electronics

Article

Hardware Acceleration of Identifying Barcodes in Multiplexed Nanopore Sequencing

Wenjie Hu, Yuxin Zhang, Hongrui Zhang and Weigang Chen *

School of Microelectronics, Tianjin University, Tianjin 300072, China
* Correspondence: chenwg@tju.edu.cn; Tel.: +86-138-2121-0869

Abstract: In multiplexed sequencing, the identification of DNA sequencing barcodes can effectively reduce the probability of sample misassignment. However, the great quantity of sequence data requires a high-throughput identification method. Therefore, based on a barcode identification scheme combining cyclic shifting with dynamic programming (DP), this paper proposes, implements and tests a hardware accelerator that can accelerate barcode identification. In the accelerator, considering that the computational complexity of the DP algorithm can be expressed as the multiplication of the lengths of both involved sequences, we design a systolic array structure with simplified processing element (PE) and a parallel circuit architecture to identify the insertion and deletion errors based on the traceback. The accelerator is implemented on a field-programmable gate array (FPGA), and its performance is compared with that of software implemented on a general-purpose computer. The experimental results indicate that, compared with the software implementation, the accelerator can achieve speedups of two orders of magnitude for longer barcodes.

Keywords: multiplexed sequencing; DNA sequencing barcode; dynamic programming; FPGA

Citation: Hu, W.; Zhang, Y.; Zhang, H.; Chen, W. Hardware Acceleration of Identifying Barcodes in Multiplexed Nanopore Sequencing. *Electronics* **2022**, *11*, 2596. https://doi.org/10.3390/electronics11162596

Academic Editors: R. Chad Webb, John A. Rogers and Cunjiang Yu

Received: 5 July 2022
Accepted: 11 August 2022
Published: 19 August 2022

1. Introduction

Next-generation sequencing (NGS) technologies can generate up to one billion reads of DNA sequence per slide, which continuously increase at a rapid pace [1,2]. Multiplexed nanopore sequencing has become an essential strategy to effectively utilize the rising sequencing capacity of NGS technologies [3]. In the multiplexing strategy, specific sample tags, also termed barcodes [4,5], are added to DNA fragments from multiple samples and used to separate sequencing reads belonging to different samples after sequencing. However, DNA barcodes are often corrupted by insertions, deletions, and substitutions during DNA synthesis and sequencing, resulting in sample index misassignments [6]. Therefore, some barcode construction and identification methods with different error correction capacity have been proposed in the past.

Conventional construction and identification methods for DNA barcodes are based on Hamming distance [7,8], Levenshtein distance [9,10], watermark code [11], etc. Recently, in [12], a scheme that combines a pseudorandom sequence with a cyclic block code was proposed to construct barcodes while the cyclic shifting and DP algorithm were used to identify barcodes. This scheme had the advantages of a low complexity and high robustness. However, if we still rely on software tools to realize these approaches, it will fail to meet the high-throughput sequencing requirement due to the fact that more and more sequencing reads need to be sorted and separated [13–15]. Compared to central processing units (CPUs), FPGAs provide a higher bit-level data parallelism and can customize hardware according to the requirements, which leads to a higher computing speed [16,17]. Therefore, taking the scheme proposed in [12] as an example, this paper designs and implements an FPGA-based hardware accelerator.

The cyclic shifting and DP algorithm are the important components of the accelerator. The cyclic shifting can be realized by a shift register in hardware. As a commonly used

algorithm for sequence alignment, the DP algorithm has the computational complexity of the square of the length of the sequences, so it is crucial to accelerate the algorithm [18]. Recently, an FPGA used to realize a linear systolic array (LSA) to accelerate sequence alignment attracted extensive attention, and a series of studies were conducted. The general structures of the LSA were introduced in [19–22], but there were some deficiencies in these works. On one hand, the iterative equation of the sequence alignment algorithm was directly mapped to the PE. On the other hand, the above works did not describe the implementation of the identification of insertions and deletions in detail.

In this paper, a hardware accelerator for a barcode identification algorithm with a high robustness and low complexity is implemented and evaluated. Aiming at the quadratic complexity of the DP algorithm in the identification scheme, we propose an optimization strategy to improve the mapping efficiency of the PE by simplifying the mapping circuit of the iterative equations. Meanwhile, according to the traceback algorithm, a parallel circuit is designed to simultaneously mark and correct insertion and deletion errors on barcodes. The function simulation and performance verification platform are designed and built, and the comparison between a CPU platform and an FPGA acceleration platform is completed. The hardware results indicate that the accelerator has a significantly higher identification speed of sequencing barcodes than the software version of the same algorithm running on the general-purpose computer.

2. Background

This section reviews the related works, including the barcode identification scheme proposed in [12] and the DP algorithm for identifying an insertion/deletion (indel).

2.1. Sequencing Barcode Identification Method

In multiplexed sequencing, DNA barcodes are added to each DNA fragment as special sample tags. After sequencing, sequencing reads belonging to different samples are separated by identifying barcodes. However, the insertions, deletions, and substitutions may occur randomly on barcodes due to the defects in DNA synthesis and sequencing, as shown in Figure 1. NGS technologies itself have error rates of 10^{-3}∼10^{-4} [10]. For nanopore sequencing, the sequencing reads are quite long. Correspondingly, its sequencing error rate is very high (15.1%), of which indels are the main error type (10%) and the rest consists of substitutions (5.1%) [23]. These errors confuse the source of samples, which in turn reduces the accuracy of sequencing. Therefore, DNA barcodes need to be designed to resist insertion, deletion, and substitution errors, which is important for improving the sequencing accuracy of NGS.

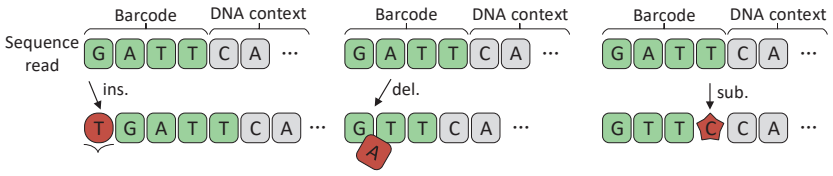

Figure 1. Insertion/deletion/substitution errors on DNA sequencing barcodes.

Aiming at various types of errors introduced during multiplexed nanopore sequencing, a new DNA barcode construction scheme was proposed in [12], which combined a pseudorandom sequence with a cyclic block code and converted them into bases by bit pairs. Here, the pseudorandom sequence acted as a hidden component of sequencing barcodes to identify indels and the cyclic block code was used against the substitution errors. For the barcode generated by the above construction scheme, an identification method using cyclic shifting and DP was proposed. The specific process of decoding barcodes in this scheme is summarized in Figure 2. We took the constructed barcode "GATGCTA" as an example, and it was assumed that a base "C" was inserted into the second position of the barcode and the base "C" at the fifth position of the barcode was deleted during sequencing.

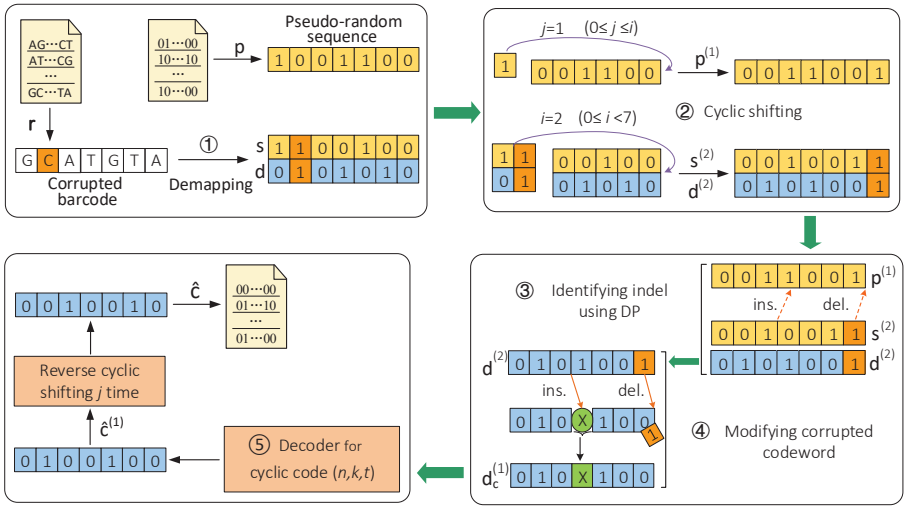

Figure 2. Sequencing barcode identification scheme based on cyclic shifting and DP algorithm [12]. (Reprinted with permission from ref. [12]. Copyright 2022 Springer Nature.)

The decoding process of the corrupted barcode is described below. Specifically, we read the corrupted barcode **r** and the known pseudorandom sequence **p** from the text files as the input of the identification scheme. First, the corrupted barcode **r** is demapped into the corrupted sequence **s** and the corrupted codeword **d**. Subsequently, we found that when **s** and **d** are cyclically shifted to the left $i = 2$ times, the inserted base is moved to the end. When we cyclically shift the pseudorandom sequence **p** to the left $j = 1$ time and use DP to compare the $\mathbf{p}^{(1)}$ and $\mathbf{s}^{(2)}$, it can identify the positions of an indel and modify the codeword $\mathbf{d}^{(2)}$. Then, the modified codeword $\mathbf{d}_c^{(1)}$ is sent to the decoder for decoding. Finally, the decoded codeword $\hat{\mathbf{c}}^{(1)}$ is cyclically reverse-shifted to obtain the final result $\hat{\mathbf{c}}$, which was output to a text file. Using the information bits of the codeword $\hat{\mathbf{c}}$, the sequencing reads are assigned to their respective samples. The above method is summarized in Algorithm 1. (Adapted with permission from ref. [12]. Copyright 2022 Springer Nature.)

Algorithm 1 Barcode identification algorithm in [12]

Input: the corrupted barcode **r** and the predetermined pseudorandom sequence **p**
Output: the decoded codeword $\hat{\mathbf{c}}$

Demap barcode $\mathbf{r} = (r_0, r_1, ..., r_{n-1})$ to sequence $\mathbf{s} = (s_0, s_1, ..., s_{n-1})$ and code $\mathbf{d} = (d_0, d_1, ..., d_{n-1})$;
for $i = 0; i < n; i$++ **do**
  Shift cyclically **s** and **d** to the left i times, denoted as $\mathbf{s}^{(i)}$ and $\mathbf{d}^{(i)}$;
  for $j = 0; j \leq l; j$++ **do**
    Shift cyclically **p** to the left j times, denoted as $\mathbf{p}^{(j)}$
    Identify indel positions in sequence $\mathbf{s}^{(i)}$ using DP
    Modify $\mathbf{d}^{(i)}$ with marked positions, obtaining the modified codeword $\mathbf{d}_c^{(j)}$
    Send $\mathbf{d}^{(i)}$ to cyclic code decoder for decoding, obtaining the codeword $\hat{\mathbf{c}}^{(j)}$
    Calculate the syndrome $\mathbf{S}^{(j)}$ for $\hat{\mathbf{c}}^{(j)}$
    if $\mathbf{S}^{(j)} = 0$ **then**
      Select $\hat{\mathbf{c}}^{(j)}$ as the final decoded result
      Shift cyclically $\hat{\mathbf{c}}^{(j)}$ to the right j times to obtain $\hat{\mathbf{c}}$
      Goto final
    end if
  end for
end for
final

2.2. DP Algorithm for Identifying Indel

The essence of the DP algorithm is to find an optimal path from the target sequence to the reference sequence and judge the position of an indel through this path. The basic process of identifying indels using DP algorithms can be described as follows. First, we build a two-dimensional matrix. Next, we use an iterative equation to calculate the value of the edit distance between the two sequences and store it in the matrix. Then, we use the backtracking algorithm to find an optimal path in the two-dimensional matrix. Finally, we use the optimal path to determine the position of indels in the target sequence.

In the barcode identification scheme, the DP algorithm is exploited to calculate the edit distance and obtain the optimum alignment for the sequences **s** and **p**. Based on the optimum alignment, the positions of insertions and deletions can be determined [24]. It can be seen that the edit distance calculation and the optimum path backtracking are the key links that affect the complexity of the DP algorithm [25].

Assuming that the lengths of two sequences A and B are, respectively, m and n, and the elements in the sequence are, respectively, denoted as $A[i]$ and $B[j]$ ($1 \le i \le m, 1 \le j \le n$), the specific process of using DP to realize the indel identification is as follows.

1. Calculate the edit distance $D(i,j)$ between sequence elements $A[i]$ and $B[j]$; the calculation equation is

$$D(i,j) = \min \begin{cases} D(i-1,j) + w_d, \\ D(i,j-1) + w_i, \\ D(i-1,j-1) + w_z, \end{cases} \quad 0 \le i \le m, 0 \le j \le n. \tag{1}$$

   where $w_d = w_i = 1$ and $w_z = 0$ if $A[i] = B[j]$, otherwise $w_z = 1$ [26]. The initial values of the iterative equation are given by $D(i,0) = i$ and $D(0,j) = j$ for $0 \le i \le m$ and $0 \le j \le n$.
2. Backtrack from $D(m,n)$ to $D(0,0)$ to find the optimum path $\mathbf{I_{path}}(i,j)$, which can be expressed as

$$\mathbf{I_{path}}(i,j) = \begin{cases} (i-1,j), & \min = D(i-1,j). \\ (i,j-1), & \min = D(i,j-1). \\ (i-1,j-1), & \min = D(i-1,j-1). \end{cases} \tag{2}$$

3. Use the optimum path $\mathbf{I_{path}}(i,j)$ to get the vector $\mathbf{I_{mark}}(i,j)$ that marks the indel positions, as follows

$$\mathbf{I_{mark}}(i,j) = \begin{cases} 1, & \mathbf{I_{path}}(i,j) = (i-1,j)(\text{deletion}). \\ 2, & \mathbf{I_{path}}(i,j) = (i,j-1)(\text{insertion}). \\ 3, & \mathbf{I_{path}}(i,j) = (i-1,j-1)(\text{mis/match}). \end{cases} \tag{3}$$

4. Based on the marker vector $\mathbf{I_{mark}}(i,j)$, the insertion and deletion errors on the corrupted sequence can be corrected.

Observing the above iterative equation for calculating edit distance, we can see that the time and space complexity of the DP algorithm are both $O(mn)$. The general method to reduce the time complexity of the algorithm is to establish the LSA structure [27,28]. Since the performance of the LSA mainly depends on the number of PEs and the clock frequency of the PE [29], a simplified design of the PE can improve the performance of the LSA to a certain extent.

3. Barcode Identification Accelerator

Based on the barcode identification method proposed in [12], the overall architecture of the hardware accelerator designed in this paper is illustrated in Figure 3. The accelerator consists of several modules, including demapping, cyclic shifting, dynamic programming, cyclic code decoder, and so on. The input of the accelerator includes an n-bit predetermined

pseudorandom sequence **p** and a 2*n*-bit corrupted barcode **r** converted from *n*-nt bases (strings), and the output is the final result $\hat{\mathbf{c}}$ obtained by cyclically shifting the decoded codeword $\hat{\mathbf{c}}^{(k)}$ to the right *k* times.

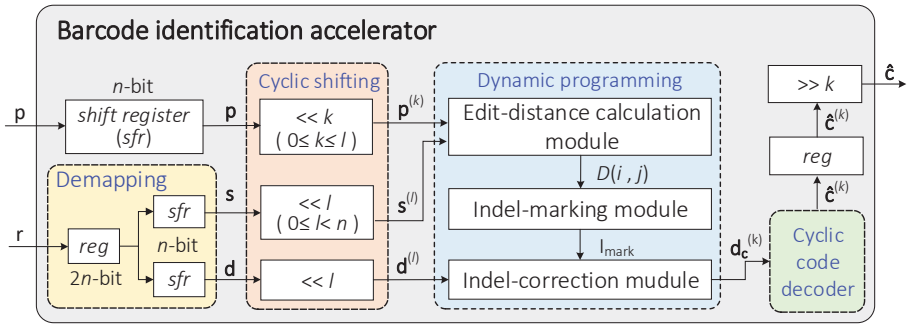

Figure 3. Hardware architecture of the sequencing barcode identification accelerator.

The specific process of hardware implementation is explained as follows. The working state of each module is controlled by a finite state machine.

1. The 2*n*-bit barcode $\mathbf{r} = (s_0d_0, s_1d_1, ..., s_{n-1}d_{n-1})$ is transformed into the sequence $\mathbf{s} = (s_0, s_1, ..., s_{n-1})$ and code $\mathbf{d} = (d_0, d_1, ..., d_{n-1})$ through the demapping module;
2. In the cyclic shifting module, it is assumed that the code **d** and the sequence **s** have been cyclically shifted to the left *l* times ($0 \leq l < n$) to obtain the sequence $\mathbf{s}^{(l)}$ and codeword $\mathbf{d}^{(l)}$. Meanwhile, the predetermined sequence **p** has been cyclically shifted to the left *k* times ($0 \leq k \leq l$) to obtain the sequence $\mathbf{p}^{(k)}$;
3. The shifted sequences $\mathbf{s}^{(l)}$, $\mathbf{p}^{(k)}$, and code $\mathbf{d}^{(l)}$ are input into the dynamic programming module. Through the processing of this module, the indel identification of code $\mathbf{d}^{(l)}$ is realized by calculating the edit distance of the sequences **s** and **p** and backtracking the optimal path, and we get the modified codeword $\mathbf{d}_\mathbf{c}^{(k)}$;
4. Subsequently, the remaining substitution errors on the codeword $\mathbf{d}_\mathbf{c}^{(k)}$ are decoded by the cyclic code decoder. If the decoded codeword $\hat{\mathbf{c}}^{(k)}$ meets the condition, that is, all syndromes equal to zero, it is cyclically shifted to the right *k* times as the final output result. Otherwise, the accelerator returns to the cyclic shifting module.
5. In the cyclic shifting module, if $(k+1) < l$, we cyclically shift the sequence $\mathbf{p}^{(k)}$ to the left once, and then perform indel identification. When $(k+1) = l$, if $(l+1) < (n-1)$, the sequence $\mathbf{s}^{(l)}$ and the codeword $\mathbf{d}^{(l)}$ are cyclically shifted to the left once, and then indel identification is performed; otherwise, the accelerator directly outputs the final decoding result $\hat{\mathbf{c}}$.

Considering that the computational complexity of the DP algorithm in this paper is $O(n^2)$, with the increase of the number of cyclic shifts, the overall operation efficiency of the barcode identification algorithm will be seriously affected. Therefore, the hardware acceleration of the dynamic programming module is described in detail in this paper. Specifically, the hardware circuit of this module includes three submodules: (a) an edit-distance calculation module; (b) an insertion-and-deletion-errors-marking module, called indel-marking module; and (c) an insertion-and-deletion-errors-correction module, called indel-correction module.

3.1. Dynamic Programming Module

This section details the hardware acceleration of the dynamic programming module, mainly including an LSA structure with the simplified PE and a parallel circuit for identifying insertion and deletion errors.

3.1.1. Edit-Distance Calculation Module

To realize hardware acceleration of the DP algorithm, an LSA structure was constructed to implement the parallel computation of the edit distance between the predetermined sequence $\mathbf{p}^{(k)}$ and the corrupted sequence $\mathbf{s}^{(l)}$. All of the edit distance values $D(i,j)$ can be stored in a two-dimensional array of size $(n+1)(n+1)$. Figure 4 describes the LSA structure constructed in this paper. The number of PE in the LSA is determined by the length n of the pseudorandom sequence $\mathbf{p}$. The $n+1$ PEs are placed horizontally, and each PE is responsible for calculating a column of elements in the two-dimensional array.

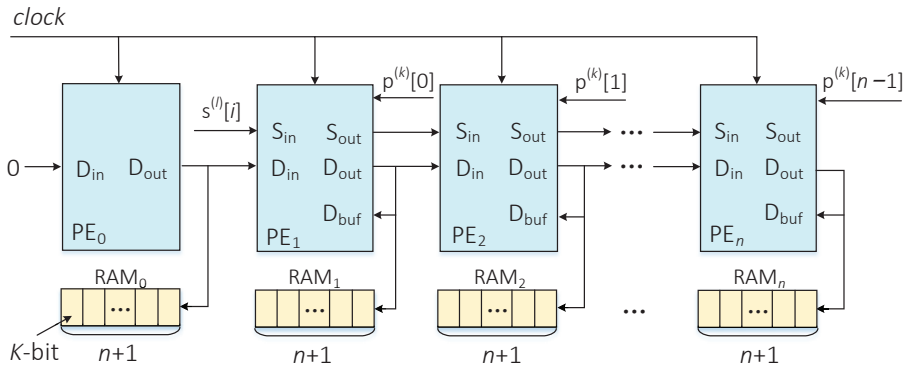

Figure 4. LSA structure for parallel calculation of edit distances.

PE_0 is responsible for the calculation of the first column element in the two-dimensional array, and its accumulation operation can be implemented with a simple accumulator. In the first clock cycle, an initial value is assigned to the D_{buf} port of each PE, which comes from the corresponding first row element in the two-dimensional array. The start and end signals of two adjacent PEs are always separated by one clock cycle. In each clock cycle, all PEs calculate the edit distance in parallel. After $2n$ clock cycles, the minimum edit distance $D(n,n)$ is output by PE_n.

The architecture of the PEs except PE_0 is illustrated in Figure 5. In previous works, researchers directly mapped the iterative equation to PE [19,20]. Such a PE is shown in Figure 5a. Obviously, the directly mapped PE contains three addition operations and two comparison operations. On this basis, we propose a simplified PE architecture, as depicted in Figure 5b. The design of this architecture is also based on Formula (1), but the mapping circuit of the equation is simplified. Specifically, we adjust the execution order of the comparison operation and the addition operation, thereby reducing the number of adders in the circuit when directly mapping the iterative equation and improving the mapping efficiency of the PE. The redesigned PE architecture first performs the comparison operation between $D(i-1,j)$ and $D(i,j-1)$, then performs the comparison operation between $Yout$ and $Xout$, and finally outputs the minimum value $D(i,j)$.

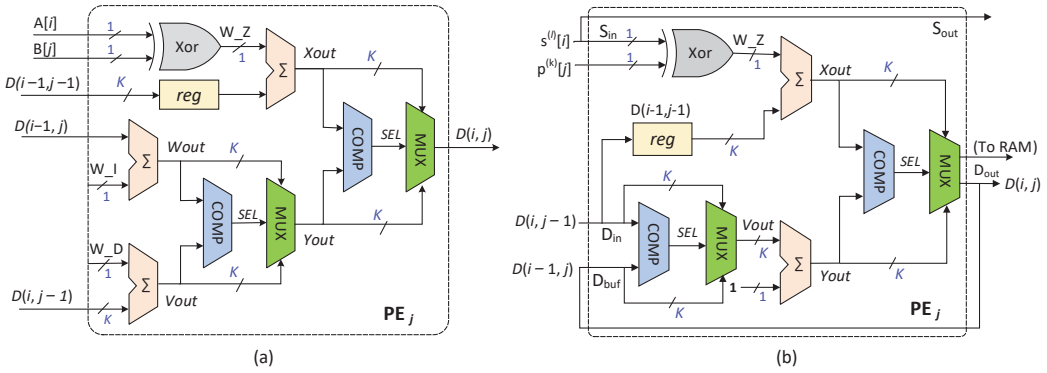

Figure 5. Details of the PEs except PE_0: (**a**) directly mapped PE; (**b**) simplified PE.

Figure 5b describes the PE_j responsible for calculating the editing distance of the elements in the jth column. Two sequences are simultaneously sent to PE_j for alignment, and all the needed data for the PE_j come from itself and other PEs in the systolic array. The details are as follows:

- PE_1 sequentially receives $\mathbf{s}^{(l)}[i]$ in every clock cycle and sends it to PE_2 in the next clock cycle. Similarly, PE_j receives the 1-bit $\mathbf{s}^{(l)}[i]$ from PE_{j-1} and outputs it to PE_{j+1} in the following clock cycle;
- The elements in every column correspond to one bit, respectively, in the sequence $\mathbf{p}^{(k)}$. In other words, as far as PE_j is concerned, the corresponding bit in the sequence (i.e., $\mathbf{p}^{(k)}[j]$) is constant, and it is only read at the beginning of the process;
- At the tth clock cycle, the input $D(i-1,j)$ of PE_j is the output result of the same PE_j in the $(t-1)$th clock cycle. Moreover, the input $D(i,j-1)$ and $D(i-1,j-1)$ of PE_j are the output results of the PE_{j-1} in the $(t-1)$th and $(t-2)$th clock cycles, respectively;
- The calculated result $D(i,j)$ is not only the input of PE_{j+1}, but also stored in a random access memory (RAM) with a width of K bits and depth of $(n+1)(n+1)$, where n is the sequence's length, and it satisfies $2^K \geq n$.

3.1.2. Indel-Marking Module

According to the intermediate calculation results stored in RAM, the indel-marking module searches the optimum path by traceback and marks the position where the indel appears on the sequence $\mathbf{s}^{(l)}$. Contrary to the process of calculating the edit distance, the traceback needs to find the optimal path starting from the lower right corner element of the two-dimensional array. The execution time of this module is related to the length n of the pseudorandom sequence and the number N of indel errors. Since each search for a traceback point takes one clock cycle, the number of clocks required to find the entire traceback path is $T = n + N$.

The hardware architecture of the indel-marking module is shown in Figure 6, and the implementation of this architecture is divided into the following three steps. First, starting from the RAM$[n][n]$ and taking the RAM$[i][j]$ as the reference element, the minimum value of the three elements is selected in the order from the upper left(RAM$[i-1][j-1]$), left(RAM$[i-1][j]$), to top(RAM$[i][j-1]$). Then, the insertion, deletion, and substitution or correct transmission are detected according to the current position where the minimum value is located. Finally, the element with the minimum value is taken as the next reference cell, and the previous two steps are repeated until the traversal reaches RAM[0][0].

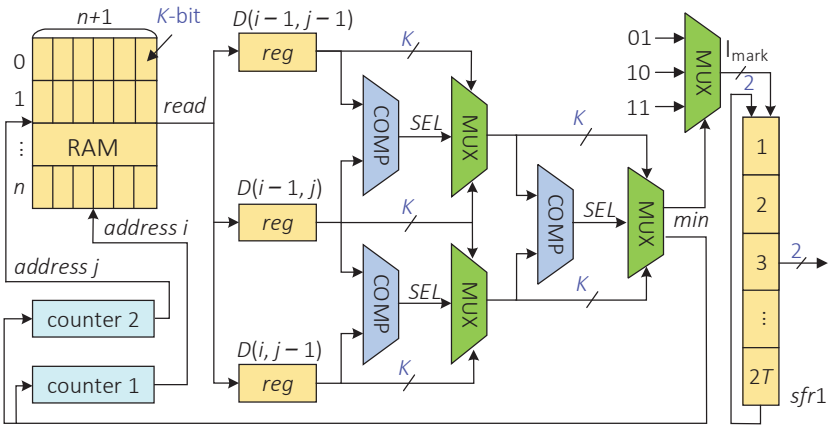

Figure 6. The hardware architecture of the indel-marking module.

Specifically, the rules of the second step are described as follows.

- If $D(i, j-1)$ is the minimum, it means that an insertion error appears in the sequence $\mathbf{s}^{(l)}$ at the position $i-1$ to i. In this case, the shift register *sfr*1 writes 2-bit data "01" in the upper direction;
- If $D(i-1, j)$ is the minimum, it indicates that a deletion error appears in the sequence $\mathbf{s}^{(l)}$ at the position $i-1$ to i. In this case, the shift register *sfr*1 writes 2-bit data "10" in the upper direction;
- If $D(i-1, j-1)$ is the minimum, it means that the data are transmitted correctly or there is a substitution error in the sequence $\mathbf{s}^{(l)}$ at position $i-1$. In this case, the shift register *sfr*1 writes 2-bit data "11" in the upper direction.

The traceback process finishes after T clock cycles. At this time, the shift register *sfr*1 stores $2T$-bit data. These data contain the position information of insertions and deletions.

3.1.3. Indel-Correction Module

The indel-correction module corrects the insertion and deletion errors on the corrupted codeword $\mathbf{d}^{(l)}$ according to the error locations provided by the indel-marking module. Therefore, the indel-marking module and the indel-correction module can be implemented in parallel on hardware. Figure 7 describes the hardware implementation architecture of the indel-correction module. The locations of the insertion and deletion errors on the sequence $\mathbf{s}^{(l)}$ are stored in the shift register *sfr*1, and the corrupted codeword $\mathbf{d}^{(l)}$ is stored in the shift register *sfr*2.

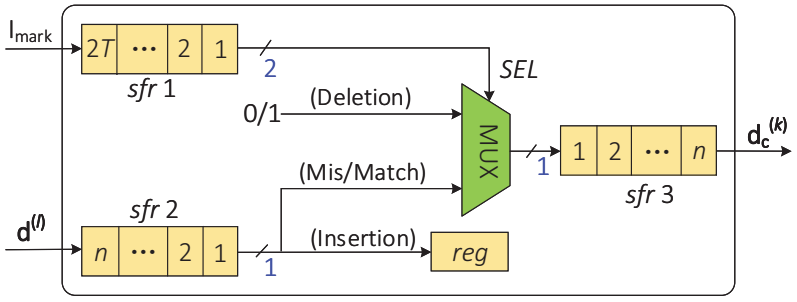

Figure 7. The hardware architecture of the indel-correction module.

At the same clock pulse, if the output data of the *sfr*1 are "11", *sfr*2 will shift out one bit of the data and these data will be written to the high position of the shift register *sfr*3. If the output data element of the *sfr*1 are "10", *sfr*2 will stay still, and random data (i.e., "0" or "1") will be written into the *sfr*3. If the output data of *sfr*1 are "01", *sfr*2 will shift out one bit of the data to an additional register and *sfr*3 will stay still. After T clock cycles, the piece of data stored in *sfr*3 is the corrected codeword $\mathbf{d}_c^{(k)}$.

For the dynamic programming module, the total number of clock cycles required to calculate edit distance, mark indels, and correct indels is $2n + T$. Compared with the serial computation amount of $n^2 + 2T$ in the software implementation, the computing time is greatly shortened.

3.2. Cyclic Code Decoder

The cyclic code decoder is another key error correction module in the accelerator. Due to the use of cyclic block codes in barcodes, the remaining substitution errors can be corrected. In this paper, the Bose–Chaudhuri–Hocquenghem (BCH) codes consistent with the software program were chosen as the coding scheme. BCH codes have been proven to be excellent error-correcting codes with relatively strong error-correcting capability, which are simple to encode and decode [30]. Figure 8 shows the overall architecture of a BCH decoder, which mainly includes three blocks, namely, the syndrome calculation block, the key equation solver(KES) block and the Chien search block.

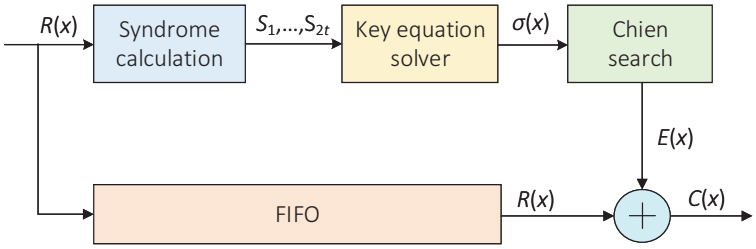

Figure 8. The architecture of the BCH decoder.

In the accelerator, the BCH decoder is responsible for correcting the remaining substitution errors on the codeword $\mathbf{d}_{\mathbf{c}}^{(k)}$. In the meantime, the syndrome output by the syndrome calculation block can be used to judge whether the codeword is correct. If all the syndromes are zero, the judgment condition is satisfied and the codeword is error-free; otherwise, it is not satisfied. The most important and complicated part of the BCH decoder is the KES block. It requires a lot of hardware resources to calculate the error locator polynomial. In this paper, the inversionless Berlekamp–Massey (IBM) algorithm [31] was adopted as the iterative decoding algorithm because it had a relatively low hardware implementation complexity. Meanwhile, its execution speed and device utilization can be improved by parallel processing methods. Thus, this paper adopted and implemented a BCH decoder with a high-speed parallel architecture [32].

4. Results

This section focuses on the implementation results of the accelerator. We comprehensively evaluate the accelerator's function and performance from three aspects: robustness, resource utilization, and acceleration effect.

4.1. Performance Verification

To verify the accuracy and robustness of the barcode identification accelerator, the same noisy channel model was adopted in hardware. It included two different error scenarios (random scenario and context scenario) and a simplified channel model of insertion/deletion/substitution (IDS) [33,34].

The two scenarios were generalized as follows. For the random scenario, the same number of insertion and deletion occurred on the barcode, so the length of the barcode was unchanged. For the context scenario, the number of insertions or deletions that occurred on the barcode varied, and the length of the barcode was thus changed [6]. To simulate various errors introduced on the barcode, the sequencing process was modeled as a noisy channel model similar to that in communication systems. In this channel, barcodes were inserted, deleted, and substituted with a certain probability. The channel was described by three parameters: insertion probability P_i, deletion probability P_d, and substitution probability P_s. We validated the robustness of the barcode identification scheme by estimating the identification error rate P_e (i.e., the probability that a corrupted barcode is decoded incorrectly).

The software simulation was realized by using the software program in [12]. This program includes the generation of a barcode, the setting of the sequencing channel and the identification of the barcode. In the software program, cyclic codes and predetermined pseudorandom sequences were combined bit by bit to form sequencing barcodes. After the generated barcodes passed through the noisy channel, the corrupted barcodes were obtained. In the sequencing channel, we could choose different error scenarios and set different mutation probability of barcodes. Taking the corrupted barcodes as input, the software simulation and hardware implementation of the barcode identification were performed.

First, we investigated the identification error rate P_e of the hardware implementation and software simulation for the 15-nt sequencing barcode constructed with BCH(15, 5, 3) in the two different scenarios. For each scenario, 200 barcodes were randomly selected from the relevant database for validation. The mutation probability of each base on the barcode

was $P_{\text{mut}} \in [0.12, 0.33]$ $(P_i = P_d = P_s)$, where $P_{mut} = P_i + P_d + P_s$. The simulation results are shown in Figure 9. The experimental results show that the hardware implementation results are consistent with the software simulation results regardless of the random scenario or the context scenario. Therefore, it is verified that the accelerators can be applied to two different error scenarios and achieve the same performance as the software simulation for different error scenarios.

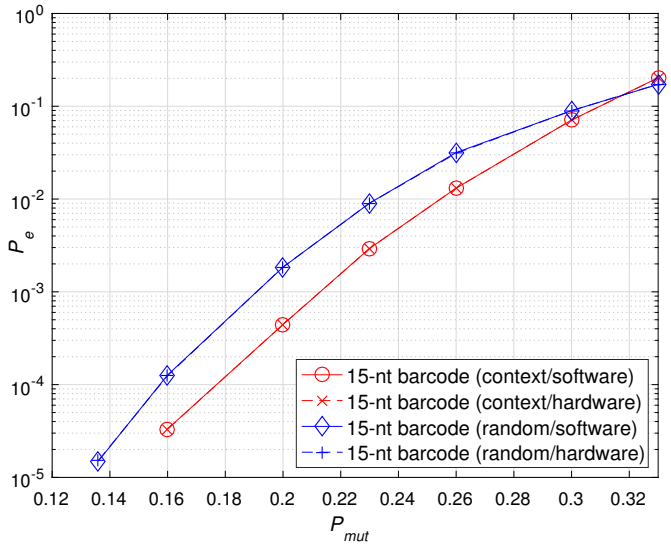

Figure 9. Performance verification of software and hardware implementations of sequencing barcodes under different error scenarios.

Then, to verify the robustness of the hardware accelerators, two types of barcodes were constructed by BCH codes with the same error correction ability but different lengths. Specifically, the 31-nt barcodes were constructed with BCH(31,16,3) and the 15-nt barcodes were constructed with BCH(15,5,3). For each length, 200 barcodes were randomly selected from the relevant database for hardware and software validation. The simulation experiment was performed under the same error scenario. Figure 10 shows the identification error rate P_e of the sequencing barcodes implemented by hardware and software in the random scenario under different mutation probabilities $P_{\text{mut}} \in [0.1, 0.33]$ $(P_i = P_d = P_s)$. It is observed that the hardware accelerators built for barcodes of different lengths can also achieve the same robustness as that of software simulations under the same error scenario.

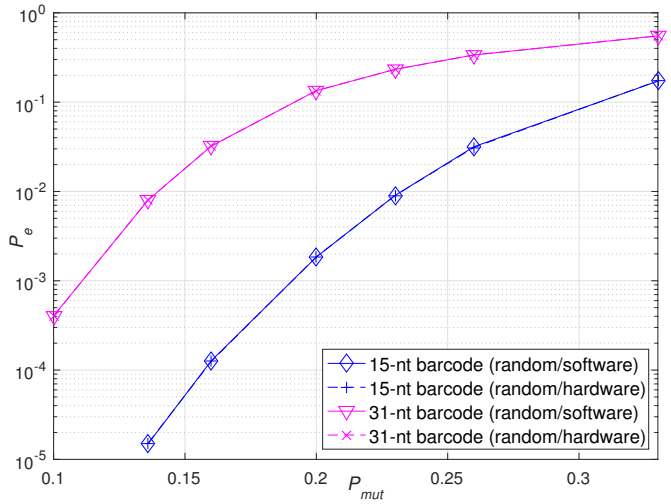

Figure 10. Performance verification of software and hardware implementations of sequencing barcodes with different lengths under the same error scenario.

4.2. Hardware Implementation Results

We wrote Verilog codes for accelerators with barcode lengths of 12-nt, 15-nt, and 31-nt. All designs were mapped on a Virtex-6 XC6VLX240T FPGA using the Xilinx ISE 14.7 development suite. This family was only used because of its availability and the fact that we did not need to change the core code to use different chips. Using larger or newer FPGAs should enable longer barcode identification and yield better results. The accelerator took the FPGA chip as the core and realized the communication between a host CPU and the FPGA through the PCIe. The PCIe interface controller was implemented based on Xilinx PCIe IP. On the CPU side, corrupted barcodes and pseudorandom sequences were transferred from the main memory via direct memory access (DMA). On the FPGA side, two asynchronous FIFOs were used to buffer the input data.

Taking the 12-nt, 15-nt, and 31-nt sequencing barcodes constructed from BCH(12, 4, 2), BCH(15, 5, 3), and BCH(31, 16, 3) codes as an example, we observed the implementation results of the three accelerators on the FPGA. Among them, BCH(12, 4, 2) was a shorted cyclic BCH code, which was generated using BCH(15,7,2). The overall on-chip resource utilization and maximum clock frequency of the three accelerators are listed in Table 1. It is seen that each of these three types of accelerators occupies a small amount of logic resources of the FPGA, and the maximum frequency of the accelerator designed for barcodes of length 15 can reach 228.7 MHz. However, it is apparent that for larger barcodes, the FPGA resources consumed by the accelerator increase significantly.

Table 1. The implementation results of the three accelerators.

Barcode Length	Registers	LUTs	Slices	f_{max}/MHz
12-nt	1084 (1%)	2676 (1%)	983 (2%)	226.4
15-nt	1860 (1%)	4583 (3%)	1366 (3%)	228.7
31-nt	7093 (2%)	24,712 (16%)	8171 (21%)	198.6

In addition, we compared the runtime of software and hardware for barcodes identification. To obtain the runtime of the pure software implementation, the original software code was run on a computer configured with an Intel Core i7-10700 microprocessor (eight-core, sixteen-thread, 4.8 GHz) and main memory with a capacity of 16 GB. The software solution came from the software program in [12], which compiles and runs directly in Visual Studio 2019. The hardware execution time was the time from the hardware startup to the output of the decoded result to the memory. The time required by the CPU and FPGA to identify barcode sequences of different lengths is listed in Table 2, where the FPGA clocked at 100 MHz. For each length, 200,000 barcodes with the mutation probability $P_{\text{mut}} \in [0.12, 0.33]$ $(P_i = P_d = P_s)$ were randomly selected from the relevant database for identification. To improve the accuracy of the results, the identification of each length was performed three times and the running time was averaged.

Table 2. Comparison of the runtime for different lengths of sequencing barcodes identification based on CPU and FPGA.

Barcode Length	BCH Code	CPU	FPGA	Speedup
12-nt	(12,4,2)	4 m 30.3 s	3.9 s	69×
15-nt	(15,5,3)	6 m 54.5 s	5.3 s	78×
31-nt	(31,16,3)	54 m 16.1 s	17.6 s	183×

It can be observed in Table 2 that the accelerator is approximately 70–180× faster than the C implementation for representative input sizes, and this performance improvement increases with the length of the barcode. As a result, for longer barcodes, hardware-based accelerators outperform the software by two orders of magnitude in terms of the identification speed.

It was also important to compare the current implementation with the methods described in other literature works. In multiplex sequencing, the traditional DNA barcode design and identification methods are mostly realized by software. For example, Costea et al. [35] reported a fully customizable, fast, and accurate software package called TagGD (Tag Generator and Demultiplexer); Tambe et al. [36] proposed a software tool for identifying and correcting barcodes in single cell genomics, etc. The above methods were compared with the work in this paper, as shown in Table 3. Since different platforms use different processors, algorithms, and datasets, the number of barcode reads that can be processed per second was used to evaluate the processing speed of different software and hardware solutions. It should be noted that for a given method, the longer the barcodes are, the longer the processing time is required. The comparison shows that although the barcodes identified by our hardware accelerator are the longest, the processing speed of the barcodes is the fastest.

Table 3. Comparison with other previous approaches.

Approach	Processor Type	Barcode Length	Number of Barcodes	Time	Processing Speed (reads/s)
Tambe et al. (2019) [36]	CPU	12	100,000	6 m 39 s	251
Costea et al. (2013) [35]	CPU	18	200,000	27.8 s	7194
Our Work	FPGA	31	200,000	17.6 s	11,363

5. Conclusions

In this paper, we presented the design and implementation of an FPGA-based hardware accelerator for a highly robust sequencing barcode identification algorithm. Considering that the DP algorithm in the identification scheme had the time complexity of $O(n^2)$, a systolic array structure was proposed, which improved the mapping efficiency of a PE by simplifying the circuit structure when the iterative equation was mapped to the PE. Moreover, according to the two-dimensional array stored in RAM, a parallel circuit using traceback to identify insertion and deletion errors was designed. The simulation results on an FPGA indicated that the accelerators constructed for sequencing barcodes with different lengths could achieve a speedup of about 70–180$\times$ compared to the general-purpose computer platform. Our future work will mainly focus on the hardware acceleration of sequencing barcode identification algorithms based on graphic processing unit (GPU).

Author Contributions: Conceptualization, W.C. and W.H.; methodology, W.H.; software, W.H.; validation, W.H.; formal analysis, W.C.; investigation, W.C., H.Z. and Y.Z.; writing—original draft preparation, W.C. and W.H.; writing—review and editing, W.C., W.H., Y.Z. and H.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data presented in this study are available on request from the corresponding author. The data are not publicly available due to privacy restrictions.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

FPGA	Field-programmable gate array
NGS	Next-generation sequencing
DNA	Deoxyribonucleic acid
CPU	Central processing unit
DP	Dynamic programming
PE	Processing element
LSA	Linear systolic array
RAM	Random-access memory

BCH	Bose–Chaudhuri–Hocquenghem
KES	Key equation solver
IBM	Inversionless Berlekamp–Massey
GPU	Graphics processing unit

References

1. Hardwick, S.A.; Deveson, I.W.; Mercer, T.R. Reference standards for next-generation sequencing. *Nat. Rev. Genet.* **2017**, *18*, 473–484. [CrossRef] [PubMed]
2. Hu, T.; Chitnis, N.; Monos, D.; Dinh, A. Next-generation sequencing technologies: An overview. *Hum. Immunol.* **2021**, *82*, 801–811. [CrossRef] [PubMed]
3. Krishnan, A.R.; Sweeney, M.; Vasic, J.; Galbraith, D.W.; Vasic, B. Barcodes for DNA sequencing with guaranteed error correction capability. *Electron. Lett.* **2011**, *47*, 236–237. [CrossRef]
4. Abeynayake, S.W.; Fiorito, S.; Dinsdale, A.; Whattam, M.; Crowe, B.; Sparks, K.; Campbell, P.R.; Gambley, C. A rapid and cost-effective identification of invertebrate pests at the borders using MinION sequencing of DNA barcodes. *Genes* **2021**, *8*, 1138. [CrossRef] [PubMed]
5. Chen, W.; Han, M.; Zhou, J.; Ge, Q.; Wang, P.; Zhu, S.; Song, L.; Yuan, Y. An artificial chromosome for data storage. *Natl. Sci. Rev.* **2021**, *8*, nwab028–nwab037. [CrossRef]
6. Chen, W.; Wang, L.; Han, M.; Han, C.; Li, B. Sequencing barcode construction and identification methods based on block error-correction codes. *Sci. China Life Sci.* **2020**, *63*, 1580–1592. [CrossRef]
7. Hamady, M.; Walker, J.J.; Harris, J.K.; Gold, N.J.; Knight, R. Error-correcting barcoded primers for pyrosequencing hundreds of samples in multiplex. *Nat. Methods* **2008**, *5*, 235–237. [CrossRef]
8. Bystrykh, L.V. Generalized DNA barcode design based on hamming codes. *PLoS ONE* **2012**, *7*, e36852. [CrossRef]
9. Buschmann, T.; Bystrykh, L.V. Levenshtein error-correcting barcodes for multiplexed DNA sequencing. *BMC Bioinform.* **2013**, *14*, 272–281. [CrossRef]
10. Hawkins, J.A.; Jones, S.K.; Finkelstein, I.J.; Press, W.H.; Callan, C.G.; Troyanskaya, O.G.; Weissman, J.S. Indel-correcting DNA barcodes for high-throughput sequencing. *Proc. Natl. Acad. Sci. USA* **2018**, *115*, e6217–e6226. [CrossRef]
11. Kracht, D.; Schober, S. Insertion and deletion correcting DNA barcodes based on watermarks. *BMC Bioinform.* **2015**, *16*, 50. [CrossRef] [PubMed]
12. Chen, W.; Wang, P.; Wang, L.; Zhang, D.; Han, M.; Han, M.; Song, L. Low-complexity and highly robust barcodes for error-rich single molecular sequencing. *3 Biotech* **2021**, *11*, 78. [CrossRef] [PubMed]
13. Goenka, S.D.; Gorzynski, J.E.; Shafin, K.; Fisk, D.G.; Pesout, T.; Jensen, T.D.; Monlong, J. Accelerated identification of disease-causing variants with ultra-rapid nanopore genome sequencing. *Nat. Biotechnol.* **2022**, *40*, 1035–1041. [CrossRef] [PubMed]
14. Gorzynski, J.E.; Goenka, S.D.; Shafin, K.; Jensen, T.D.; Fisk, D.G.; Grove, M.E. Ultrarapid nanopore genome sequencing in a critical care setting. *N. Engl. J. Med.* **2022**, *386*, 700–702. [CrossRef]
15. Pomerantz, A.; Peñafiel, N.; Arteaga, A.; Bustamante, L.; Pichardo, F.; Coloma, L.A.; Prost, S. Real-time DNA barcoding in a rainforest using nanopore sequencing: Opportunities for rapid biodiversity assessments and local capacity building. *Gigascience* **2018**, *7*, giy033. [CrossRef]
16. Awad, M. FPGA supercomputing platforms: A survey. In Proceedings of the International Conference on Field Programmable Logic, Prague, Czech Republic, 31 August–2 September 2009; pp. 564–568.
17. Leiserson, C.E.; Thompson, N.C.; Emer, J.S.; Kuszmaul, B.C.; Tao, B.S. There is plenty of room at the top: What will drive computer performance after Moore's law. *Science* **2020**, *368*, e9744. [CrossRef]
18. Stivala, A.; Stuckey, P.J.; Banda, M.G.D.L.; Hermenegildo, M.; Wirth, A. Lock-free parallel dynamic programming. *J. Parallel Distrib. Comput.* **2010**, *70*, 839–848. [CrossRef]
19. Guo, X.; Hong, W.; Devabhaktuni, V. A systolic array-based FPGA parallel architecture for the BLAST algorithm. *ISRN Bioinform.* **2012**, *2012*, 195658. [CrossRef]
20. Casale-Brunet, S.; Bezati, E.; Mattavelli, M. Design space exploration of dataflow-based Smith-Waterman FPGA implementations. In Proceedings of the 2017 IEEE International Workshop on Signal Processing Systems (SiPS), Lorient, France, 3–5 October 2017; pp. 1–6.
21. Shah, H.A.; Hasan, L.; Koo, I. Optimized and portable FPGA-based systolic cell architecture for Smith-Waterman-based DNA sequence alignment. *J. Inf. Commun. Converg. Eng.* **2016**, *14*, 26–34. [CrossRef]
22. Koliogeorgi, K.; Voss, N.; Fytraki, S.; Xydis, S.; Soudris, D. Dataflow acceleration of Smith-Waterman with traceback for high throughput next generation sequencing. In Proceedings of the 2019 29th International Conference on Field Programmable Logic and Applications (FPL), Barcelona, Spain, 8–12 September 2019; pp. 74–80.
23. Jain, M.; Koren, S.; Miga, K.H.; Quick, J.; Rand, A.C.; Sasani, T.A. Nanopore sequencing and assembly of a human genome with ultra-long reads. *Nat. Biotechnol.* **2018**, *36*, 338–345. [CrossRef]
24. Kruskal, J.B. An overview of sequence comparison: Time warps, string edits, and macromolecules. *SIAM Rev.* **1983**, *25*, 201–237. [CrossRef]
25. Nawaz, Z.; Nadeem, M.; Someren, H.V.; Bertels, K. A parallel FPGA design of the Smith-Waterman traceback. In Proceedings of the 2010 International Conference on Field-Programmable Technology, Beijing, China, 8–10 December 2010; pp. 454–459.

26. Tithi, J.J.; Crago, N.C.; Emer, J.S. Exploiting spatial architectures for edit distance algorithms. In Proceedings of the 2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Monterey, CA, USA, 23–25 March 2014; pp. 23–34.
27. Benkrid, K.; Liu, Y.; Benkrid, A.S. A highly parameterized and efficient FPGA-based skeleton for pairwise biological sequence alignment. *IEEE Trans. Very Large Scale Integr. VLSI Syst.* **2009**, *17*, 561–570. [CrossRef]
28. Rucci, E.; Garcia, C.; Botella, G.; Giusti, A.D.; Naiouf, M.; Prieto-Matias, M. Swifold: Smith-Waterman implementation on FPGA with OpenCL for long DNA sequences. *BMC Syst. Biol.* **2018**, *12*, 96. [CrossRef] [PubMed]
29. Gok, M.; Yilmaz, C. Efficient cell designs for systolic Smith-Waterman implementations. In Proceedings of the 16th International Conference on Field Programmable Logic and Applications, Madrid, Spain, 28–30 August 2007; pp. 1–4.
30. Reed, I.S.; Shih, M.T. VLSI design of inverse-free Berlekamp-Massey algorithm. *IEE Proc. E (Comput. Digit. Tech.)* **1991**, *138*, 295–298. [CrossRef]
31. Chen, C.; Han, Y.S.; Wang, Z.; Bai, B. A new inversionless Berlekamp-Massey algorithm with efficient architecture. In Proceedings of the 2019 IEEE International Workshop on Signal Processing Systems (SiPS), Nanjing, China, 20–23 October 2019; pp. 48–53.
32. Hwang, T. Parallel decoding of binary BCH codes. *Electron. Lett.* **1991**, *27*, 2223–2225. [CrossRef]
33. Davey, M.C.; Mackay, D.J.C. Reliable communication over channels with insertions, deletions, and substitutions. *IEEE Trans. Inf. Theory* **2001**, *42*, 687–698. [CrossRef]
34. Ezpeleta, J.; Krsticevic, F.J.; Bulacio, P.; Tapia, E. Designing robust watermark barcodes for multiplex long-read sequencing. *BMC Bioinform.* **2017**, *33*, 807–813. [CrossRef]
35. Costea, P.I.; Lundeberg, J.; Akan, P. Tag GD: Fast and accurate software for DNA tag generation and demultiplexing. *PLoS ONE* **2013**, *8*, e57521. [CrossRef]
36. Tambe, A.; Pachter, L. Barcode identification for single cell genomics. *BMC Bioinform.* **2019**, *20*, 32–41. [CrossRef]