Hardware acceleration of the novel two dimensional Burrows-Wheeler Aligner algorithm with maximal exact matches seed extension kernel

Mahdi T Taheri | Mohammad Saeed Ansari | Sebastian Magierowski | Ali Mahani

1Department of Electrical Engineering, Reliable and Smart Systems Lab (RSS), Shahid Bahonar University of Kerman, Kerman, Iran
2Eideticom Computational Storage, Calgary, Alberta, Canada
3Department of Electrical Engineering and Computer Science, Lassonde School of Engineering, York University, Toronto, Ontario, Canada

Abstract

Next-generation sequencing techniques have dramatically increased the amount of genomic data being sequenced, which calls for the acceleration of the alignment algorithms. This article proposes an FPGA-based accelerated implementation of the seed extension kernel of the Burrows–Wheeler alignment genomic mapping algorithm. The well-known Smith–Waterman algorithm is used during the seed extension to find the optimum alignment between the sequences. The state-of-the-art architectures in the literature use one-dimensional (1-D) systolic arrays to fill a similarity matrix, based on the best score out of all match combinations, mismatches and gaps are computed. The cells on the same anti-diagonal are computed in parallel in these architectures. We propose a novel 2-dimensional architecture in which all the cells on the same rows and the same columns are computed in parallel and, thereby, significantly accelerated the process. The similarity matrix cells are computed in two phases: (1) the calculation phase and (2) error compensation phase. The calculation phase roughly approximate the cell values and the approximation error is fixed up during the error compensation phase. Our simulation results show that the proposed architecture can be up to 718x and 1.7x faster than the software execution and the 1-D systolic arrays, respectively.

1 | INTRODUCTION

As a result of advances in next-generation sequencing (NGS) techniques, the amount of genomic data is accumulating extremely rapidly [1]. Some project a growth of the order of 10^18 bytes per year by the next decade [2,3]. Naturally, processing such large-scale data sets imposes a significant computational load for many applications in bioinformatics. The new NGS techniques reduce the cost of generating a whole human genome, which in return increases the amounts of data even more [4]. Typically, NGS techniques produce millions of short reads, usually less than 200 bases in length [5]. In many cases, the short reads must be aligned to a reference sequence to obtain useful information about the sequenced molecule [6]. There are a number of different alignment algorithms to choose from, among which dynamic programming solutions, such as Bowtie [7], Burrows–Wheeler alignment (BWA) [8] and SOAP2 [9] are shown to provide a better trade-off between the accuracy and speed. The most well-known BWA-MEM algorithm, which is famous for its high accuracy, is considered in this article as the target for FPGA acceleration. The BWA-MEM algorithm, introduced in Ref. [10], manages superior alignment to a number of its recent papers; however, at the cost of a higher computational load. The three main operations in the BWA-MEM algorithm are: (1) generation of the super-maximal exact matches (SMEMs), (2) extending the seeds and (3) generating the final outputs. The three kernels that perform these operations are listed in Table 1 along with their main bottlenecks [11]. As shown in Table 1, the step with the most severe computational bound is the seed extension kernel that accounts for almost 33% of the execution time.

During seed extension a similarity matrix is dynamically filled. To compute the value of cell \((i, j)\) in the similarity matrix, the values of the three following cells are required: (1) the west cell at \((i, j−1)\), (2) the north cell at \((i−1, j)\) and (3) the northwest cell at \((i−1, j−1)\). The state-of-the-art architectures proposed to accelerate this problem use one-dimensional (1-D) systolic arrays to fill the similarity matrix [8]. Specifically, for...
such computations, a 1-D array of $N$ processing elements (PEs), where $N$ is the number of cells on the main diagonal [11], is employed. Note that increasing the number of PEs in such architectures does not reduce the execution time.

This article proposes a novel FPGA-based architecture to speedup the process of filling the similarity matrix, the computational bottleneck of the seed extension kernel. To the best of our knowledge, this is the first attempt to design a two-dimensional (2-D) architecture for achieving a higher throughput by increasing the number of PEs. The proposed architecture calculates the values of the cells on the same rows and on the same columns in parallel; while, the existing architectures are only capable of compute the values of the cells located on the same anti-diagonal in parallel. Additionally, novel PEs are proposed that compute the cells in two phases: (1) the calculation phase that roughly approximates the cells and (2) the error compensation phase that fixes the potentially introduced errors during the first phase. The two phases are described in detail in Section 3.

The rest of this paper is organized as follows: Section 2 provides general information on the BWA-MEM algorithm and related work. The mathematical description of the proposed architecture, the PE designs, and the final hardware implementation are discussed in Section 3. Section 4 evaluates the proposed architecture and, finally, Section 5 concludes this article.

## 2 | BACKGROUND

DNA sequences are chains comprised of the four nucleotide monomers Adenine (A), Cytosine (C), Guanine (G) and Thymine (T). Alternative permutations of these nucleotides typically encode alternate biochemical functions and products within the DNA. A two-bit representation is used in our implementation for each of these four nucleotides. Although even simple organisms possess contiguous DNA exceeding a million nucleotides in length, DNA measurements (i.e. the products of sequencing machines) rarely produce sequences this long due to a combination of technical limits at present. In fact, it is common for leading sequencers to produce outputs, ‘reads’, on the order of 100 nucleotides in length, obtained from unknown regions of the native DNA originally fed into the sequencer.

As a result, pairwise local alignment, searching for similarities between a new read (the query) and anticipated DNA pattern (reference genome), is one of the first steps in bioinformatics algorithms [12]. The main workload of the BWA-MEM algorithm is aligning millions of short DNA reads against a reference genome (usually the human genome) [13].

The BWA-MEM algorithm consists of three main kernels, which are explained below:

1. SMEM generation: This kernel is used to find seeds (substrings of the reads) that are likely mapping locations of the read against the reference genome. There is a chance of generating several seeds with variable length for each read [8]. This step is an exact-match-finding phase that uses the Burrows–Wheeler transform. For this work, seeds are at least nineteen characters and a maximum of 131.

2. Seed extension: This steps is an inexact-matching step that executes chaining and extending of seeds in two directions by using a Smith-Waterman-like algorithm [14]. This part of the BWA-MEM algorithm finds the optimal local alignment by using a scoring system.

3. Output generation: In this step the best alignment (i.e., the one with the highest score) is finalized and provided as the output in SAM-format, if necessary.

Note that the seed extension kernel used in BWA-MEM is different from the Smith-Waterman algorithm in two substantial ways [11]: (1) Non-zero initial values: the initial values in the first column and row depend on the alignment score of the seed found by the SMEM Generation kernel. (2) Additional output generation: other than the local and global alignment scores, the exact location inside the similarity matrix and a maximum offset (indicating the distance from the diagonal at which a maximum score has been found) are also generated.

Several techniques have been proposed to accelerate the Smith–Waterman inexact alignment algorithm. However, the seed extension step of this algorithm makes it inherently a slow design. A hardware acceleration of the BWA-MEM genomics short read mapping for longer read length is implemented in Ref. [15]. This design is based on a previously proposed architecture in [11], where an FPGA-based 1-D systolic array is used to accelerate the BWA-MEM genomics mapping algorithm. The main idea is to insert some exit points between the PEs of the 1-D systolic array to avoid unnecessary calculations for shorter reads. By doing so, shorter reads do not have to go through all of the PEs and can exit the array once they get to the first exit point.

## 3 | PROPOSED ARCHITECTURE

This section describes the proposed methodology for filling the similarity matrix and introduces the new architecture for implementing the seed extension kernel of the BWA-MEM algorithm.
Unlike the 1-D systolic array architectures, we propose a 2-D architecture in which two different strings of PEs are assigned to all of the cells in the same rows and on the same columns. In fact, \(2N - 1\) PEs are required for a \(N \times N\) matrix compared to \(N\) PEs in the 1-D architectures. Doing so, the Smith–Waterman algorithm is accelerated compared to the conventional 1-D systolic array architectures, as confirmed by our simulation results. Although the proposed architecture is more resource-hungry than the 1-D systolic arrays, it runs significantly faster. Moreover, speed is usually the main bottleneck and hardware resource usage is less of a concern.

Each PE in the proposed architecture computes the values of its assigned cells (a.k.a. filling the similarity matrix) in two phases that, in total, takes only one clock cycle. During the first phase, namely the error editing phase, if the \(\text{Match}_{\text{flag}}\) signal was activated from the last cycle in the calculation phase, the value of each cell is edited by the corresponding PEs. Then the second phase, that is, the calculation phase, starts and \(\text{Match}_{\text{flag}}\) signal gets a new value if match occurs in calculations and the value of each cell is roughly computed by the corresponding PEs. The main advantage of the proposed structure is that the rows and columns calculation of similarity matrix are performed independently. In some rare cases, the parallel calculations lead to approximation in obtained scores. So the editing phase is needed to correct the approximated scores which are obtained from the last clock cycle. Then, the new calculation for other elements of the matrix starts in the second phase.

Figure 1 shows the difference of PEs assignment in the similarity matrix between the proposed method and the 1-D systolic array architectures. The cells with the same number in Figure 1a are computed simultaneously. On the other hand, as shown in Figure 1b, the cells with the same colours are computed in parallel in the proposed architecture. It is evident from Figure 1 that the similarity matrix can be filled in fewer steps by using the proposed architecture. As is indicated from this figure, systolic arrays need five cycles to solve \(3 \times 3\) matrix, while our proposed method needs three clock cycles to solve the same matrix. As previously mentioned, in 1-D systolic arrays the cell on diagonals are computed in each cycle concurrently; however, the cells on the same rows and columns are computed in parallel in the proposed architecture. This is also shown in Figure 1b, where the index of the PE responsible for calculating the value of each cell is written inside that cell. For example, PE1 is in charge of calculating the value of the cell located in the first column and second row (during the first clock cycle), and in the second clock cycle, it calculates the value of the cell on the second column and second row. Furthermore, each PE has to store and share its calculated value with other PEs for further calculations. Another consideration is that if PE0 calculates the value of the cell located on the first column and the first row, it is task is completed and it only needs to share the \(\text{Match}_{\text{flag}}\) with the next PEs (i.e. PE 1, 2, 3, 4) for their editing phase. It also needs to share its stored value with PE1, as it is required for calculating the value of the cell on the main diameter in next clock cycle.

The two phases of the proposed architecture and the details of its hardware implementation are thoroughly explained below:

3.1 Calculation phase

During the calculation phase, an approximation of the value of each cell is calculated. Depending on the position of the cell in the similarity matrix, this approximation follows different rules, as described below:

3.1.1 Cells on the main diagonal

For the cells on the main diagonal, we use the exact PE functionality, as given by:

\[
DP_{(i,i)} = \max\left\{\begin{align*}
DP_{(i-1,i-1)} & + T_{\text{Match,Miss-match}}; \\
DP_{(i-1,i)} & + T_{\text{Gap}}; \\
DP_{(i,i-1)} & + T_{\text{Gap}}; \\
0 & ,
\end{align*}\right. 
\]  

(1)

where \(DP\) denotes the similarity matrix, \(T_{\text{Match,Miss-match}}\) is the assigned score for when a match or a mismatch occurs (we use
+1 and 0 for a match and a mismatch, respectively), and \( T_{(Gap)} \) is the gap penalty (we use \(-1\) in this article).

As shown, Equation (1) considers all the three contributing neighbour cells to cell \((i, j)\) on the main diagonal and its value is thus ‘exact’. However, since the adjacent cells that is, west and north neighbours, might have approximate values (see immediately following sub-section), the output of this equation may be an erroneous value. However, this will not happen (based on the first edit that has been applied in editing phase), as will be discussed shortly in the editing phase. Therefore, the cells on the main diagonal always have an exact value.

### 3.1.2 Other cells

As implied above, the elements that are not placed on the main diagonal use only two of the three adjacent cells. Note that this approximation is a property of the underlying algorithm and not a specific hardware related design choice. For the cells below the main diagonal we use the values of the west and the north-west cells as given by:

\[
DP_{(i,j)} = \text{MAX} \left\{ \frac{DP_{(i-1,j-1)} + T_{(\text{Match, Miss-match})}}{DP_{(i-1,j)} + T_{(\text{gap})}} \right\}
\]

(2)

For the cells on the top of the main diagonal, on the other hand, we use the values of the north and the north-west cells as given by:

\[
DP_{(i,j)} = \text{MAX} \left\{ \frac{DP_{(i-1,j-1)} + T_{(\text{Match, Miss-match})}}{DP_{(i-1,j)} + T_{(\text{gap})}} \right\}
\]

(3)

Figure 2 provides a graphical description of the three above-mentioned equations to make the proposed architecture clearer.

As shown in Figure 2, the cells on the main diagonal (i.e. cells \((1, 1)\) and \((2, 2)\)) use the values of the three adjacent cells while the cell below the main diagonal, that is, \((2, 1)\) uses the values of the west and the north-west adjacent cells. Finally, the cell on the top of the main diagonal that is, \((1, 2)\) uses the values of the north and the north-west adjacent cells.

Note that according to Equations (1-3), two types of PEs are required in the proposed architecture, horizontal and vertical PEs. Horizontal PEs are used to calculate the values of the cells on the same row, from the first column to the column on which the cell on the main diagonal is located and vertical PEs calculate the values of the cells on the same column, from the first row to the row just below the row on which the cell on the main diagonal is located. Note that all PEs are distinct from each other as they use the values of different neighbours to calculate their outputs in each cycle. Unlike the existing architectures, including the 1-D systolic arrays, the distinction between the horizontal and vertical PEs in this work allows us to perform the calculations independently and in parallel. Thus the proposed architecture is notably faster than the existing designs, as will be discussed later in the simulation results section, Section 4.

Consider that if a match occurs in related computations of the calculation phase for each cell (based on the above-mentioned equations), \( \text{Match\_flag} \) gets one value for the corresponding PE and it will be used later for editing phase. With the given approximations in Equations (2) and (3), cells would most likely have erroneous values. Thus another phase is required, that is, the editing phase to compensate for the introduced errors.

### 3.2 Editing phase

Similar to the calculation phase, the editing phase is also done differently for the cells on the main diagonal and the other cells.

#### 3.2.1 Cells on the main diagonal

For the cells on the main diagonal, our calculation phase was performed based on Equation (1). Consider Figure 2, where a subset of the similarity matrix is shown. Cells \((0, 0)\), \((1, 1)\) are explained here and calculations for other cells on the main diagonal would be just like the cell \((1, 1)\).

**Cell \((0, 0)\)**: The value of this cell is calculated according to (1) by using \( PE_{11} \), see Figure 2, during the calculation phase. Since (1) is the exact equation and the three cells that are used for calculating the value of cell \((0, 0)\) have all exact values (i.e. the initial values), there is no calculation error. Hence, the editing phase is not performed for cell \((0, 0)\).

**Cell \((1, 1)\)**: The value of cell \((1, 1)\) is calculated based on the values of cells \((1, 0)\) and \((0, 1)\), which are computed according to Equations (2) and (3), respectively, and the value of cell \((0, 0)\). There is no error in the value of cell \((0, 0)\) and, therefore, to edit the value of cell \((1, 1)\) we only need to edit the values of cells \((1, 0)\) and \((0, 1)\).
The value of cell (1, 0) is initially computed by ignoring the value of cell (0, 0). Hence, the first step in editing the value of this cell would be figuring out if cell (0, 0) was effective in the value of cell (1, 0). This can be checked with the following condition: $DP_{(0,0)} + T_{(gap)} > DP_{(1,0)}$. If this condition is true, it means that an error has occurred in the calculation of the value of cell (1, 0). As mentioned earlier, we asserted the match score to 1, mismatch score to 0 and gap penalties to −1, therefore, if a match occurs while calculating the value of cell (0, 0), the $Match_{lag}$ of $PE_{(11)}$ is set and this shows that the value of (1, 0) needs some probable correction, see Algorithm 1. Similarly, cell (0, 1) should be checked for possible errors that may happen in some seldom cases. Finally, the value of cell (1, 1) is computed based on Equation (1) to avoid any possible error if $Match_{lag}$ signals for the corresponding PE is set.

The value of the other cells that are not on the main diameter, that is, cell $(i, j)$ for $i > 1$, do not face any error, according to the above descriptions. An illustrative example is provided in Figure 3. This figure shows how a $4 \times 4$ matrix is filled based on the proposed architecture. Note that the calculation and edition phase are completed within one clock cycle and, therefore, a $4 \times 4$ matrix is large-enough to explain the entire process. In fact, for larger matrices, the errors for the cells which are located further away will be completely edited within one clock cycles and, thus, we do not need to consider them in our calculations.

### 3.2.2 Other cells

Unlike the main diagonal, editing the values of the cells which are not located on the main diagonal is necessary. Examples of error editing for the cells that are not on the main diagonal are provided in Algorithms (1) and (2). To make these two algorithms clearer, they are developed based on Figure 2.

---

#### Algorithm 1 Editing phase for vertical cells that are not located on the main diagonal

```plaintext
1: if $Match_{lag}(PE_{i})$ or $Match_{lag}(PE_{i+1})$ or ... or $Match_{lag}(PE_{i+n}) = 1$ then
2: $GRN$ ← greatest PE number that matches
3: $GRV$ ← The value of greatest PE number that matches
4: if $DP_{(i, j−1)} < GRV − (i − GRN)$ then
5: $DP_{(i, j−1)} ← GRV + (i − GRN)$
6: $DP_{(i, j)} = MAX\left\{DP_{(i−1, j−1)} + T_{(Match\_Miss\_match)}$, $DP_{(i−1, j)} + T_{(gap)}\right\}$
7: if $DP_{(i, j)} = DP_{(i−1, j−1)} + T_{(Match)}$ then
8: $Match_{lag}(PE_{i}) ← 1$
9: else
10: $Match_{lag}(PE_{i}) ← 0$
11: end if
12: $Match_{lag}(PE_{i}) ← 0$
13: end if
```

#### Algorithm 2 Editing phase for horizontal cells that are located on the main diagonal

```plaintext
1: if $Match_{lag}(PE_{i})$ or $Match_{lag}(PE_{i+1})$ or ... or $Match_{lag}(PE_{i+n}) = 1$ then
2: $GRN$ ← greatest PE number that matches
3: $GRV$ ← The value of greatest PE number that matches
4: if $DP_{(i−1, j)} < GRV − (j − GRN)$ then
5: $DP_{(i−1, j)} ← GRV + (j − GRN)$
6: $DP_{(i, j)} = MAX\left\{DP_{(i−1, j−1)} + T_{(Match\_Miss\_match)}$, $DP_{(i−1, j)} + T_{(gap)}\right\}$
7: if $DP_{(i, j)} = DP_{(i−1, j−1)} + T_{(Match)}$ then
8: $Match_{lag}(PE_{i}) ← 1$
9: else
10: $Match_{lag}(PE_{i}) ← 0$
11: end if
12: $Match_{lag}(PE_{i}) ← 0$
13: end if
```

### Algorithms (1) and (2)

Algorithms (1) and (2) have the same foundations. However, Algorithm (2) is more detailed as it also calculates the values of the elements on the main diagonal. Thus, this algorithm is chosen in this section to be further explained. The first step is obtaining the value of cell $(i, j)$, that requires an edition on the current value of cell $(i−1, j)$. Note that based on what have been discussed so far, PE $(i−1, j)$ can have a faulty value only if there has been a match in the PE before the PE $(i−1, j)$. Therefore, the address of last PE in which a match has occurred ($GRN$) is required. Let the value of the PE with the last match be $GRV$, if the value of PE $(i−1, j)$ is less than $GRV − (j − GRN)$, we replace $DP$ value of PE $(i−1, j)$ with $GRV − (j − GRN)$, which is a simple comparator and a combinational adder circuit.

After the
edition phase is done, the calculation phase for $DP_{(i,j)}$ can be done following Equation (1). Then $Match_{\text{flag}}$ of the current PE is set if a match occurs in calculation phase. The rest of this algorithm is similar to the calculation phase without editing, that is, when no prior $Match_{\text{flags}}$ in the last cycle are set.

To make Algorithms (1) and (2) clearer, we provide a numerical example of one cell that needs to be edited on Figure 3. Consider the first cycle that PE03 calculated the value of $DP_{(0,3)}$ and as a result, it has gained 47, that is not correct, because one match occurs in cell (0, 0). Now, based on the Algorithm (2) for second cycle, we have to first edit the value of $DP_{(0,3)}$ and then calculate the value of $DP_{(1,3)}$ with PE03 as in the Equation (4).

$Match_{\text{flag}}(PE_{03}) = 1$

\[
\begin{align*}
GRN & = 0 \\
GRV & = 51 \\
DP_{(i-1,j)} & = 47 \\
j & = 3 \\
DP_{(i-1,j)} & < GRV - (j - GRN) \\
& \Rightarrow 47 < 51 - (3 - 0) \\
DP_{(i-1,j)} & \leq GRV - (j - GRN) \\
DP_{(0,3)} & = 48
\end{align*}
\]

where GRN indicates the column number of the cell that has match state. Note that the reason of using this equation $DP_{(i-1,j)} < GRV - (j - GRN)$ is that we assigned gap penalty value of (−1) and if the distance between the cell number that match occurs in its equation and the desired cell become less than the differences between their values, means that an error appearance.

### 3.3 Hardware implementation

In this section, the hardware implementation of the proposed horizontal and vertical PEs are described. Afterwards, the methodology of calculating phase regarding roughly approximated values and error editing phase for each cell are explained.

The execution time for the accelerated phase considers only the kernel execution time, not including data transfer times, as performance in the limiting case will only be determined by the computational part of the Seed Extension. Moreover, to illustrate the relative insignificance to this particular application, total data transfer time excluding the transfer of the reference genome, which is done only once at the start of the program execution, is less than one second in total. Considering that we do not discuss about the memory access of reading query and reference for the first time as the other articles do, and we just focus on the processing time and internal usage of registers for calculating and sharing between...
PEs. The score arrays of each PE will be sent to the local memory for the editing and calculation phases and a proper wiring will give access to this data by other corresponding PEs. It is also worth mentioning that in this manuscript we only consider short reads of up to 150 base pairs, which is the read length of contemporary sequencers, such as the Illumina HiSeq X [16]. The minimum seed length for BWA-MEM is 19 symbols. Therefore, each inexact mapping engine contains 150-19 = 131 symbols, which means, in total, 2 × 131 = 262 PEs are used. This number of PEs need to read 262 characters (A, T, C and G) which are represented by two bits. That means we need to read only 2 × 262 = 524 bits in the worst case, which can be easily done internally. We use shared memory for data exchange between PEs that are calculated in reported time and delay for each matrix calculation. Furthermore, the memory bottleneck is not limited to our work, and 1-D systolic
architectures have to read these 262 characters. Other than the control unit, an important part of the hardware implementation is the design of PEs. As mentioned earlier, two types of PEs are used in the proposed architecture that is, (1) vertical PEs and (2) horizontal PEs. The detailed hardware implementation of these PEs is described below.

### 3.3.1 Vertical PE

A vertical PE does not compute the cells on the main diagonal and, therefore, implements simpler equations. Hence, they are less complex and, of course, faster designs than the horizontal PEs. Figure 4a shows the functionality of the proposed architecture. The proposed architecture is exactly based on the editing and calculating mechanism that was discussed before. First of all, Subtractor0, Subtractor1 and comp0 are responsible for editing phase, based on the Algorithm (1). The signal \( Match_{flag} \) is set to one if the comp1 output is caused of match.

### 3.3.2 Horizontal PE

Unlike vertical PEs, horizontal PEs are further used to compute the cells on the main diagonal. Hence, they are more complex and, consequently, slower designs. The hardware implementation of such a horizontal PE is shown in Figure 4b.

In Figure 4b, there is an addition part in which a counter is used to count each cycle and a comparator that shows if the counter reaches the PE number, this means that the PE has to calculate a cell that is placed on the main diagonal and the scenario changes. At this state, PE doesn't need to edit any values (\( Mux0 \) is added for this sake) but it has to compare three values with each other instead of two and therefore a multiplexer (\( Mux1 \)) is added to do so based on the Algorithm (2). To make both the Figure 4a,b clearer, the same indexing as in Algorithms (1) and (2) is used.

### 4 SIMULATION RESULTS

This section is further divided into two sections. The first section provides the FPGA implementation results of the proposed architecture, while the second section analyses the performance of the proposed architecture in terms of the total execution time.

#### 4.1 Synthesis results

The proposed architecture is implemented in VHDL hardware description language and synthesized on FPGA.

Table 2 reports the resource utilization of the horizontal and vertical PEs when implemented on Virtex-7 and Spartan-7 FPGAs and also, a \( 4 \times 4 \) matrix with seven PE’s.

Our simulation results, show that the proposed architecture can operate at 321 MHz. Note that this is the worst-case scenario, that is, all of the cells have erroneous values and the error editing phase needs to be performed for all of them.

#### 4.2 Performance

In this section, we analyse the worst-case execution time of the proposed architecture. Given that each PE requires one clock cycles to accomplish its computations, the total execution time, \( \tau_{total} \) for a \( (N + 1) \times (N + 1) \) similarity matrix (a seed with the length of \( N \)) can be calculated as:

\[
\tau_{total} = N \times \frac{1}{clk_{horizontal}},
\]

where \( clk_{horizontal} \) is the maximum clock frequency of the horizontal PEs, that is, the slowest part of the proposed architecture. Note that, in a \( (N + 1) \times (N + 1) \) similarity matrix with a seed length of \( N \), the first row and the first column of the matrix are initial values, which are calculated by the SMEM kernel. We need one clock ticks for each PE and all of the matrix elements will be calculated in \( N \) cycles of calculation that is shown in Figure 1. Moreover, Equation (6) shows our speedup in comparison of state-of-the-art 1-D systolic arrays architecture. It is worth mentioning that the numbers 2.9 and 3.1 are the clock periods in (ns) for our work and the implemented work in Ref. [11], respectively. Both of these architectures were implemented on Virtex-6 LX760 FPGA family under the same conditions (timing constraints, etc.).
Compared to the state-of-the-art 1-D systolic array-based architecture that computes a \((N + 1) \times (N + 1)\) matrix in \(2N - 1\) clock cycles, the proposed architecture is faster as it only requires \(N\) clock cycles.

We compare our architecture performance with the original algorithm in terms of execution time on Intel(R) Core(TM) i7-4702 MQ CPU. The BWA-MEM 0.7.11 is used and run over the free NCBI data set [17]. Since we are interested in comparing the execution time of the Seed extension kernel, we only used the output of the SMEM kernel as the input to the proposed design. Note that the speedup in Table 3 is the average speedup over 1000 randomly selected inputs from the data set. The comparison results are provided in Table 3 and Figure 5.

Figure 5a provide a comparison of execution time of our proposed method and 1-D systolic architecture, based on different seed length. As indicated from this figure by the growth of seed length, our execution time decreases asymptotically and for longer seed lengths our acceleration pace is fixed, which this speed-up ratio is illustrated in Figure 5b.

Area overhead of both 1-D systolic arrays and our approach are enlightened herein. Based on Ref. [11], implementation results considering a 131 \(\times\) 131 array on Xilinx Virtex-6 LX760 FPGA show 4% of total FPGA resources, while our approach utilizes almost 13.5% of Xilinx Virtex-6 LX760 FPGA resources that is about 3.3x more area overhead than the 1-D systolic. This increase in area overhead due to addition of editing phase, which let us for the first time eliminate data dependency to calculate Smith–Waterman matrix in a 2-D manner Figure 1. However, area is not usually a bottleneck in implementation of the BWA-MEM algorithm on FPGA and the execution time is a more important concern. The main contribution of our work lays in the fact that we have eliminated the data dependency to devise a new 2-D approach in speeding up the Seed Extension kernel. It is obvious that our proposed architecture can benefit from more parallelism for computing different seeds in parallel, and we will not face lack of FPGA resources.

5 | DISCUSSION

The proposed architecture can be implemented on a Virtex-6 FPGA device with the typical 131 \(\times\) 131 matrix dimension
718x and 1.7x acceleration and speedup in comparison of software execution and the 1-D systolic arrays, respectively in case of the 131 x 131 matrix dimension on Virtex-6 FPGA.

6 | CONCLUSION

This work proposed a novel 2-D, high-throughput FPGA-based seed extension kernel for the well-known BWA-MEM sequence alignment algorithm. In comparison with the Intel(R) Core(TM) i7-4702 MQ CPU, our design can achieve up to 718x speedup and, while operating at 321 MHz as well, we achieved up to 1.7x speed up in comparison of the 1-D systolic arrays that was implemented on the same hardware as our architecture. By introducing the calculation and error editing phase, we notably reduced the number of required clock cycles for filling an \(N \times N\) similarity matrix. In fact, the state-of-the-art 1-D systolic array-based architectures fill the \(N \times N\) similarity matrix in \(2N - 1\) cycles, while our proposed architecture only takes \(N - 1\) cycles to fill the same matrix.

ORCID
Mahdi Taheri  https://orcid.org/0000-0001-5405-992X
Ali Mahani  https://orcid.org/0000-0003-4916-202X

REFERENCES
1. Jung, H., et al.: Tools and strategies for long-read sequencing and de novo assembly of plant genomes. Trends Plant. Sci. 24, 700–724 (2019)
2. Pham-Quoc, C., Kieu-Do, B., Thinh, T.N.: A high-performance FPGA-based bwa-mem dna sequence alignment. Concurr. Comput. Pract. Exp., e5328 (2019)
3. Wu, L., et al.: Fpga accelerated indel realignment in the cloud. In 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 277–290. IEEE (2019)
4. Pham-Quoc, C., Kieu-Do, B., Thinh, T.N.: An fpga-based seed extension ip core for bwa-mem dna alignment. In 2018 International Conference on Advanced Computing and Applications (ACOMP), pp. 1–6. IEEE (2018)
5. Canzar, S., Salzberg, S.L.: Short read mapping: an algorithmic tour. Proc IEEE. 105(3), 436–458 (2015)
6. Fujiki, D., et al.: Genax: a genome sequencing accelerator. In Proceedings of the 45th Annual International Symposium on Computer Architecture, pp. 69–82. IEEE Press (2018)
7. Langmead, B., et al.: Ultrafast and memory-efficient alignment of short dna sequences to the human genome. Genome Biol. 10(3), R25 (2009)
8. Li, H., Durbin, R.: Fast and accurate short read alignment with burrows-wheeler transform. Bioinformatics. 25(14), 1754–1760 (2009)
9. Li, R., et al.: Soap2: an improved ultrafast tool for short read alignment. Bioinformatics. 25(15), 1966–1967 (2009)
10. Li, H.: Aligning sequence reads, clone sequences and assembly contigs with bwa-mem (2013). arXiv preprint arXiv:1303.3997
11. Hougast, E.J., et al.: 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), IEEE pp. 221–227 (2015)
12. Cohen, J: Bioinformaticsâ“ an introduction for computer scientists. ACM Comput. Surv. 36(2), 122–158 (2004)
13. Cao, H., et al.: A short-read multiplex sequencing method for reliable, cost-effective and high-throughput genotyping in large-scale studies. Hum. Mutat. 34(12), 1715–1720 (2013)
14. Smith, T.F., et al.: Identification of common molecular subsequences. J Mol. Biol. 147(1), 195–197 (1981)
15. Hougast, E.J., et al.: Hardware acceleration of bwa-mem genomic short read mapping for longer read lengths. Comput. Biol. Chem. 75, 54–64 (2018)
16. El-Metwally, S., Ouda, O.M., Helmy, S.: Next generation sequencing technologies and challenges in sequence assembly. SpringerBriefs in Systems Biology, vol. 7, XII Springer-Verlag, New York (2014)
17. Wheeler, D.L., et al.: Database resources of the National Center for Biotechnology Information. Nucleic Acids Res. 33(suppl_1), D5–D12 (2006)

How to cite this article: Taheri M, Ansari MS, Magierowski S, Mahani A. Hardware acceleration of the novel two dimensional Burrows-Wheeler Aligner algorithm with maximal exact matches seed extension kernel. IET Circuits Devices Syst. 2021;15:94–103. https://doi.org/10.1049/cds2.12005