2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI) | 978-1-6654-3946-6/21/$31.00 ©2021 IEEE | DOI: 10.1109/ISVLSI51109.2021.00042

PipeBSW: A Two-Stage Pipeline Structure for Banded Smith-Waterman Algorithm on FPGA

Luyi Li, Jun Lin, *IEEE Senior Member* and Zhongfeng Wang, *IEEE Fellow*
School of Electronic Science and Engineering
Nanjing University
Nanjing, China
Email: luyli@smail.nju.edu.cn, jlin@nju.edu.cn, zfwang@nju.edu.cn

***Abstract*—With the development of bioinformatics, there is unprecedented progress and wide applications of DNA sequencing technology, especially in helping biological and medical researchers obtain and analyze complete DNA sequences. Smith-Waterman (S-W) algorithm is one of the critical algorithms used in this technology. However, the tremendous amount of DNA sequence data makes it very inefficient to implement the algorithm only with software, so many recent works have tried to accelerate it on hardware. Unfortunately, many of them implemented on the CPU+FPGA heterogeneous platform are likely to require high memory bandwidth and large hardware resource. To mitigate these drawbacks, in this paper we propose PipeBSW, an hardware acceleration implementation on FPGA based on the concept of banded S-W. Using the lookahead calculation technique in cells and a two-stage pipeline structure for processing elements and the backtracking module, the experimental results show that the proposed system can achieve a 720.9Mbps throughput with a 58.4% LUT consumption reduction, compared with the prior work without the pipeline structure.**

***Keywords*-FPGA, Smith-Waterman Algorithm, Hardware Acceleration, DNA Alignment**

I. Introduction

Since the Human Genome Project was put forward, people have never stopped exploring DNA sequences in recent decades. DNA sequencing technology, which was invented to obtain and analyze the complete sequences, has also been developing rapidly.

In the process of DNA sequencing, sequence alignment is a very critical technique. How to quickly align sequences has become a focus of research by bioinformatics workers. Software tools such as Minimap2 [1], BWA-MEM [2], and Bowtie2 [3] are designed to simplify the operational complexity of the complete process. Many classic algorithms, such as Burrows-Wheeler Transform (BWT) [4], and Smith-Waterman local alignment (S-W) [5], [6], are widely applied in these tools. Take a state-of-the-art whole-genome sequencing (WGS) process as an example. Short reads sampled randomly from human DNA, with the length of 100-200bp (short for base pair), are mapped to a human gene reference (3,000Mbp). During this period, each short read firstly uses BWT to locate some candidate segments in the reference quickly and roughly, where the read and reference are likely to be matched. Then the alignment between the read and the reference segment could be further optimized by S-W, and in this period, high match accuracy is always pursued.

In recent years, with the introduction of the high-throughput sequencing (HTS) technologies [7], there is a tremendous increment in the amount of gene data. Unfortunately, only using software to accelerate these algorithms has been insufficient to meet the surge. Therefore, this situation has attracted wide attention from the field of high-performance computation. Researchers have been focusing on accelerating the process with the aid of high-performance infrastructures such as CPU clusters, GPUs, and FPGAs [8].

Because of its highly customizable and reconfigurable features, FPGA has always been an excellent platform to accelerate specific algorithms, both in academia and industry. As we know, S-W can be divided into two steps: scoring and backtracking. Most of the relative works use FPGA to decrease the time consumption caused by the scoring, which almost takes up to 90% of the total time. A scoring matrix and the position of the maximum score are recorded in the memory and then transferred back to the CPU to do the backtracking. However, as data expands massively, it is predictable that there will be a memory bottleneck [9], which is likely to cause unpredictable stalls.

In this paper, we propose a new complete S-W implementation to mitigate the problems above, called PipeBSW. We optimize the scoring step using the lookahead calculation technique [10]. For the transmission bandwidth issue, we choose to implement the algorithm entirely on FPGA, so a hardware backtracking module (BTU) is also designed for the backtracking step. Furthermore, we observe that in a WGS process, S-W usually only needs to calculate the scores along a diagonal band in the matrix called banded S-W [11], [12]. Therefore we design a specific processing element (PE) structure to construct this band. At last, a novel two-stage pipeline structure between PE and BTU can improve the module's reuse rate. The structure minimizes resource consumption, and meanwhile it maintains parallel computing. In summary, the main benefits of the proposed technique are as follows:

978-1-6654-3946-6/21/$31.00 ©2021 IEEE
DOI 10.1109/ISVLSI51109.2021.00042

- The lookahead calculation technique improves parallelism of cells, reducing the number of cycles it takes to complete the calculation.
- The pipeline structure only needs to adjust the calculation timing between PE and BTU, without the need to make significant changes to original hardware modules.
- The complete solution can be implemented on a pure FPGA platform without needing a CPU, which mitigates the potential transmission bandwidth issue.

II. Background

A. Smith-Waterman Algorithm

The Smith-Waterman algorithm [5] realizes alignment between two sequences. It consists of two phases: (1) filling a scoring matrix and (2) backtracking the matrix to find an optimal alignment between the read and reference sequence. There are three types of errors: mismatch, insertion, and deletion, and the latter two are collectively referred to as gap errors. The following recurrence formula can describe the scoring step, where $H_{i,j}$ represents the score of the i-th row and the j-th column in the matrix:

$$H_{i,j} = \max \begin{cases} H_{i-1,j-1} + S(i,\ j) & : \text{(Mis)Match} \\ H_{i-1,j} \ - W_1 & : \text{Insertion} \\ H_{i,j-1} \ - W_1 & : \text{Deletion} \\ 0 & : \text{Baseline} \end{cases} \quad (1)$$

The complete algorithm can be demonstrated by Fig. 1. Firstly, the first row and column are initialized with a certain value. In this case it is set as 0. Secondly, the scoring matrix is filled following the formula mentioned above. In this case, we set W_1 as 1, and $S(i,j)$ is 2 if $Ref[i] = Read[j]$ or otherwise is -2. Taking $H_{1,1}$ as an example, $S(1,1)$ is 2 because $Ref[1] = Read[1]$ = "A". As a result, $H_{1,1}$ is the maximum of $\{H_{0,0}+2=2, H_{1,0}-1=-1, H_{0,1}-1=-1, 0\}$, which is namely 2. It means that this score comes from its upper left position. Finally, the maximum score of the matrix is located, which is 5 in this case. And then the backtracking starts from the position of the maximum. If this score comes from the upper left, it is a match or mismatch. In the Figure, the maximum comes from the upper left score 3, and $S(4,4)=2$, so this is a match, and it is recorded in the alignment information: "G"-"G". If the score inherits from the left or top, the corresponding gap error is also recorded. Take the score 1 on the backtracking path as an example. This score comes from the left score 2, so it is a deletion and a "-" character is inserted into the read sequence: "T"-"-". Otherwise, it is an insertion, and the "-" is added to the reference. This process is repeated until the path comes across the baseline 0, and finally the optimal alignment result is output.

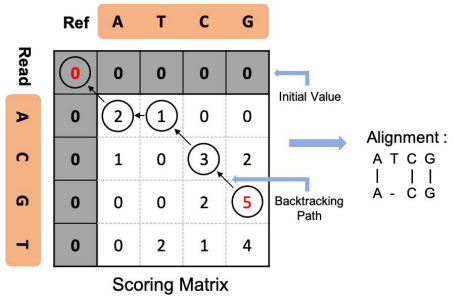

Figure 1: An example of the S-W algorithm. By backtracking the scoring matrix, the read sequence is aligned with "A - CG" to the reference sequence "ATCG".

B. Banded S-W

We assume that our work is employed in WGS. As mentioned in Section I, in the WGS workflow, S-W is usually used to select the best match among the candidate reads. In such a situation, the number of errors (mismatch, insertion, and deletion) in reads is limited because BWT has already filtered most of the bad candidates. As observed, the optimal alignment path is usually not far away from the main diagonal of the matrix. For instance, if the system only accepts a number of errors within 10% of the sequence length, the path can be only 10% upper or lower than the main diagonal [13]. Instead of computing the complete scoring matrix, only positions within the diagonal band need to be calculated, which is called **banded S-W** [11], [12]. A hardware structure is designed to implement this banded S-W, which will be detailed in Section III.

C. Direction Matrix

To compress data and save resources, the direction matrix is always used to replace the scoring matrix [9]. In this application, we assume that our structure is utilized to do the global alignment, so the situation that the stop in the backtracking caused by meeting the baseline is categorized into a mismatch error. Therefore, all the four situations: match (the score inherits from the upper left $\nwarrow$), mismatch (from the upper left $\nwarrow$), insertion (from the top $\uparrow$), and deletion (from the left $\leftarrow$), can be represented within 2 bits. We call it **direction record rule** in this paper. As the length of the sequence expands, the direction matrix's advantage over the scoring matrix is strengthened. Compared with larger and larger bit-widths to represent continuously increasing scores, the direction matrix still keeps 2 bits per element, which saves resources to a large extent.

III. Hardware Architecture

A. Overall Workflow

There have been several methods of parallel computing for S-W concluded in [14], two of which utilized in our work are demonstrated in Fig. 2. Fig. 2(a) illustrates a

systolic array structure which consists of several small calculation cells. Positions in the main diagonal direction can do parallel calculation. It is adopted in a 3×3 calculation cell structure(detailed in Section III-B). Fig. 2(b) illustrates a parallel structure for global alignment in the banded S-W. This structure is used among different PEs to construct the main diagonal band.

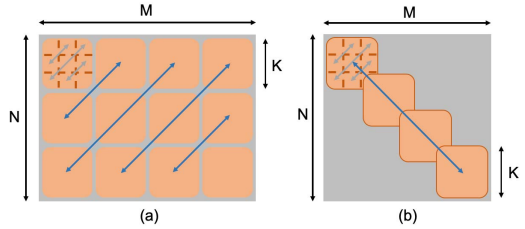

Figure 2: In a $M\times N$ sequence alignment, square cells sized by $K\times K$ compose a systolic array structure. Arrows in gray and blue show that the scoring in the diagonal direction can run parallel.

Our complete workflow is depicted in Fig. 3. Read and reference are input through FIFO, sliced into segments, and then sent to relative PEs. The scoring step is executed by parallel PEs, which consists of small calculation cells. Once the scoring finishes, the direction matrix of each PE is generated.

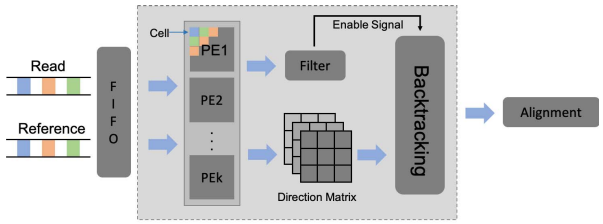

Figure 3: The complete banded S-W algorithm workflow on FPGA. Major modules are plotted in deep gray. Blue arrows show the direction of data flow.

The system decides whether to do the backtracking based on the filter module. Some reads turn out to be bad candidates after the backtracking, and in order to identify them early in the workflow, we employ an **error counting mechanism** running together with the scoring. By this mechanism, the number of different types of errors can be obtained once after the scoring. If the sum of errors exceeds the threshold, the filter will disable the backtracking module, and the backtracking step will be skipped, which saves computational time.

B. Calculation Cell

Utilizing the method in Fig. 2(a), one parallel calculation round is finished within one clock cycle. Therefore in a 3×3 matrix, because the calculation of $H_{1,2}$ relies on $H_{1,1}$, they must be in a different clock cycles. Usually, it takes five cycles in total to finish filling the 3×3 matrix.

Inspired by the carry lookahead adder, a **lookahead calculation technique** can be adopted to fill a 3×3 scoring matrix within three clock cycles [10] as shown in Fig. 4. Taking $H_{1,1}$ and $H_{1,2}$ as an example again, we can obtain their results at the same cycle because all six possible paths leading to $H_{1,2}$ are directly compared with each other, without knowing $H_{1,1}$ as shown in Fig. 5. At first, $H_{3,3}$ is categorized into the calculation in the first cycle together with $H_{1,2}$. However, it is observable that the performance of this technique is quite sensitive to the critical path of the comparison unit, especially that the lookahead calculation of $H_{3,3}$ contains too many times comparison operations. Therefore, we further optimize the calculation cell in the following two aspects:

- **More balanced critical path length per cycle.** As illustrated in Fig. 4, 3 positions are calculated in each clock cycle after changing the cycle of the calculation of $H_{1,3}, H_{3,1}, H_{2,3}, H_{3,2}$. Within the same 3 cycles, only 3 positions $(H_{1,2}, H_{2,1}, H_{3,3})$ need lookahead technique after modification, a half less than before $(H_{1,2}, H_{2,1}, H_{1,3}, H_{3,1}, H_{2,3}, H_{3,2})$. In the initial version, $H_{1,3}$ and $H_{3,1}$ have to compare all the eight possible results. While now they only have 4 candidate results to be compared because $H_{1,2}$ and $H_{2,1}$ have been calculated in the last clock cycle.
- **More parallel and reusable comparison unit.** Take $H_{1,2}$ as an example in Fig. 5. Instead of comparing all possible results one by one, they are divided into pairs. The previous 6-level serial comparison is now reduced to a 3-level parallel comparison. What is more, part of the candidate results of $H_{1,2}$ can be shared with the calculation of $H_{1,1}$, so there is no redundant calculation and comparison.

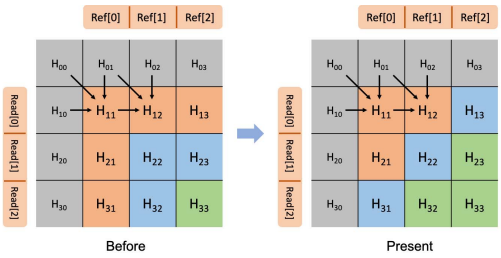

Figure 4: A 3×3 calculation cell using lookahead calculation before and after optimization. The gray area represents the input from other cells. Positions in the same color are processed in the same cycle.

As mentioned before, in order to filter reads with too many errors, we design a simple **error counting mechanism** for counting different types of errors. Mistakes of each position

inherit from its source of the score. Take $H_{1,1}$ as an example:

- If the score of $H_{1,1}$ equals $H_{0,0}+S(0,0)$ where $S(0,0)$ equals -2, mismatch errors of $H_{1,1}$ will be added by one on that of the input $H_{0,0}$. The other two types of errors are the same with those of $H_{0,0}$.
- If the score of $H_{1,1}$ equals $H_{0,1}-W_1$ ($H_{1,0}-W_1$), the relative insertion (deletion) errors will be increased by one from that of $H_{0,1}$ ($H_{1,0}$). The other two types of error also inherit from $H_{0,1}$ ($H_{1,0}$).

Based on this mechanism, the number of errors is accumulated along the path to the end of the cell, and the results will be passed to other cells for the following rounds of calculation.

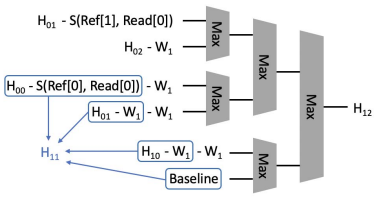

Figure 5: The parallel comparison unit in the cell. The framed part indicates the shared resources.

The 2-bit direction information of each position in the 3×3 matrix also needs to be stored for generating a direction matrix in the PE (detailed in Section III-C).

C. Main Diagonal Band

The systolic array is often used in the hardware implementation of matrix multiplication. The method is simple in structure, easy to control, and suitable for parallel computing [15]. In this S-W application, different cells connect with each other to construct the structure. Thirteen 3×3 cells are sufficient to constitute a 36×36 processing element (PE).

As depicted in Fig. 6, cells can be reused in different cycles. Different cells in the anti-diagonal direction do the scoring synchronously. After the scoring, 25 positions at the end of the PE compose a "L" region. The index of the maximum among the 25 scores is recorded, which implies the beginning position of the backtracking.

To provide a diagonal band, there is a 12×12 overlap between each two PEs. Furthermore, the band inside the PE is constructed by expanding the "L" region from the end to the beginning of the PE. All cells along this band need to record the direction information to generate a direction matrix. In this matrix, each "L" region takes up one row. According to our **direction record rule** in Section II-C, each row needs 50 bits, which means one matrix takes $50\times24=1200$ bits, 48% less than 2304 bits if all the positions are stored. A buffer with 24 entries is employed to store the matrix, with each entry storing the information of one 50-bit "L" region, as shown in Fig. 7.

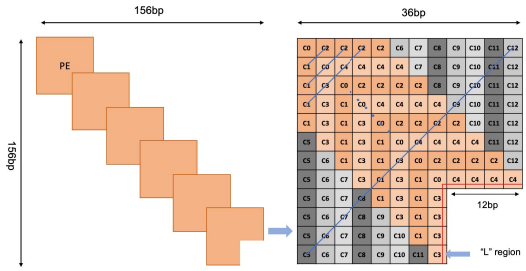

Figure 6: A 6-PE parallel structure and a 13-cell systolic array structure. Positions in origin are in the diagonal band. The blue line represents one round of cell calculation.

Only cells along the diagonal band record the direction information because the filter module employing the **error counting mechanism** ensures that the backtracking path is bounded in the band. If the path goes beyond the region, it means that in the current PE, the number of errors must satisfy the inequality: $|Insertion-Deletion|\geq12bps$. It indicates that the number of gap errors is no less than 12bps, which exceeds the threshold of the filter that is set as 10 in this system. Under this situation, the read is judged as a low-quality candidate whose match accuracy does not meet the requirement of S-W in WGS.

D. Backtracking Module

The backtracking step follows the scoring step. As the volume of data surges largely, the traditional method of sending the matrix to a general-purpose processor is likely to encounter bandwidth issues. To mitigate this, we design a backtracking module completely implemented on hardware, which is called BTU, as shown in Fig. 7.

The BTU starts reading the direction matrix buffer as soon as the scoring finishes. From the last entry to the first, in each clock cycle, the BTU reads 2-bit direction information at a specific location and then appends these 2 bits to the alignment path, which is later combined with read and reference sequences to generate the final alignment information.

In order to locate the 2-bit direction information, a pointer of entry is used to select the row of the matrix, and a pointer of position is to determine the specific column in that row. At the beginning of the backtracking, the entry pointer is initialized to 23, and the position pointer is initialized to the index number of the maximum in the last "L" region. These two pointers are updated in each cycle based on the direction information that is read. At last, the backtracking ends when the entry pointer is smaller than 0. Therefore, the BTU spends 24-36 clock cycles to finish generating the alignment path of one PE.

E. Two-Stage Pipeline Structure

A 6-PE structure is considered in this section. Our initial design is that 6 PEs begin the scoring step synchronously,

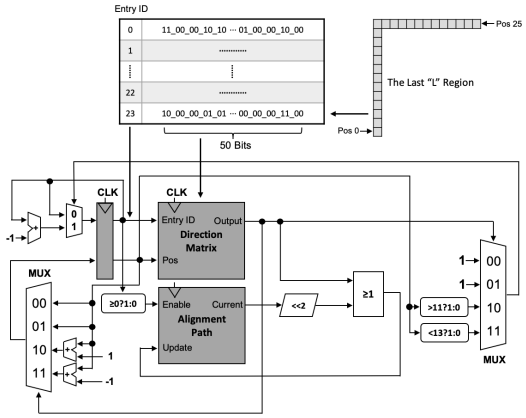

Figure 7: An "L" region, a direction matrix buffer, and the general structure of the BTU.

followed by 6 BTUs running parallel to acquire the alignment information of different sequence segments. Unfortunately, it is difficult to handle the problem of integrating the results of different segments, so we turn to a strategy of parallel scoring with serial backtracking: starting from the last entry of the last PE's direction matrix buffer, one BTU serially backtracks a path along the diagonal band to the first entry of the first PE's buffer. However, this strategy causes a waste of resources needed to instantiate the PE module because after finishing the scoring, PEs in the front of the structure are in the idle state when waiting for BTU to read their matrix buffers.

In order to increase the reuse rate of the PE module while keeping the total running time unchanged, we propose a **two-stage pipeline structure** for the PE and BTU, where the two stages represent the two steps of the algorithm: scoring and backtracking. It only changes the timing between the modules without modifying the logic implementation inside them.

First, take a 2-PE structure as an example to elaborate on this structure. Including time cost by the input stage of each cell, it takes 80 clock cycles for one PE to finish the scoring, and one BTU spends m cycles ($24 \leq m \leq 36$). In the initial serial backtracking strategy, the 2-PE structure spends $80 + 2m$ cycles running the algorithm. In this new structure, the scoring of PE1 ends simultaneously with the backtracking of PE0, which is implemented by delaying the start time of PE1 by m cycles, as depicted in Fig. 8. It takes $80 + 2m$ clock cycles, same with the initial version.

For the 2-PE structure, the advantage of this structure is not significant. However, as the length of the sequence increases, the reuse rate of PE modules can be significantly improved, and the resources saved are approaching 50% (if the sequence is long enough). In a 6-PE structure, as Fig. 8 shows, 3 PEs using a 2-stage pipeline structure are sufficient to complete the algorithm. If m is no less than

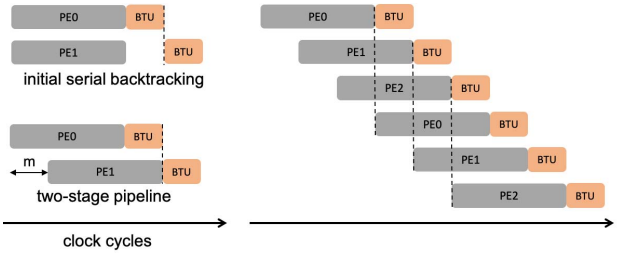

Figure 8: The two-stage pipeline structure for S-W.

27, $3m$ cycles will be longer than 80 cycles cost by one PE to finish the scoring. In this case it is noticeable that PE0 is in the idle state when the fourth segment of the sequence begins the scoring. Therefore, PE0 can be reused to continue calculating this segment. PE1 and PE2 follow the same technique, and the complete system saves up to a half less PE's instantiation than before, in the case of the same total running time.

When it comes to determining the value of m, it depends upon the number of errors in the sequence. The more gap errors, the longer time the BTU costs. Since filtered low-quality candidate reads have been filtered at the end of the scoring step, in most cases, errors are distributed in each segment. In this system, m is set as 27, the minimum requirement for implementing the pipeline structure on a 6-PE structure. What needs to be ensured is that when the BTU begins backtracking, the direction matrix of the relative PE has been generated, so there are likely to be a few cycles wasted when the BTU and PE wait for each other. However, it is acceptable compared with the resources saved in the system.

IV. Experimental Validation

A. Experiment Environment

Chisel HDL from UC Berkeley is employed to construct the RTL code. We complete simulation and synthesis on Vivado platform, using Xilinx Virtex-7 series FPGA.

B. Resource Consumption

In terms of resource consumption, in the cell, part of the comparison results are shared among different positions as mentioned in Section III-B, which reduces LUTs per cell by 31.7%, from 918 to 627. Moreover, the two-stage pipeline structure reduces the number of instantiated PE by a half. Consequently, the system achieves a 58.4% LUT consumption decrease, which means more sequence alignments can be implemented to run parallel on a single FPGA.

C. Circuit Performance

Reducing the length of the critical path is of great importance to increase the circuit performance. In our work, the main bottleneck exists in the lookahead calculation. We

Table I: Usage of FPGA Resources

Resources	Before	Present	Decrease
LUTs per cell	918	627	31.7%
LUTs per PE	15875	11956	24.7%
LUTs of System	98585	40985	58.4%

have balanced the critical path and improved the comparison unit as mentioned in Section III-B. To further enhance it, we delay part of the comparison of $H_{1,2}$ and $H_{2,1}$, and forward part of the $H_{3,3}$ to the second clock.

Table II: Property Overview

Property	Value
Platform	Virtex-7
Frequency	166.7Mhz
Throughput	720.9Mbps

Several typical S-W accelerators are listed in TABLE III. The result has shown that our implementation can achieve both high performance and low resource consumption. Moreover, PipeBSW runs the algorithm entirely based on FPGA, which is easier to implement.

Table III: Comparison Between Some Relative Works

Paper	Backtrack	Freq (Mhz)	LUTs	Arch
Nawaz [9]	Yes	79.3	-	FPGA
Fei [15]	Yes	150.0	57870	FPGA
Liao [16]	Yes	125.0	70839	FPGA
Our	Yes	166.7	40985	FPGA

V. Previous work

The first hardware banded S-W implementation was proposed in [12], which accelerates the S-W for NCBI BLASTP software. [16], [17] are two more classical accelerators based on this idea. In [9] Nawaz et al pointed out the memory bandwidth problem in the FPGA+CPU architecture, so he chose to record the direction information and implement the algorithm on FPGA. The most related work is [10]. Zhang et al proposed SW core using the lookahead calculation. However, the design of the SW core did not pay attention to the critical path, and the whole system is only able to do an evaluation. Observing the potential of this approach, we continue to improve it to a new level. Compared with it, our work has fully implemented the Smith-Waterman algorithm without much performance loss and resource increase.

VI. Conclusion

This paper proposes PipeBSW, a re-configurable system implementing the whole Smith-Waterman algorithm on FPGA. We employ a lookahead calculation technique and a two-stage pipeline structure on the hardware implementation, achieving a high degree of parallelism with low resource consumption. The experiment results show that PipeBSW's throughput reaches 720.9Mbps and that there is a 58.4% reduction in total LUT consumption. However, the problem with PipeBSW is that it is only able to backtrack one alignment path. We will be committed to handling this defect in the future.

References

[1] H. Li, "Minimap2: pairwise alignment for nucleotide sequences," *Bioinformatics*, vol. 34, no. 18, pp. 3094–3100, 2018.

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

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

[4] H. Li and R. Durbin, "Fast and accurate short read alignment with burrows–wheeler transform," *bioinformatics*, vol. 25, no. 14, pp. 1754–1760, 2009.

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

[6] M. Alser, H. Hassan, H. Xin, O. Ergin, O. Mutlu, and C. Alkan, "Gatekeeper: a new hardware architecture for accelerating pre-alignment in dna short read mapping," *Bioinformatics*, vol. 33, no. 21, pp. 3355–3363, 2017.

[7] J. A. Reuter, D. V. Spacek, and M. P. Snyder, "High-throughput sequencing technologies," *Molecular cell*, vol. 58, no. 4, pp. 586–597, 2015.

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

[9] Z. Nawaz, M. Nadeem, H. van Someren, and K. Bertels, "A parallel fpga design of the smith-waterman traceback," in *2010 International Conference on Field-Programmable Technology*, pp. 454–459, IEEE, 2010.

[10] Y. Zhang, J. Wu, M. Li, J. Lin, and Z. Wang, "A three-level scoring system for fast similarity evaluation based on smith-waterman algorithm," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, IEEE, 2020.

[11] K.-M. Chao, W. R. Pearson, and W. Miller, "Aligning two sequences within a specified diagonal band," *Bioinformatics*, vol. 8, no. 5, pp. 481–487, 1992.

[12] B. Harris, A. C. Jacob, J. M. Lancaster, J. Buhler, and R. D. Chamberlain, "A banded smith-waterman fpga accelerator for mercury blastp," in *2007 International Conference on Field Programmable Logic and Applications*, pp. 765–769, IEEE, 2007.

[13] S. Salamat and T. Rosing, "Fpga acceleration of sequence alignment: A survey," *arXiv preprint arXiv:2002.02394*, 2020.

[14] X. Chang, F. A. Escobar, C. Valderrama, and V. Robert, "Optimization strategies for smith-waterman algorithm on fpga platform," in *2014 International Conference on Computational Science and Computational Intelligence*, vol. 1, pp. 9–14, IEEE, 2014.

[15] X. Fei, Z. Dan, L. Lina, M. Xin, and Z. Chunlei, "Fpgasw: Accelerating large-scale smith–waterman sequence alignment application with backtracking on fpga linear systolic array," *Interdisciplinary Sciences: Computational Life Sciences*, vol. 10, no. 1, pp. 176–188, 2018.

[16] Y.-L. Liao, Y.-C. Li, N.-C. Chen, and Y.-C. Lu, "Adaptively banded smith-waterman algorithm for long reads and its hardware accelerator," in *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 1–9, IEEE, 2018.

[17] P. Chen, C. Wang, X. Li, and X. Zhou, "Hardware acceleration for the banded smith-waterman algorithm with the cycled systolic array," in *2013 International Conference on Field-Programmable Technology (FPT)*, pp. 480–481, IEEE, 2013.