

However, the candidate seed read-reference pairs are not randomly examined. Bowtie2 ranks them so that the more unique ones, i.e. occur less frequent, are tried first. As a result, there is very high probability that the read-reference candidate that delivers the valid alignment is among the highly ranked ones. Furthermore, if an alignment has not been found within the first tries, it is less probable that this read aligns at all.

Based on these facts, the proposed solution manages to achieve very high accuracies when examining 8 candidates per read: 0.72% instead of 0.69% of the total reads fail to align, 69.66% instead of 69.55% align exactly once and finally 29.62% as opposed to 29.76% align more than once.

Performance sustainability over increased datasets: Fig.7 depicts the execution latency for different implementations of Matrix Fill with Traceback for variable input dataset size. The total execution latency of Bowtie2 Smith-Waterman is computed by adding the partial times of executing the matrix fill and traceback. Although, the traceback does not greatly affect the latency, in the proposed design it is completely pipelined with the matrix computation. The importance of implementing Traceback on hardware is not the execution time savings but avoiding data transfer overhead.

The execution latency of the accelerator is compared with the latency of the highly optimized SIMD Bowtie2 implementation, run on an Intel Xeon E5-2658A, 2.20GHz. The speedup values in Fig.9 are achieved by our proposed design synthesized at 200MHz. For small datasets, the accelerator is actually slower that the highly optimized Bowtie2 implementation. In this case the computation time on the FPGA is too short and the transferring of input and output data dominates and worsens the performance. For larger datasets though the speedup reaches $\times 18$.

C. *Evaluating integration efficiency*

In order to evaluate the accelerator's efficiency under realistic integration, the restructured Bowtie2 with the integrated accelerator is executed for 60 million reads (16GB). The software version runs for 8978 seconds while the accelerated one for 5827, thus delivering 35% performance gain. Despite the moderate performance enhancement, a straightforward integration of the matrix-fill accelerator without the traceback and the restructuring, would invoke the kernel at least 60 million times. Without taking into account the transfer time of the matrices, but only the call overhead (measured 0.5ms in the specific architecture), the total overhead sums up to 3000000 seconds, which is more that the initial execution time.

VII. Conclusions

In this paper, we design a novel high performance reconfigurable accelerator for Smith-Waterman matrix fill and traceback, integrated in Bowtie2 aligner for short-read alignment of NGS data. The implementation of Traceback on hardware diminishes data transfer overhead in the codesigned architecture. Bowtie2 source code restructuring accommodates data aggregation through an interleaving scheme for maximum fpga resources utilization and therefore minimizes the number of accelerator calls. Experimental results show that the accelerator is $\times 18$ faster than Bowtie2 SIMD implementation and the codesigned Bowtie2 aligner exhibits 35% performance gain.

References

[1] S. T. Park and J. Kim, "Trends in next-generation sequencing and a new era for whole genome sequencing," *International neurourology journal*, vol. 20, no. Suppl 2, p. S76, 2016.

[2] S. J. Bielinski, J. E. Olson, J. Pathak, R. M. Weinshilboum, L. Wang, K. J. Lyke, E. Ryu, P. V. Targonski, M. D. Van Norstrand, M. A. Hathcock *et al.*, "Preemptive genotyping for personalized medicine: design of the right drug, right dose, right timeusing genomic data to individualize treatment protocol," in *Mayo Clinic Proceedings*, vol. 89, no. 1. Elsevier, 2014, pp. 25–33.

[3] M. A. DePristo, E. Banks, R. Poplin, K. V. Garimella, J. R. Maguire, C. Hartl, A. A. Philippakis, G. Del Angel, M. A. Rivas, M. Hanna *et al.*, "A framework for variation discovery and genotyping using next-generation dna sequencing data," *Nature genetics*, vol. 43, no. 5, p. 491, 2011.

[4] M. D. Robinson, D. J. McCarthy, and G. K. Smyth, "edger: a bioconductor package for differential expression analysis of digital gene expression data," *Bioinformatics*, vol. 26, no. 1, pp. 139–140, 2010.

[5] B. Schmidt and A. Hildebrandt, "Next-generation sequencing: big data meets high performance computing," *Drug discovery today*, vol. 22, no. 4, pp. 712–717, 2017.

[6] B. Langmead and S. L. Salzberg, "Fast gapped-read alignment with bowtie 2," *Nature methods*, vol. 9, no. 4, p. 357, 2012.

[7] H. Li, "Aligning sequence reads, clone sequences and assembly contigs with bwa-mem," *arXiv preprint arXiv:1303.3997*, 2013.

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

[9] http://www.aegle uhealth.eu/en/, "Aegle: An analytics framework for integrated and personalized healthcare services in europe."

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

[11] M. Gok and C. Yilmaz, "Efficient cell designs for systolic smith-waterman implementations," in *Field Programmable Logic and Applications, 2006. FPL'06. International Conference on*. IEEE, 2006, pp. 1–4.

[12] P. Zhang, G. Tan, and G. R. Gao, "Implementation of the smith-waterman algorithm on a reconfigurable supercomputing platform," in *Proceedings of the 1st international workshop on High-performance reconfigurable computing technology and applications: held in conjunction with SC07*. ACM, 2007, pp. 39–48.

[13] Y.-T. Chen, J. Cong, Z. Fang, J. Lei, and P. Wei, "When apache spark meets fpgas: a case study for next-generation dna sequencing acceleration," in *The 8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)*, 2016.

[14] J. Arram, K. H. Tsoi, W. Luk, and P. Jiang, "Reconfigurable acceleration of short read mapping," in *2013 IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2013, pp. 210–217.

[15] ——, "Hardware acceleration of genetic sequence alignment," in *International Symposium on Applied Reconfigurable Computing*. Springer, 2013, pp. 13–24.

[16] W. Tang, W. Wang, B. Duan, C. Zhang, G. Tan, P. Zhang, and N. Sun, "Accelerating millions of short reads mapping on a heterogeneous architecture with fpga accelerator," in *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on*. IEEE, 2012, pp. 184–187.

[17] E. J. Houtgast, V.-M. Sima, K. Bertels, and Z. Al-Ars, "An fpga-based systolic array to accelerate the bwa-mem genomic mapping algorithm," in *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), 2015 International Conference on*. IEEE, 2015, pp. 221–227.

[18] H.-C. Ng, S. Liu, and W. Luk, "Reconfigurable acceleration of genetic sequence alignment: A survey of two decades of efforts," in *Field Programmable Logic and Applications (FPL), 2017 27th International Conference on*. IEEE, 2017, pp. 1–8.

[19] S. A. Guccione and E. Keller, "Gene matching using jbits," in *International Conference on Field Programmable Logic and Applications*. Springer, 2002, pp. 1168–1171.

[20] X. Jiang, X. Liu, L. Xu, P. Zhang, and N. Sun, "A reconfigurable accelerator for smith–waterman algorithm," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 54, no. 12, pp. 1077–1081, 2007.

[21] S. Lloyd and Q. O. Snell, "Hardware accelerated sequence alignment with traceback," *International Journal of Reconfigurable Computing*, vol. 2009, p. 9, 2009.

[22] K. Benkrid, Y. Liu, and A. Benkrid, "A highly parameterized and efficient fpga-based skeleton for pairwise biological sequence alignment," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 4, pp. 561–570, 2009.

[23] Y. Turakhia, G. Bejerano, and W. J. Dally, "Darwin: A genomics co-processor provides up to 15,000 x acceleration on long read assembly," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2018, pp. 199–213.

[24] E. E. Schadt, S. Turner, and A. Kasarskis, "A window into third-generation sequencing," *Human molecular genetics*, vol. 19, no. R2, pp. R227–R240, 2010.

[25] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, and R. Durbin, "The sequence alignment/map format and samtools," *Bioinformatics*, vol. 25, no. 16, pp. 2078–2079, 2009.

[26] ——, "The sequence alignment/map format and samtools," *Bioinformatics*, vol. 25, no. 16, pp. 2078–2079, 2009.

[27] O. Pell, O. Mencer, K. H. Tsoi, and W. Luk, "Maximum performance computing with dataflow engines," in *High-performance computing using FPGAs*. Springer, 2013, pp. 747–774.