Bi-WaveX: a SIMD AVX-512 implementation of the Bi-directional Wavefront Alignment Algorithm

Dane Marcus Chan[1,2,*], Christian Mir Castillo[1,3], Raphael Jeremiah Tan Ai[1,4], and Roger Luis Uy[1,5]

[1] De La Salle University, 2401 Taft Avenue, Manila, Philippines
[2] christian_mir_g_castillo@dlsu.edu.ph
[3] dane_chan@dlsu.edu.ph*
[4] raphael_tanai@dlsu.edu.ph
[5] roger.uy@dlsu.edu.ph

Abstract. Sequence alignment is the process of aligning two genomic sequences. It is the first step in various bioinformatics processes and is essential in many fields of genomics and bionformatics, such as *de novo* genome assembly and variant detection. One such sequence alignment algorithm is the Bi-Directional Wavefront Alignment (BiWFA) algorithm, which guarantees optimal alignment with $O(ns)$ time and $O(s)$ space complexity. In this paper, a novel approach is presented wherein AVX-512 Single Instruction Multiple Data (SIMD) instructions are integrated into the BiWFA algorithm. Its execution time is then measured and compared against the original BiWFA library, wherein it exhibited speedups of up to 4.67x from the original.

Keywords: read-mapping, alignment, wavefront, bi-directional wavefront alignment algorithm, single instruction multiple data.

1 Introduction

Pairwise sequence alignment is used widely in genome analysis to identify variants or mutations and more. Traditional alignment algorithms, like Needleman-Wunsch (NW) algorithm [1], the Smith-Waterman (SW) algorithm [2], and the further improved Smith-Waterman-Gotoh (SWG) algorithm [3] were developed, but scale poorly in time and memory as sequence lengths increase due to their output-insensitive nature. Heuristic algorithms like FASTA [4] and BLAST [5] were also developed, substantially reducing computation times at the cost of not guaranteeing optimal alignment.

The Wavefront Alignment algorithm (WFA) [6] builds off the SWG algorithm to guarantee optimal alignment in $O(ns)$ time and $O(s^2)$ memory, where n is the read length and s is the alignment score. This is a marked improvement over the previous NW and SWG algorithms that guarantee optimal alignment at $O(nm)$ time, with n and m representing both sequence lengths. Afterwards,

J. Caro et al. (eds.), *Proceedings of the Workshop on Computation: Theory and Practice (WCTP 2025)*, Atlantis Highlights in Computer Sciences 24,
https://doi.org/10.2991/978-94-6239-638-8_23

the Bi-directional Wavefront Alignment algorithm (BiWFA) was developed, reducing the memory complexity of the algorithm to $O(s)$ by performing WFA concurrently on both ends of the matrix [7]. These algorithms had been adapted to support parallel processes, such as using Field Programmable Gate Arrays (FPGA) [8, 9], Graphics Processing Units (GPU) [10] and Processing-In-Memory (PIM) [11], showing speedups from 1.5x to 17x. However, no implementation using single-instruction multiple data (SIMD) architecture, where single operations can be performed on multiple data of the same type, for this algorithm has yet to be proposed.

This paper proposes a handwritten SIMD implementation of the BiWFA algorithm using AVX-512, which is an extension of the Advanced Vector Extensions (AVX) family. AVX-512 provides 512-bit ZMM registers and introduces a new kind of register called opmask registers, which enable filtering of elements during operations. AVX-512 was used to optimize parts of the code that are vectorizable to reduce the execution time while maintaining the optimal alignment that the algorithm outputs.

The rest of the paper is discussed as follows. Section 2 discusses the main functions of BiWFA that were translated to AVX-512 which make up the core loop of the algorithm and then details the dataset and methodology used for testing. Section 3 discusses the results of the testing process, particularly about correctness and execution time. Finally, Section 4 discusses suggestions for future works and research.

2 DNA TO DNA ALIGNMENT WITH BIWFA

The inputs of the WFA and BiWFA algorithms are the strings pattern (q) and text (t), with lengths defined as n, and m, respectively. The pattern will be compared against the text to determine the minimum set of edits needed to maximize the amount of matches between the two sequences. Afterwards, it returns the Compact Idiosyncratic Gapped Alignment Report (CIGAR), a string that stores all edits made to achieve the optimal match [12], and the alignment score, which is a quantification of the edits made.

The core functions of WFA [7], namely WF_Extend and WF_Next, is used by the BiWFA algorithm to diagonally traverse the DP matrix. However, the key difference between WFA and BiWFA is that BiWFA approaches the matrix from both ends, leading to a lower space complexity of $O(s)$ compared to WFA's $O(s^2)$.

2.1 Wavefront Alignment Algorithm

The types of alignment that are supported by the WFA algorithm are global, or end-to-end, alignment, and semi-global, or ends-free, alignment. Since the type of alignment this implementation is concerned with is global alignment, wherein the sequences are aligned to their entirety, n and m will be assumed as equal for the purposes of this paper.

The goal of the algorithm is to traverse the DP matrix of dimensions (n, m) from the starting index $(0, 0)$ while minimizing the score by incurring the least amount of penalties through mismatches, substitutions, and gaps.

The WFA algorithm is characterized by its nature as a DP-based solution, solving recurrent equations of the SWG algorithm in the form of wavefronts. It uses three DP matrices, referred to as the match-mismatch (M), insert (I), and delete (D) matrices that make up the wavefront. Additionally, it uses a gap-affine penalty scheme, which takes account of existing gaps to better emulate real-world insertions and deletions (indels) scheme to score the alignment between the text and the pattern. Particularly, the match (a), mismatch (x), open (o), and extend (e) operations are denoted by the penalty values of *0, 4, 6, 2*, respectively.

The recurrence relation of the wavefront components are observable in Table 1.

Table 1. WFA Recurrency Relations

$I_{v,h} = min\{M_{v,h-1} + o + e, I_{v,h-1} + e\}$
$D_{v,h} = min\{M_{v-1,h} + o + e, D_{v-1,h} + e\}$
$M_{v,h} = min\{I_{v,h}, D_{v,h}, M_{v-1,h-1} + s(q_{v-1}, t_{h-1})\}$

The diagonals are centered around the main diagonal at $k = 0$, with the offsets of each diagonal being the distance to the point, $p(v, h)$, with respect to the origin. The matrices are explored based on a coordinate system wherein the current position p is defined by $p = (v, h)$, where $0 \leq v < n$ and $0 \leq h < m$. For each score s and diagonal k, the furthest-reaching (f.r.) point refers to the DP-cell on the diagonal that has the largest offset within the diagonal. Each diagonal in each matrix has its own f.r. point $\mathscr{F}_{s,k}$. The components of wavefront s (WF_s) are $\widetilde{I}_s$, $\widetilde{D}_s$, and $\widetilde{M}_s$ which are each represented by a vector of offsets. Finally, let $\widetilde{M}^{hi}$ be the index of the rightmost diagonal, and $\widetilde{M}^{lo}$ the index of the bottommost diagonal [6].

The matrices are explored using two functions: WF_Extend, which extends diagonals until a mismatch is encountered, and WF_Next, which evaluates possible paths to minimize the score while maximizing the offset. The symbols defined and used by WFA are shown in Table 2.

The algorithm is initialized at $p(0, 0)$ and score $s = 0$, then WF_Extend and WF_Next are repeatedly called in an iterative loop until either (1) the algorithm has found a traceable path to (n, m) or (2) if the offset of the score at a certain diagonal has exceeded the maximum offset, which is the text length (m).

WF_Extend takes the pattern (q), the text (t), and the M component of the s-wavefront (WF_s) as inputs to iterate through all available diagonals, from *lo* to *hi*, and extend each sequentially until it encounters a mismatch, in which case it will return the offset of the new f.r. point in that diagonal.

It should be noted, however, that certain parts of the library have already been implemented in C-based AVX-512 intrinsics. These parts are limited in

Table 2. Symbol Notations used in WFA [6]

Symbol	Description
q, t	Input strings
$p = \{a, x, o, e\}$	Gap-affine penalty scores
M, I, D	SWG Alignment Matrices
$\mathscr{F}_{s,k}$	Furthest-reaching (f.r.) point on diagonal k with score s
$\widetilde{M}_{s,k}$	Offset in the diagonal to the corresponding f.r. point $\mathscr{F}_{s,k}$
$\bar{I}_s$, $\bar{D}_s$, $\bar{M}_s$	Components of the wavefront WF_s
WF_s	Wavefront of score s
$\widetilde{M}^{hi}$, $\widetilde{M}^{lo}$	Index of the rightmost and leftmost diagonal respectively

scope and is mostly restricted to WF_Extend, particularly the exploration of the initial four characters of each diagonal. However, the rest of WF_Extend is performed sequentially, allowing for more room for vectorization.

Then, for every iteration of the main loop, after WF_Extend finishes executing but before WF_Next begins, the score of the algorithm gets incremented.

WF_Next takes in the $\{M, I, D\}$ wavefronts, the pattern (q), the text (t), and the current score (s) as inputs, while $\widetilde{M}^{hi}$ and $\widetilde{M}^{lo}$ are calculated inclusively. The next step goes through the wavefronts of the diagonals and gets the maximum score between the two possible outcomes for each wavefront, then the three possible outcomes for the max of the M, seen in Equations 1, 2, and 3, while additionally checking for the exit condition, which is if it exceeds the lengths of either the pattern or text.

$$\widetilde{I}_{s,k} = \left\{ \begin{matrix} \widetilde{M}_{s-o-e,k-1} & \text{(Open Insertion)} \\ \widetilde{I}_{s-e,k-1} & \text{(Extend Insertion)} \end{matrix} \right\} + 1 \tag{1}$$

$$\widetilde{D}_{s,k} = \left\{ \begin{matrix} \widetilde{M}_{s-o-e,k+1} & \text{(Open Deletion)} \\ \widetilde{D}_{s-e,k+1} & \text{(Extend Deletion)} \end{matrix} \right\} \tag{2}$$

$$\widetilde{M}_{s,k} = \left\{ \begin{matrix} \widetilde{M}_{s-x,k} + 1 & \text{(Substitution)} \\ \widetilde{I}_{s,k} & \text{(Insertion)} \\ \widetilde{D}_{s,k} & \text{(Deletion)} \end{matrix} \right\} \tag{3}$$

The backtrace is done by taking the offsets of the wavefront and tracing back the path by determining the f.r. point of the previous wavefronts that the current offset originates from, with the number of matching characters determined by the difference between the offsets.

The string that is generated from the backtrace uses the symbols seen in Table 3, wherein each symbol in the CIGAR is accompanied by a number to denote how many consecutive times its respective operation was performed. For example, in

a sample CIGAR $5M3X6D9M$, $5M$ refers to five continuous matches, $3X$ refers to three consecutive mismatches, and so on and so forth. The insert and delete operations assume that a gap is opened so the number it is accompanied by also refers to the length of the gap, therefore, the score is calculated with the formula $6 + 2n$ wherein 6 is the gap opening penalty and $2n$ is the cumulative gap extension penalty.

Table 3. Table of CIGAR Operations

Operation	Symbol	Score
Match	M	0
Mismatch	X	4
Insert	I	6 + 2n
Delete	D	6 + 2n

2.2 Bidirectional Wavefronts and Breakpoint Detection

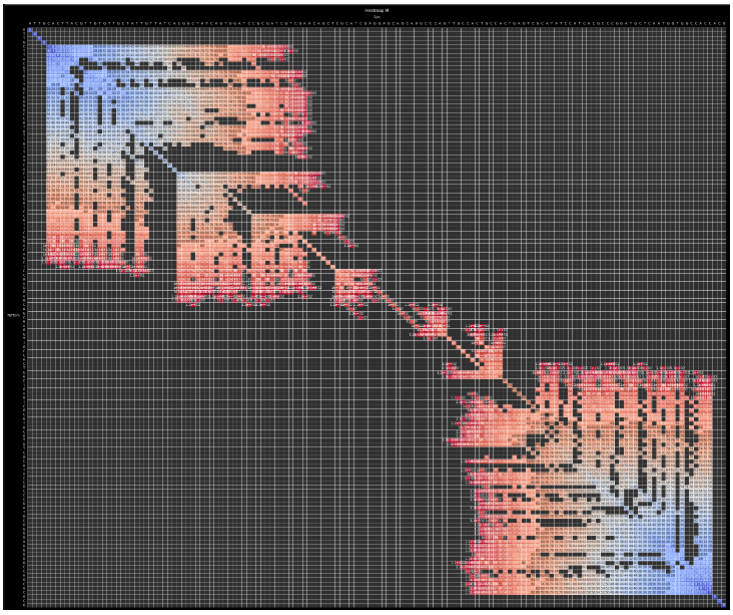

Fig. 1. BiWFA Heatmap of a Sample Pattern tested against a Sample Text

BiWFA extends WFA by aligning sequences from both ends simultaneously. One wavefront starts at $(0, 0)$ and progresses forward, while another begins at

(n, m) and moves in reverse. These wavefronts are computed alternately using WF_Extend and WF_Next until they overlap, at which point a breakpoint is identified. The optimal alignment score can be generalized as the sum of the score of two wavefronts (*eq. 4*). The WFA computes partial alignments of increasing scores until the alignment reaches the coordinates (n, m). This means that a partial alignment at score s is, at most, $p = max\{x, o + e\}$ steps away from the partial alignment of the previous score.

$$s_f + s_r = s_{opt} \tag{4}$$

$$s_f + s_r - o - p + 1 \tag{5}$$

When two wavefronts meet at a diagonal insertion or deletion, the gap opening penalty will be counted twice in the score because each wavefront is computed independently. Thus, the gap opening must be subtracted once from $s_f + s_r$ to obtain the optimal alignment score. To check if an overlap between the two wavefronts is valid, a score-balanced breakpoint is needed. Since each alignment will store p wavefronts, the forward wavefront will be checked against $p - 1$ reverse wavefronts, to ensure that both wavefronts are within valid distance for overlap detection. The best breakpoint score, then, becomes the sum with the gap open and $p-1$ subtracted (*eq. 5*). Both wavefronts are explored and checked for overlaps, the overlap score compared to *equation 5*, until the breakpoint is found.

2.3 SIMD Implementation

The resulting implementation is modular in nature, in that certain core functions of the algorithm have been implemented in AVX-512, but still relies on existing code in the library to properly run the implemented modules. The functions implemented in AVX-512 were the WF_Extend, WF_Next, BIWFA_Breakpoint functions which make up the core of the algorithm. Along with the backtrace function for generating the CIGAR. Additionally, the algorithm itself is, by nature, still a sequential algorithm, as the parallelism being implemented here is in the form of data-level parallelism, not multi-threaded or multicore parallelism.

In WF_Extend, each diagonal is extended independently from each other, allowing for parallelization. ZMM registers contain 512 bits, equivalent to 64 characters at a time. However, since each diagonal is stored by its 32-bit integer offset, only 16 diagonals at a time can be accommodated, with each diagonal having 4 characters to be processed at a time. Then, each character is compared using a vector XOR operation, where the output vector will have a 0x00 element for a match. Thus, by counting the trailing zeroes and dividing the count by 8, the number of consecutive matches per diagonal can be obtained. If a diagonal has encountered a mismatch, it will no longer be processed by further comparison operations. This is repeated until all 16 diagonals encounter mismatches, in which case the amount of consecutive matches are added to its respective offset and the loop can move on to the next batch of diagonals.

After the score has been incremented, the program will call WF_Next where it compares the scores at the f.r. point of the different matrices $\{M, I, D\}$. As the diagonals and matrices are mutually exclusive from one another, the process of iterating over multiple diagonals may be vectorized. Since the offset is a signed 32-bit integer, the ZMM registers allows the comparison of at most 16 diagonals at a time, while following the order of operations of the original function.

BiWFA checks for overlaps and breakpoints from diagonal to diagonal which allows for vectorization of the process, with AVX-512 it efficiently compares up to 16 diagonals per iteration, ensuring that breakpoint detection becomes computationally efficient while still being accurate due to the preserved linearity of the iteration through all the diagonals.

Finally, a part of the backtrace operation has been converted to AVX-512, specifically when tracing matches as there can be long chains of matches which do not affect the alignment score, previously only 8 characters at a time could be processed, but now 64 characters can be processed at a time through the use of ZMM registers allowing the program to compare the characters with another register filled with 'M' or 0x4D, the character for matches, and in the scenario that there are lower than 64 characters to match, the function will simply fall back to its original implementation.

2.4 Testing Methodology

For the datasets used in the evaluation phase, two different datasets are obtained from an existing database from the first human chromosome sequence GRCH38.p13 assembly which seervd as the input reference sequnce, creating 10K bp and 100K bp tset casse. The 100K bp test case was broken down into smaller datasets, containing sequences of lengths 30K, 50K, 80K base pairs. Eventually leading to 5 different sets of test cases, those being 10K, 30K, 50K, 80K, and 100K bp.

Datasets are separated into two categories, the first containing test cases that exclusively contain substitution of characters (mismatches), and the second containing test cases that contain both mismatches and insertion or deletion of characters (indels). The part of the sequence that contains the mismatches and indels are expressed as the error rates these changes were limited to a percentage of the string, to be specific the breakpoints are as follows: 1%, 5%, 10%, 20%, 30%, and 40%; an example of how the error rate functions is that if a 100K bp test case has an error rate of 10% that means 10K characters of the string have been modified. Following these breakpoints, the ratio of substitutions to indels followed was 4:1 meaning that for every four substitutions there was one indel as it is proposed to be a common ratio for the human genome [13], additionally the lengths of the indel gaps follows a geometric distribution to represent real data better where large indel gaps are lower in occurrence [14].

The method of testing compared queries and texts. The queries and texts will be aligned using three different libraries, namely Bi-WaveX, BiWFA-Intrinsics, and BiWFA-Sequential. The difference between each library is as follows: Bi-WaveX is the proposed implementation that uses handwritten assembly func-

tions called by the larger C program, BiWFA-Intrinsics is an implementation of the BiWFA library that uses C-based AVX512 intrinsics, and BiWFA-Sequential is a sequential implementation of the BiWFA library where no form of SIMD is used.

The performance of the Bi-WaveX implementation is evaluated against the original sequential BiWFA library and an intrinsics-based version (2/2/2025) that uses C-based AVX-512 intrinsics for some of its functions. Edlib is also included amongst the libraries evaluated against, but it cannot serve as a direct comparison as it solves a different problem, which is attaining the edit distance/edit-alignment between two sequences; that said, it still acts as a good reference for the upper-bound in terms of performance [7].

The hardware specifications of the computer that ran the tests are as follows: 11th Gen Intel(R) Core(TM) i5-11320H @ 3.20GHz, 3187 Mhz, 4 Core(s), 8 Logical Processor(s) with 16 GB of DDR4 RAM.

3 Results and Analysis

As outliers were encountered in the initial data analysis, the z-score method is used to determine which data points are outliers and to be filtered out.

The most important metric for evaluating Bi-WaveX is its correctness. In this regard, it will be evaluated against the original sequential BiWFA implementation for all available test cases. Next is the execution time of the algorithm, with the average execution time being calculated individually per library and dataset.

For the data concerning the BiWFA implementations, the average increase or decrease in execution time of the algorithms are compared to one another. The implementations themselves are Bi-WaveX, the intrinsics-based implementation discussed in Section 4.3 (BiWFA-Intrinsics), and the original sequential implementation of BiWFA, (BiWFA-AVXless).

3.1 Correctness

To confirm whether Bi-WaveX arrived at the correct output, its score and CIGAR were taken and compared to the score and CIGAR given by the sequential BiWFA library, which serves as the ground truth. If the outputs match, then it can be said that Bi-WaveX is correct for that particular test case as BiWFA returns the optimal alignment.

As shown in Table 4, Bi-WaveX was tested for correctness for all available test cases, and it produced the correct outputs for all 5800 test cases, exhibiting a 100% correctness rate. This means that Bi-WaveX returns the exact same output, and therefore the optimal alignment, as the original BiWFA library.

Table 4. Total Reads and Correct Reads for Substitution and Indel Test Cases

Error Rate	10K		30K		50K		80K		100K	
	Total	Correct	Total	Correct	Total	Correct	Total	Correct	Total	Correct
0%	200	200	200	200	200	200	200	200	200	200
1%	200	200	200	200	200	200	200	200	200	200
5%	200	200	200	200	200	200	200	200	200	200
10%	160	160	160	160	160	160	160	160	160	160
20%	120	120	120	120	120	120	120	120	120	120
30%	80	80	80	80	80	80	80	80	80	80
40%	40	40	40	40	40	40	40	40	40	40

3.2 Average Execution Time

Table 5. Execution time (in seconds) of pairwise alignment implementations on substitution test cases

Method	Time (s)					
	Error Rate	10K	30K	50K	80K	100K
Bi-WaveX	0%	0.021	0.020	0.022	0.025	0.026
	1%	0.019	0.028	0.039	0.059	0.096
	5%	0.036	0.149	0.344	0.864	1.377
	10%	0.077	0.502	1.270	3.199	5.306
	20%	0.248	2.014	4.848	12.727	20.700
	30%	0.483	4.252	10.532	30.532	45.214
	40%	0.823	6.963	18.419	46.899	79.059
BiWFA-Intrinsics	0%	0.021	0.024	0.019	0.032	0.027
	1%	0.021	0.027	0.034	0.064	0.095
	5%	0.039	0.137	0.327	0.853	1.328
	10%	0.079	0.476	1.255	3.405	5.072
	20%	0.238	1.836	4.774	12.703	19.960
	30%	0.461	3.936	10.445	28.438	44.433
	40%	0.812	6.671	18.472	49.781	74.888
BiWFA-Sequential	0%	0.023	0.022	0.020	0.032	0.027
	1%	0.026	0.028	0.036	0.065	0.098
	5%	0.045	0.143	0.352	0.952	1.422
	10%	0.096	0.501	1.343	3.842	5.334
	20%	0.296	1.933	5.097	14.084	19.914
	30%	0.598	4.282	11.521	28.609	45.539
	40%	1.030	7.232	19.895	49.613	78.343

Substitution Cases Table 5 shows the average execution times of all of the libraries across various error rates of the test cases. It is apparent that the versions containing AVX are consistently faster than the sequential version. Between Bi-WaveX and Intrinsics, the differences are minimal in lower text lengths. However, at 80Kbp onward, the intrinsics implementation runs faster than Bi-WaveX.

Table 6. Time performance of pairwise alignment implementations on indel test cases

Method	Error Rate	Time (s) 10K	30K	50K	80K	100K
Bi-WaveX	0%	0.016	0.018	0.020	0.025	0.027
	1%	0.019	0.028	0.048	0.094	0.164
	5%	0.046	0.242	0.623	1.656	2.645
	10%	0.116	0.850	2.257	6.094	9.582
	20%	0.352	2.911	7.732	20.913	34.220
	30%	0.662	5.500	15.429	41.614	66.920
	40%	0.983	8.240	22.681	59.407	100.227
BiWFA-Intrinsics	0%	0.016	0.019	0.020	0.024	0.028
	1%	0.020	0.031	0.048	0.093	0.163
	5%	0.046	0.243	0.636	1.572	2.565
	10%	0.117	0.859	2.309	5.762	9.093
	20%	0.353	2.918	7.885	20.074	32.873
	30%	0.667	5.618	15.295	39.589	62.072
	40%	0.999	8.372	23.005	60.060	93.021
BiWFA-Sequential	0%	0.018	0.019	0.021	0.021	0.027
	1%	0.021	0.032	0.053	0.092	0.162
	5%	0.050	0.267	0.693	1.593	2.678
	10%	0.128	0.948	2.530	5.858	9.988
	20%	0.386	3.209	8.682	20.705	33.458
	30%	0.734	6.165	16.771	43.208	66.830
	40%	1.087	9.211	25.308	65.586	100.583

Indel Cases Table 6 shows a similar trend to the substitution cases wherein Bi-WaveX and Intrinsics are consistently faster than the Sequential implementation, however specifically for test cases of length 100K the performance of intrinsics is substantially better compared to lower sequence length cases.

Best Case Table 7 shows the best case of Bi-WaveX, particularly which test case produced the largest speedup as compared to the sequential implementation. The execution time of the other libraries on the same test case is also shown for reference. Bi-WaveX exhibits a maximum of 4.67x increase in speedup as compared to BiWFA-Sequential, and is about 2x faster than BiWFA intrinsics for this particular test case. On the other hand, BiWFA-Intrinsics has a maximum of 3.56x increase in speed, and even in this best-case scenario, Bi-WaveX is competitive with it in terms of execution time.

Table 7. Fastest Speedup of BiWFA-Intrinsics and Bi-WaveX Compared to BiWFA-Sequential

	Libraries	
	BiWFA-Intrinsics	**Bi-WaveX**
Speedup	3.56x	4.67x
BiWFA-Sequential	3134 ms	80.04 ms
BiWFA-Intrinsics	879.8 ms	34.88 ms
Bi-WaveX	874.3 ms	17.13 ms
Error Rate	5%	1%
Text Length	80K	10K

On average, Bi-WaveX achieves a 12% speedup over the sequential BiWFA, with peak performance reaching 4.67× in low-error, medium-length cases.

Time Complexity

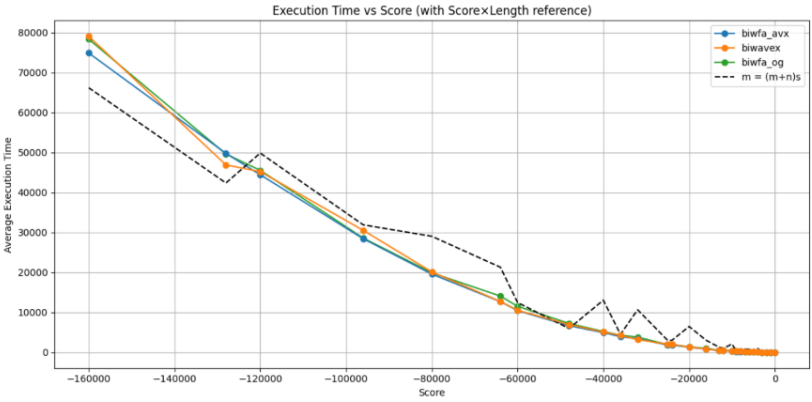

Fig. 2. A line graph of Execution Time vs Score,with a dotted line representing the function of the algorithm's time complexity

In terms of the time complexity of the algorithm, BiWFA-Sequential had a time complexity of $\mathbf{O}((m+n)s)$ where m and n refer to the pattern and text lengths of the input, and s refers to the outputted score of the alignment, using this as a basis, the researchers compared the execution times of the algorithm to see if it falls within the bounds of the worst case scenario as there are concerns that the modifications to the algorithm may have caused its overall time complexity to change. However, as shown in figure 2, the algorithm performs within bounds.

Memory

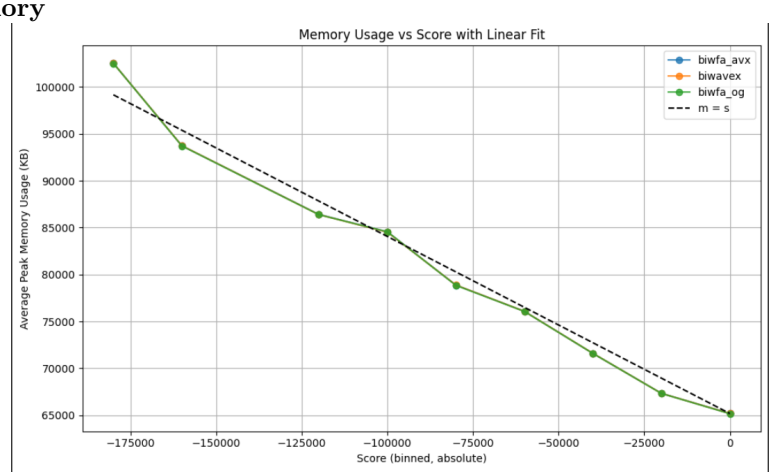

Fig. 3. A line graph of Memory Usage vs the Score

As shown in Table 3, the memory usage of the three implementations of the algorithm consumed the same amount of memory for all of the tests that were run whether they were a substitution or indel case; the introduction of SIMD into the program flow did not affect the overall memory usage.

3.3 Synthesis

A common trend present in both the substitution and indel cases is that the increase in speed is relatively similar until the 100Kbp test cases wherein BiWFA-Intrinsics is faster than Bi-WaveX by about 6-7%, especially in test cases with error rates of 30% and 40%, which indicates that the implementation has issues when encountering greater amounts of substitutions and/or indels, as the trend is present on both categories of test cases. This is likely caused by an increased amount of iterations as input size grows. Due to the modular design of Bi-WaveX, data structures are continually transformed into forms that better align with AVX-512 before being passed to the corresponding assembly function. This results in overhead that may offset the benefits provided by using SIMD.

The slowdowns of the AVX versions relative to the sequential version was also investigated. However, outside of outlier cases, the slowdowns are largely negligible across different test cases. Scrutinizing the particular patterns wherein slowdowns occurred yielded no clear patterns, and more test data and metrics would need to be used to get clearer trends. Additionally, the relatively modest improvements provided by both AVX implementations are likely due to the nature of SIMD itself. Since SIMD is data-level parallelism, the speedup provided would be smaller than a multi-threaded or algorithmic level of parallelism. Its

benefits also become more pronounced in more computationally intensive operations such as multiplication or division, which are largely absent from BiWFA.

Overall, the intrinsics-based version and Bi-WaveX both prove to be consistently faster than the sequential version. The difference in execution time between the two AVX versions is largely negligible, wherein either version can be faster than the other. The significant exception to this would be the 100Kbp test cases, wherein the intrinsics version is often faster than Bi-WaveX.

4 Future Works

The resulting implementation Bi-WaveX returns the correct output and exhibits speedups averaging at 12% compared to the sequential version, up to a maximum speedup of 367%. Compared to the intrinsics version, Bi-WaveX performed variably but was able to reach a maximum speedup of about 2x.

The primary drawback of Bi-WaveX is its modular implementation. Due to it not being an implementation of the algorithm from scratch, Bi-WaveX was unable to take advantage of the full capabilities of AVX-512. Thus, while the modular approach to development did save large amounts of time, it came at the cost of the potential performance of the implementation itself. For instance, the memory initialization of the data structures could be modified to better suit aligned vector assignment operations, or certain extra instructions could be foregone if the larger program was written in assembly.

Additionally, Bi-WaveX only supports global alignment, so future implementations could attempt to adapt the algorithm and its vectorization to support other scopes of alignment, such as semi-global or local alignment.

The other potential room for improvement is the integration of Bi-WaveX with SIMT multi-threading, which could potentially present an even greater speedup as compared to either type of parallelism alone. As discussed prior, SIMT implementations of the WFA algorithm already exists, so one could attempt to combine both implementations.

Finally, more robust testing can still be performed on Bi-WaveX. Other metrics such as memory footprint and scalability were also not tested in this paper, so further testing can be done in that regard as well. The implementation can also be tested against other pairwise alignment libraries, such as the other parallel implementations of the BiWFA or WFA algorithm. Additionally, stronger hardware is also recommended when testing these pairwise alignment algorithms, so as to eliminate potential bottlenecks introduced by weaker hardware.

References

1. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology **48**(3), 443–453 (1970). DOI 10.1016/0022-2836(70)90057-4
2. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. Journal of Molecular Biology **147**(1), 195–197 (1981). DOI 10.1016/

0022-2836(81)90087-5. URL https://www.sciencedirect.com/science/article/pii/0022283681900875

3. Gotoh, O.: An improved algorithm for matching biological sequences. Journal of Molecular Biology **162**(3), 705–708 (1982). DOI 10.1016/0022-2836(82)90398-9. URL https://www.sciencedirect.com/science/article/pii/0022283682903989
4. Pearson, W.R., Lipman, D.J.: Improved tools for biological sequence comparison. Proceedings of the National Academy of Sciences of the United States of America **85**(8), 2444–2448 (1988). DOI 10.1073/pnas.85.8.2444
5. Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J.: Basic local alignment search tool. Journal of Molecular Biology **215**(3), 403–410 (1990). DOI 10.1016/S0022-2836(05)80360-2. URL https://www.sciencedirect.com/science/article/pii/S0022283605803602
6. Marco-Sola, S., Moure, J.C., Moreto, M., Espinosa, A.: Fast gap-affine pairwise alignment using the wavefront algorithm. Bioinformatics **37**(4), 456–463 (2021). DOI 10.1093/bioinformatics/btaa777. URL https://doi.org/10.1093/bioinformatics/btaa777
7. Marco-Sola, S., Eizenga, J.M., Guarracino, A., Paten, B., Garrison, E., Moreto, M.: Optimal gap-affine alignment in O(s) space. Bioinformatics **39**(2), btad074 (2023). DOI 10.1093/bioinformatics/btad074. URL https://doi.org/10.1093/bioinformatics/btad074
8. S., A., S., P.V., Varghese, K.: An fpga based accelerator of the bi-directional wavefront algorithm for pairwise sequence alignment. In: 2023 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS), pp. 40–44 (2023). DOI 10.1109/APCCAS60141.2023.00021. URL https://ieeexplore.ieee.org/document/10509910. ISSN: 2768-3516
9. Haghi, A., Marco-Sola, S., Alvarez, L., Diamantopoulos, D., Hagleitner, C., Moreto, M.: Wfa-fpga: An efficient accelerator of the wavefront algorithm for short and long read genomics alignment. Future Generation Computer Systems **149**, 39–58 (2023). DOI 10.1016/j.future.2023.07.008. URL https://www.sciencedirect.com/science/article/pii/S0167739X2300256X
10. Aguado-Puig, Q., Marco-Sola, S., Moure, J.C., Matzoros, C., Castells-Rufas, D., Espinosa, A., Moreto, M.: WFA-GPU: Gap-affine pairwise alignment using GPUs (2022). DOI 10.1101/2022.04.18.488374. URL https://www.biorxiv.org/content/10.1101/2022.04.18.488374v1. Pages: 2022.04.18.488374 Section: New Results
11. Alonso-Marín, A., Fernandez, I., Aguado-Puig, Q., Gómez-Luna, J., Marco-Sola, S., Moreto, M.: BIMSA: Accelerating Long Sequence Alignment Using Processing-In-Memory (2024). DOI 10.1101/2024.05.10.593513. URL https://www.biorxiv.org/content/10.1101/2024.05.10.593513v2. Pages: 2024.05.10.593513 Section: Confirmatory Results
12. Zhang, H.: Overview of Sequence Data Formats. Methods in Molecular Biology (Clifton, N.J.) **1418**, 3–17 (2016). DOI 10.1007/978-1-4939-3578-9_1
13. Chen, J.Q., Wu, Y., Yang, H., Bergelson, J., Kreitman, M., Tian, D.: Variation in the Ratio of Nucleotide Substitution and Indel Rates across Genomes in Mammals and Bacteria. Molecular Biology and Evolution **26**(7), 1523–1531 (2009). DOI 10.1093/molbev/msp063. URL https://dx.doi.org/10.1093/molbev/msp063. Publisher: Oxford Academic
14. Wygoda, E., Loewenthal, G., Moshe, A., Alburquerque, M., Mayrose, I., Pupko, T.: Statistical framework to determine indel-length distribution. Bioinformatics **40**(2), btae043 (2024). DOI 10.1093/bioinformatics/btae043. URL https://doi.org/10.1093/bioinformatics/btae043