OpenCL-based FPGA Accelerator for Semi-Global Approximate String Matching Using Diagonal Bit-Vectors

David Castells-Rufas*, Santiago Marco-Sola*†, Quim Aguado-Puig*, Antonio Espinosa-Morales*, Juan Carlos Moure*, Lluc Alvarez†, Miquel Moretó†

*Departament d'Arquitectura de Computadors i Sistemes Operatius
Universitat Autònoma de Barcelona (UAB)
Bellaterra, Spain

†Barcelona Supercomputing Center (BSC)
Barcelona, Spain

***Abstract*—An FPGA accelerator for the computation of the semi-global Levenshtein distance between a pattern and a reference text is presented. The accelerator provides an important benefit to reduce the execution time of read-mappers used in short-read genomic sequencing. Previous attempts to solve the same problem in FPGA use the Myers algorithm following a column approach to compute the dynamic programming table. We use an approach based on diagonals that allows for some resource savings while maintaining a very high throughput of 1 alignment per clock cycle. The design is implemented in OpenCL and tested on two FPGA accelerators. The maximum performance obtained is 91.5 MPairs/s for 100×120 sequences and 47 MPairs/s for 300×360 sequences, the highest ever reported for this problem.**

***Index Terms*—FPGA, Pre-alignment Filter, Bit-Parallel Alignment, Levenshtein Distance, Genomics, Sequencing**

I. Introduction

The problem of finding substrings in a text that are similar to a given pattern has many interesting and practical applications in several domains, such as linguistics, information retrieval, and genomics. The methods to compute a similarity or dissimilarity value depend on specific aspects of the application domain. In genomics, it is generally assumed that gap-affine Smith & Waterman (SW) alignment [1] gives the best similarity value. However, less computational-intensive methods can provide coarse-grain bounds that allow skipping the time-consuming fine-grain computation.

Seed-and-Extend (SE) genomic mappers combine exact search in the seed phase with sequence alignment in the extension phase to keep the complexity under control [2]. Some mappers incorporate the idea of using less computational-intensive alternatives to SW and use pre-alignment filters to discard alignment candidates, hence, moving from the SE approach to the Seed-Filter-and-Extend (SFE) approach. Among other algorithms, the filtering phase might use Bit-Parallel Myers (BPM) [3], which has been optimized for CPUs [4], [5], GPUs [6], and FPGAs [7], [8]. However, the SFE approach introduce some constraints that allow optimizations to be applied that are not considered when using the BPM technique.

In this work, we propose a novel pre-alignment filtering algorithm and its FPGA implementation in order to be integrate it into genomic pipelines of data-centers equipped with FPGA co-processors with OpenCL support. Our approach exploits some of the Myers observations and other optimization opportunities exposed by the SFE application. This novel approach brings FPGA resource savings that can be devoted to address larger sequences or other hardware functions.

The paper is organized as follows. In section II we review the Myers algorithm in detail. In section III we propose our pre-alignment algorithm. In section IV we describe our implementations for different FPGA accelerators. In section V we present the results. Finally, we conclude in section VI.

II. The Myers Bit-Vector Algorithm

The bit-vector Myers algorithm, or BPM [3], for computing the semi-global Levenshtein distance [9] has been extensively studied and implemented within many tools. Due to its exposed bit-level parallelism, it can easily exploit vector units and SIMD instructions of modern computing platforms [4]–[6]. Given P and T, two strings of lengths m and n ($m < n$), the semi-global distance between P and T is a modification of global alignment that allows penalty-free gaps at the beginning and/or at the end of T. To some extent, it is equivalent to finding which substring of T returns the minimum Levenshtein distance (d_{Lev}) when aligned against P. To compute the semi-global Levenshtein distance, many algorithms use a dynamic programming table (D) and the same rules used to compute the global alignment. However, in the case of semi-global alignment, the first row of D must be initialized with zeros. To find the best alignment, the minimum value from the last row must be selected (equation 1).

$$d_{SG}(P,T) = \min_{\forall T' \subset T} d_{Lev}(P,T') = \min_{\forall j \in [1,n]} D_{m,j} \tag{1}$$

In [3], Myers' analysis begins with the observation that any cell from the $m \times n$ dynamic programming table D can only differ by +1,0,-1 with the preceding cell in the horizontal or vertical axis. Along the diagonals, the values of the cells are monotonically increasing (i.e., +0 or +1). From these observations, the original D table can be expressed as three alternative increment tables: Δv, Δh, and Δd, which can be build from the original D table by using the equations 2, 3, and 4 respectively.

$$\Delta v_{i,j} | i > 0 = D_{i,j} - D_{i-1,j} \tag{2}$$

$$\Delta h_{i,j} | i > 0 = D_{i,j} - D_{i,j-1} \tag{3}$$

$$\Delta d_{i,j} | i > 0, j > 0 = D_{i,j} - D_{i-1,j-1} \tag{4}$$

The original D table can be easily reconstructed from any of the three increment tables, but in Myers' case he use the horizontal table. Equation 5 showns how the value of any cell $D_{i,j}$ can be computed by taking the value of the first (pre-computed) cell of the row and aggregating all horizontal increments of the same row until the position i, j is reached.

$$D_{i,j} = i + \sum_{k=1}^{j} \Delta h_{i,k} \tag{5}$$

The great advantage of the Myers' algorithm is that it finds an alternative way to build the increment tables (Δv, Δh, and Δd) by using simple boolean expressions without computing D. To do it, he first separates positive and negative increments (equations 6, 7, and 8).

$$\Delta v_{i,j} | i > 0 = VP_{i,j} - VN_{i,j} \tag{6}$$

$$\Delta h_{i,j} | j > 0 = HP_{i,j} - HN_{i,j} \tag{7}$$

$$\Delta d_{i,j} | i > 0, j > 0 = 1 - D0_{i,j} \tag{8}$$

Then, Myers analyzes all the possible combinations to obtain simple boolean expressions to describe the increment equations 9,10,11, 12, and 13.

$$HN_{i,j} = VP_{i,j-1} \wedge D0_{i,j} \tag{9}$$

$$VN_{i,j} = HP_{i-1,j} \wedge D0_{i,j} \tag{10}$$

$$HP_{i,j} = VN_{i,j-1} \vee \neg(VP_{i,j-1} \vee D0_{i,j}) \tag{11}$$

$$VP_{i,j} = HN_{i-1,j} \vee \neg(HP_{i-1,j} \vee D0_{i,j}) \tag{12}$$

$$D0_{i,j} = \neg(P[i] \oplus T[i]) \vee VN_{i,j-1} \vee HN_{i-1,j} \tag{13}$$

Finally, the Myers' algorithm (figure 6 in [3]) follows a column approach to compute the bit-vectors for the whole table but only computing the distance values for the cells of the last row. It exploits the architecture word size (typically 32 or 64 bits) to compute the bit-vectors of a column simultaneously.

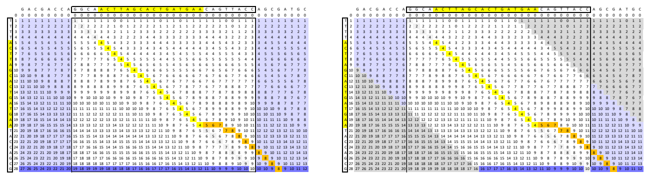

Fig. 1. D matrix use by BPM Myers's algorithm (left) compared with that of our proposal (right). Locations where the seed matched are underlined in yellow. The areas of the table that are extended to cover indel errors are underlined in light-blue. The cells (from the last row) were the actual D value is recovered are underlined in dark-blue. Cells that are not computed are underlined in light-grey.

Since the value $D_{m,0} = m$, the values from the last row can be obtained by accumulating $\Delta h_{m,j}$. The algorithm requires n iterations to find the minimum distance value from the last row.

Some FPGA implementations of the Myers algorithm have been proposed in the literature [7], [8]. In [10], Hyyrö proposes a bit-vector algorithm to compute the semi-global distance below a threshold k. It is based on Myers' binary equations and uses a banded matrix to reduce the required computation. To the best of our knowledge, there is no FPGA implementation based on that work. Our implementations shares some ideas with Hyyrö's work.

III. Proposal Using the Diagonals

One important aspect of SFE mappers is how the reference string T is selected. The extension phase starts after an exact match of a seed S of length s from the string P has been found in a certain location in the genome G. Let l_P be the location of the seed in the pattern P, so that $S = P[l_P : l_P + s]$, and l_G the location of the seed in the genome G, so that $S = G[l_G : l_G + s]$. Since k insertion or deletion (indel) errors are allowed, the size of T is expanded as defined by equation 14.

$$T = G[l_G - l_P - k : l_G - l_P + m + k] \tag{14}$$

Figure 1 (left) illustrates how this strategy is used together with Myers' BPM algorithm. Note that the extension is crucial to capture the best alignment (underlined in orange). In case we would be only considering substitution errors ($k = 0$), the only cells from the D matrix that we would be computing would be the diagonal ones. When the area is expanded to consider indel errors (underlined as light blue), this diagonal becomes the k diagonal. In BPM, all HP, HN, VP, VN, and $D0$ are computed. and only the D values for the last row (underlined in blue) are recovered.

Based on the foundation of Myers' bit equations, we have identified additional optimization opportunities:

- BPM computes the scores for every element of the last row ($D_{m,j} \forall j \in [1, n]$), but most of those scores are irrelevant since we know that, if derived from the exact match of the seed, the best alignments must be in a radius $2k$ from the k diagonal; that is, we only need to compute $D_{m,j} \forall j \in [m - k, m + 2k]$.

- The use of equation 5 to compute $D_{m,j}$ requires to use adders that can represent the worst case value, which is m. Hence, $\lceil \log_2(m) \rceil$ bits adders are required. But if we are only interested in determining if the score of the interesting cells is not greater than k we could use adders saturating at $3k+1$, requiring adders of only $\lceil \log_2(3k+1) \rceil$ bits.

As a result of the previous observations, we propose computing $D_{m,m+k}$ using the diagonal increments Δd and equation 15. We compute the rest of the elements in the neighbouring radius $2k$ from the last row using Δh and equation 16 and finally select the minimum from them (equation 17). We obviate the use of saturating adders in these equations to facilitate the reading, but we use them to reduce the required resources.

$$D_{m,m+k} = \sum_{i=1}^{m} \Delta d_{i,i+k} \tag{15}$$

$$D_{m,j} = \begin{cases} D_{m,m+k} + \sum_{i=m+k+1}^{j} \Delta h_{m,i} & \text{, if } j > m+k \\ D_{m,m+k} - \sum_{i=j+1}^{m+k} \Delta h_{m,i} & \text{, if } j < m+k \end{cases} \tag{16}$$

$$d_{est}(P,T) = \min_{\forall j \in [m-k, m+2k]} D_{m,j} \tag{17}$$

Using the diagonal approach to compute $D_{m,m+k}$ allows to prune out cells that do not contribute to the final solution. We cut the data dependency between cells by introducing hard-coded positive increments in the cells that are far enough from the main diagonal.

The amount of saved resources depend on the value k, but for low values of k they might be quite significant. The number of avoided cells are $m \times n - (m \times (4k+1)) + k^2$. Figure 1 (right) depicts the elements of table computed with this approach. The cells avoided are underlined in light-grey.

IV. Implementation

Besides the benefits associated with the use of High-Level Synthesis (HLS), a great advantage of OpenCL is the easy migration of the code to different FPGA-based accelerator devices as it is not required to invest time in the development of the infrastructure to communicate with the host CPU. Genomic datacenters could use their own FPGAs or could use computing resources from public heterogeneous Cloud providers. OpenCL multi-platform support enables easy migration of workloads among different platforms, while its HLS infrastructure allows an easy adaptation to different parameters of the workloads, such as different read-lengths or error rates.

Most FPGA accelerators use on-board DDR memory to share data between the host CPU and the accelerator. Memory transfers between them must go through the PCIe bus. Another kind of accelerators promote the direct access to main host memory through a cache coherent interface. This is the case of the Intel HARPv2 system and the devices using the Coherent Accelerator Processor Interface (CAPI) proposed by IBM [11].

We implement our design as a single workitem OpenCL kernel to promote the creation of a long pipeline that is able to process a batch of sequence pairs in a loop with an initiation interval of 1 and a single clock throughput. This throughput could be achieved if, for each input pair, we are able to fetch both sequences and store a previously computed distance result at the same time. In practice, this is not feasible as we have a limited input bandwidth, and we have to multiplex the communication channel to store results. For each input sequence pair, we use $2(m+n)$ bits, as we encode each base in 2 bits. The computed distance output value requires $\lceil log_2(k+1) \rceil$ bits. Therefore, the amount of output data is much lower than the input data to be processed. To favor long bursts in accessing input data, we postpone the writing of output results by buffering them into an FPGA on-chip memory.

The input data bandwidth is limited by the minimum between bus bandwidth and memory bandwidth. Since bus bandwidth is always less than memory bandwidth, and no data reuse occurs, the system does not benefit from having an exclusive device memory. The organization of sequence pairs in memory has an impact on the filter throughput. We use three different memory layouts depending on the length of the sequences. In the first case, both sequences are packed in a single 512 bits word. In the second case, each sequence is stored in a different 512 bits word. In the third case, each sequence is stored in different 1024 bit words. Depending on the layout, data fetch can take either 1, 2, or 4 clock cycles.

Hardware implementations of dynamic programming algorithms (like [12]) tend to compute the cells of the anti-diagonals in parallel. If the allocated processing elements (PEs) are created to address the longest anti-diagonal, the matrix computation takes several cycles and the PEs are be reused. Another alternative is to create a PE for each cell of the matrix and compute the anti-diagonals in pipeline. Unlike classic HDL designs we do not create a specific PE for cell computation. We can still interpret that there is a kind of virtual PE diluted in the OpenCL source code for each matrix cell.

Arbitrary precision integer OpenCL types are very convenient when dealing with large bit-vectors. However, their support is not consistent over different manufacturers [13]. To overcome this limitation we create our own library of arbitrary precision integers by using metaprogramming. Instead of using a C pre-processor based metaprogramming approach we use a syntax similar to Java Server Pages [14] to annotate the parts of the OpenCL source code that must be modified before compilation.

We use an iterative generation algorithm (Algorithm 1) to build the elements of the matrix, pruning out the computation of all cells farther than k from the main diagonal. The direct implementation of the algorithm in OpenCL could possibly lead the compiler to detect the data dependency between the iterations of the loops iterations preventing the unrolling of the for loops (in lines 1 and 2). In our case, we explicitly unroll the loops by using metaprogramming to force a pipelined design. After bit-vectors are computed, the algorithm requires a final phase to aggregate the diagonal cells and the required values from the last row to obtain their minimum value. These loops

Algorithm 1 Banded Bit-Vector Algorithm with error threshold

Input: The pattern text P of length m, the reference text T of length n, and the maximum edit distance k

Output: The minimum distance d of all possible substrings of T with P being $\leq k$, or $k+1$ otherwise

```
for $x \leftarrow 1$ to $n+1$ do                          ▷ Compute bit-vectors
    for $y \leftarrow 1$ to $m+1$ do
        if $x = 1$ then
            $VP_{y,x} \leftarrow 1$
            $VN_{y,x} \leftarrow 0$
        else if $y = 1$ then
            $HP_{y,x} \leftarrow 0$
            $HN_{y,x} \leftarrow 0$
        else if $|x-k-y| <= k$ then                        ▷ in-band
            $D0_{i,j} \leftarrow \neg(P[i] \oplus T[i]) \vee VN_{i,j-1} \vee HN_{i-1,j}$
            $VP_{i,j} \leftarrow HN_{i-1,j} \vee \neg(HP_{i-1,j} \vee D0_{i,j})$
            $VN_{i,j} \leftarrow HP_{i-1,j} \wedge D0_{i,j}$
            $HP_{i,j} \leftarrow VN_{i,j-1} \vee \neg(VP_{i,j-1} \vee D0_{i,j})$
            $HN_{i,j} \leftarrow VP_{i,j-1} \wedge D0_{i,j}$
        else if $|x-k-y| = k+1$ then                       ▷ band edge
            if $x < y$ then
                $VP_{y,x} \leftarrow 1$
                $VN_{y,x} \leftarrow 0$
            else
                $HP_{y,x} \leftarrow 1$
                $HN_{y,x} \leftarrow 0$
            end if
        end if
    end for
end for
$D_{m,m+k} = \min(3k+1, \sum_{i=1}^{m} \neg D0_{i,i+k})$                 ▷ last row
for $i \leftarrow m+k+1$ to $m+2k$ do
    $D_{m,i} \leftarrow \min(3k+1, D_{m,i-1} + HP_{m,i} - HN_{m,i})$
end for
for $i \leftarrow m+k-1$ downto $m-k$ do
    $D_{m,i} \leftarrow \min(3k+1, D_{m,i+1} - HP_{m,i+1} + HN_{m,i+1})$
end for
$d \leftarrow \min_{\forall i \in [m-k, m+2k]} D_{m,i}$
```

are also unrolled by our metaprogramming tool.

Another benefit from the OpenCL methodology, is that, since Hardware is defined with C, the OpenCL framework can easily emulate the circuit in software. We use emulation on every circuit to do functional verification before synthesis and execution on the final device.

The source code of our implementation is available at https://github.com/davidcastells/bpc .

V. Results

We synthesize several systems addressing a number of short-read lengths ($m = \{100, 200, 300\}$) and error rates ($e_{rate} = \{3\%, 5\%, 10\%\}$) which are representative from the typical workloads found in short-read sequencing. The implementations are synthesizedd for the Intel accelerator platforms D5005 and HARPv2 (from Intel). Table I shows the maximum clock frequency (f_{max}) and the total consumed resources, including logic elements (LEs) and Flip-Flops (FFs) for all synthesized designs. The results show how kernel clock frequency is not directly related to the complexity of the design, given by the $m \times n$ product. In fact, OpenCL hides the complexity to implement a deep-pipeline to minimize the effects o longer combinational paths. All the designs are successfully implemented with an interval initiation factor of 1.

TABLE I
Synthesis results for D5005 and HARPv2

		D5005			HARPv2		
k	$m \times n$	LEs	FFs	f_{max}	LEs	FFs	f_{max}
3	100×106	436 k (15%)	359 k (9%)	325	291 k (25%)	167 k (9%)	289
5	100×110	466 k (16%)	397 k (10%)	323	297 k (25%)	171 k (10%)	291
10	100×120	514 k (18%)	453 k (12%)	307	323 k (28%)	199 k (11%)	268
6	200×212	494 k (17%)	414 k (11%)	319	321 k (27%)	189 k (11%)	285
10	200×220	568 k (20%)	503 k (13%)	298	358 k (31%)	224 k (13%)	277
20	200×240	609 k (22%)	509 k (13%)	318	447 k (38%)	298 k (17%)	278
9	300×318	787 k (28%)	792 k (21%)	302	399 k (34%)	239 k (14%)	273
15	300×330	646 k (23%)	539 k (14%)	329	562 k (48%)	333 k (19%)	276
30	300×360	929 k (33%)	846 k (22%)	317	663 k (57%)	433 k (25%)	264

We use a benchmarking application that generates synthetic data to measure the performance of the system. The results are shown in table II. We report the throughput of the systems in millions of sequence pairs per second (MPPS) observed by the host application, i.e. including memory transfers. We also provide the number of giga cells of the dynamic table computed per second (GCUPS). This value requires some clarification, since we actually do not compute the whole D table, but only the bit-vectors for $m \times (4k+1) - k^2$ elements of the table and we only recover $3k+1$ values from the last row. The reported GCUPS only consider the cells from the subset of the table that we actually compute.

TABLE II
Performance results when computing 10 MPairs in D5005 and HARPv2

		D5005		HARPv2	
k	$m \times n$	MPPS	GCUPS	MPPS	GCUPS
3	100×106	47.4	61.2	91.5	118.1
5	100×110	47.4	98.4	91.5	189.9
10	100×120	47.4	189.6	91.5	366
6	200×212	29.2	144.9	78.4	389.2
10	200×220	29.2	236.5	78.4	635
20	200×240	29.2	461.5	78.4	1238.7
9	300×318	16.6	182.9	47	517.9
15	300×330	16.6	300	47	849.5
30	300×360	16.6	587.6	47	1663.8

TABLE III
COMPARISON WITH THE STATE OF THE ART

Ref.	FPGA	$m \times n$	MPairs/s	GCUPS
Hoffmann, 2016, [15]	Digilent Zybo	128×128	0.028	0.45
Cai, 2019, [7]	Kintex KCU1500	48×48	70	161.2
Bautista, 2020, [8]	Pico Comp. M505	112×128	0.3	4.3
Ours	D5005	100×120	47.4	189.6
Ours	HARPv2	100×120	**91.5**	366
Ours	D5005	300×360	16.6	587.6
Ours	HARPv2	300×360	47.0	**1663.8**

Our banded approach to compute the Levenshtein distance in the SFE context can compute up to 300 bp read length in a single cycle. Althrough the performance is limited by the memory bandwidth, according to the results, most of our designs are above the hundred GCUPS region and, for larger designs, above the tera cells updates per second (TCUPS) region.

To the best of our knowledge these are the highest performance values obtained using FPGAs reported in the literature. There have been few attempts to address a similar problem using FPGAs. Cai in [7], implemented an OpenCL design based on the Myers algorithm. He observed than the Myers loop could be unrolled for low values of n, providing a single clock solution for the whole table. However, although the FPGA used in his work is similar in number of resources to HARPv2, his biggest design is 48×48, getting a performance of 70 MPPS. Not only we are able to address much bigger designs up to 300×360, but in comparison, for designs of 100×100 we get up to 91.5 MPPS in HARPv2. A recent work from Bautista [8] does not provide an improvement in terms of performance. On the other hand, he provides the backtrace. Although this could be useful for other applications, pre-alignment filters used in the SFE context do not require the alignment backtrace. Table III summarize how our best designs compare with the state of the art, both in MPPS and GCUPS. Since the number of achieved MPPS depend on the memory used layout to transfer the input data, we get the maximum value (91.5 MMPS) for sequence pairs that can be packed in a single 512 bit memory transfer.

VI. CONCLUSION

We have presented a novel bit-vector algorithm to compute the banded Levenshtein distance in the context of SFE mappers. The banded approach brings resource savings that allow to address bigger problem sizes up to 300×360 bases in a single clock cycle. Up to 91.5 million semi-global alignments per second are achieved with a single module on the HARPv2 platform for 100×120 sequences, which is higher than previous results provided by [7] on 48×48 sequences. The achieved performance of 47 MPPS on 300×360 (1.6 TCUPS), the highest performance reported for this problem. We have observed that a single module already saturates the available bandwidth of the system, in future work we will investigate the use of more sophisticated memory subsystems (such as HBM2) and bus connections (such as PCIe-4, NVLink, CXL) and their impact in the obtained performance.

ACKNOWLEDGMENT

This research was supported by the EU Regional Development Fund under the DRAC project [001-P-001723], by the MINECO-Spain (contract TIN2017-84553-C2-1-R), by the MICIU-Spain (contract RTI2018-095209-B-C22) and by the Catalan government (contracts 2017-SGR-1624, 2017-SGR-313, 2017-SGR-1328). M.M. was partially supported by the MINECO under RYC-2016-21104. We thank Intel for granting us access to the DevCloud system and let us join the HARP research program. The presented HARP-2 results were obtained on resources hosted at the Paderborn Center for Parallel Computing (PC2) in the Intel Hardware Accelerator Research Program (HARP2).

REFERENCES

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

[2] N. Ahmed, K. Bertels, and Z. Al-Ars, "A comparison of seed-and-extend techniques in modern DNA read alignment algorithms," in *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE, 2016, pp. 1421–1428.

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

[4] M. Šošić and M. Šikić, "Edlib: a c/c++ library for fast, exact sequence alignment using edit distance," *Bioinformatics*, vol. 33, no. 9, pp. 1394–1395, 01 2017.

[5] Y. Chan, K. Xu, H. Lan, B. Schmidt, S. Peng, and W. Liu, "Myphi: efficient levenshtein distance computation on xeon phi based architectures," *Current Bioinformatics*, vol. 13, no. 5, pp. 479–486, 2018.

[6] A. Chacón, S. Marco-Sola, A. Espinosa, P. Ribeca, and J. C. Moure, "Thread-cooperative, bit-parallel computation of levenshtein distance on GPU," in *Proceedings of the 28th ACM international conference on Supercomputing*, 2014, pp. 103–112.

[7] L. Cai, Q. Wu, T. Tang, Z. Zhou, and Y. Xu, "A Design of FPGA Acceleration System for Myers bit-vector based on OpenCL," in *2019 International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS)*. IEEE, 2019, pp. 305–312.

[8] D. P. Bautista, R. C. Aguilera, F. A. Acevedo, and I. A. Badillo, "Bit-Vector-Based Hardware Accelerator for DNA Alignment Tools," *Journal of Circuits, Systems and Computers*, p. 2150087, 2020.

[9] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, vol. 10, no. 8, 1966, pp. 707–710.

[10] H. Hyyrö, "A bit-vector algorithm for computing levenshtein and damerau edit distances," *Nordic J. of Computing*, vol. 10, no. 1, p. 29–39, Mar. 2003.

[11] B. Wile, "Coherent accelerator processor interface (capi) for power8 systems white paper," *IBM Systems and Technology Group*, 2014.

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

[13] L. Forget, Y. Uguen, F. de Dinechin, and D. Thomas, "A type-safe arbitrary precision arithmetic portability layer for HLS tools," in *Proceedings of the 10th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*, 2019, pp. 1–6.

[14] J. Hunt, "Java server pages," in *Java and Object Orientation: An Introduction*. Springer, 2002, pp. 361–370.

[15] J. Hoffmann, D. Zeckzer, and M. Bogdan, "Using fpgas to accelerate myers bit-vector algorithm," in *XIV Mediterranean Conference on Medical and Biological Engineering and Computing 2016*. Springer, 2016, pp. 535–541.