Accelerating Zhang's Six-Frame Alignment Algorithm via Hybrid SIMT Wavefront Parallelization on CUDA

Althea Zyrie Arceta[1,2], Antonio Gabriel Mendoza[1,3] Jose Tristan Tan[1,4], and Roger Luis Uy[1,5]

[1] Department of Computer Technology, De La Salle University, Manila, Philippines
[2] altheazyrie.arceta@gmail.com
[3] agmndoza@gmail.com
[4] josetristantan@gmail.com
[5] roger.uy@dlsu.edu.ph

Abstract. Zhang's six-frame Alignment algorithm is a type of sequence alignment used to identify similarities between DNA and protein sequences and has a wide range of applications in bioinformatics. Zhang's six-frame translates the DNA sequence across all six possible reading frames to specifically account for frameshift errors and variations. Implementing Zhang's six-frame sequentially can be computationally expensive, particularly when dealing with large datasets. To address the said issue, this study explores a parallel computing approach using NVIDIA's CUDA programming model to speed up the implementation of Zhang's six-frame algorithm. Results showed speedups averaging 3.51x up to 6.03x for the *Drosophila melanogaster* dataset and speedups averaging 3.44x up to 6.44x for the *Arabidopsis thaliana* dataset when compared to the sequential implementation.

Keywords: Zhang's six-frame alignment, DNA-Protein alignment, SIMT, CUDA, Parallel Computing, High Performance Computing

1 Introduction

Sequence alignment is the process of arranging DNA, RNA, or protein sequences to detect regions of similarity [1], which is crucial in understanding the functional, structural, and evolutionary connections between them [2]. Achieving faster and more efficient sequence alignment is highly crucial given its role as the foundational first stage in genomic workflows [3]. As such, the acceleration of this process benefits researchers, enabling them to proceed more rapidly with downstream analysis, such as evolutionary biology, proteomics, and genomics. Consequently, sequence alignment provides valuable information for applications like personalized medicine, where aligning patient sequences helps uncover mutations, and drug research, where finding conserved protein domains can help with target selection.

DNA-protein alignment is a type of sequence alignment that compares DNA read sequences to a reference proteome. Zhang's six-frame alignment algorithm [5] is a

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_20

DNA-protein alignment algorithm used to compare DNA sequences that experience sequencing errors with a reference protein sequence. It uses six frames that are obtained from the forward and reverse complement strands of the DNA sequence. The algorithm computes the alignment of the six frames by using the three-frame alignment algorithm [6] twice, accounting for both directional strands. Although Zhang's six-frame alignment algorithm can be considered a well-established method [7], its sequential implementation can be computationally expensive [8]. Thus, a need for efficient solutions to sequential implementations arises to keep pace with the growth of data used in such algorithms.

Researchers often utilize parallel computing techniques to enhance the execution of these algorithms, leading to faster alignments and the ability to process larger datasets. Single Instruction, Multiple Data (SIMD) and Single Instruction, Multiple Threads (SIMT) are two parallel computing paradigms that enable effective parallel processing [9]. SIMD achieves parallelism by allowing a single instruction to operate on multiple data elements simultaneously through vectorization, utilizing multiple execution units. It provides a variety of feasible approaches for faster algorithm executions and is used in most applications that require large data processing [4]. A SIMD implementation of Zhang's six-frame algorithm [5] using AVX2 applied a staircase-like method to respect the data dependencies inherent in the three-frame alignment algorithm. This resulted in a threefold increase in execution speed over the sequential implementation. On the other hand, Graphics Processing Units (GPUs) use the SIMT execution model for parallel computing. This method allows a large number of distinct threads to execute the same instruction concurrently while operating on different data [10]. This makes the utilization of GPUs a viable approach for computational workloads that can be divided across multiple processing units. NVIDIA's Compute Unified Device Architecture (CUDA), an asynchronous SIMT programming model, is a parallel computing platform developed for general-purpose computing on graphics processing units (GPGPU) specifically for NVIDIA GPUs.

At the time of this study, no direct implementations of DNA-protein alignment algorithms exist in the SIMT paradigm. Hence, this research aims to implement Zhang's six-frame Alignment Algorithm in CUDA. Given that, this paper will discuss the following: Frameshift Alignment Algorithm in Section 2, the SIMT execution model and CUDA parallel computing platform in Section 3, the system overview in Section 4, the parallel implementation of the scoring phase of the algorithm in Section 5, the tests, results, and discussion in Section 6, and finally, the conclusion and future works will be discussed in Section 7.

2 Frameshift Alignment Algorithm

2.1 Three-Frame Alignment

The original Zhang's three-frame alignment algorithm [11] was designed to perform global alignment between DNA and protein sequences. It operates solely on the forward strand of DNA, examining three possible reading frames, as seen in Figure 1, from the 5' to 3' direction.

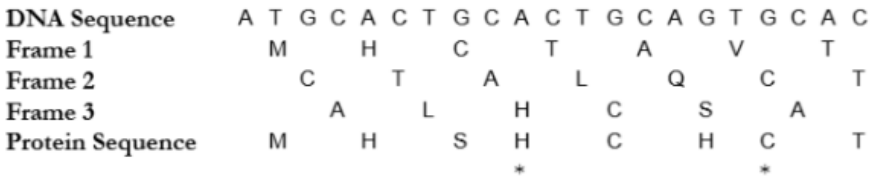

Fig. 1: Three-Frame DNA-Protein Alignment (Asterisks mean frameshift)

The approach utilizes a set of matrices, namely the scoring (C), insertion (I), and deletion (D) matrices, that are initialized with penalties for gaps, which produces an alignment that stretches across the entire length of the input sequences. The approach requires the I and D matrices to first be initialized, as seen in Equations 1-3, since the initialization of the C matrix is dependent on the values found in the two matrices, as seen in Equations 4-10.

$$I(j,0) = -\infty \tag{1}$$

$$D(0,j) = D(2,j) = D(3,j) = -\infty \tag{2}$$

$$D(1,j) = C(0,j) - gop - gep \tag{3}$$

$$C(0,0) = 0 \tag{4}$$

$$C(0,j) = I(0,j) \tag{5}$$

$$C(j,0) = D(j,0) \tag{6}$$

$$C(1,j) = \max \begin{cases} I(1,j) \\ D(1,j) \\ C(0,j-1) + S[b_1, a_j] \end{cases} \tag{7}$$

$$C(2,j) = \max \begin{cases} I(2,j) \\ C(0,j-1) + S[b_1, a_j] - \pi \end{cases} \tag{8}$$

$$C(3,j) = \max \begin{cases} I(3,j) \\ C(0,j-1) + S[b_2, a_j] - \pi \end{cases} \tag{9}$$

$$C(4,j) = \max \begin{cases} I(4,j) \\ D(4,j) \\ C(1,j-1) + S[b_4, a_j] \\ C(2,j-1) + S[b_4, a_j] - \pi \end{cases} \tag{10}$$

Once initialized, the algorithm begins processing the remainder of the matrix. For each cell, the algorithm uses the recurrence relations found in Equations 11-13 to

decide the best possible scores by comparing the different alignment paths. Finally, Equation 14 is used to fill up the last cell of matrix C.

$$I(i,j) = \max \begin{cases} I(i,j-1) - gep \\ C(i,j-1) - gep - gop \end{cases} \tag{11}$$

$$D(i,j) = \max \begin{cases} D(i-3,j) - gep \\ C(i,j-1) - gep - gop \end{cases} \tag{12}$$

$$C(i,j) = \max \begin{cases} I(i,j) \\ D(i,j) \\ C(i-2,j-1) + S[b_i, a_j] - \pi \\ C(i-3,j-1) + S[b_i, a_j] \\ C(i-4,j-1) + S[b_i, a_j] - \pi \end{cases} \tag{13}$$

$$C(N,M) = \max \begin{cases} C(N-1,M) \\ C(N-2,M) - \pi \\ C(N-3,M) - gop - gep - \pi \\ D(N-3,M) - \pi - gep \end{cases} \tag{14}$$

2.2 Modified Three-Frame Alignment

In 2021, a modified three-frame alignment version [6] was developed. The modified version of the algorithm adopts local alignment instead of global alignment. This shift is reflected in the initialization of the C matrix, where the first row is set to zero instead of applying a gap penalty as seen in Equation 15. Additionally, for the computation of the scores in the C matrix, negative values will be set to 0 to allow for local alignment.

$$C(0,j) = 0 \tag{15}$$

2.3 Six-Frame Alignment

On the other hand, Zhang's six-frame alignment algorithm [5] expands the analysis to cover all six possible reading frames. This includes the three frames from the forward strand and three additional frames from the reverse complement strand, as seen in Figure 2. The algorithm runs the modified three-frame local alignment twice: first on the original strand, and second on its reverse complement. Because each three-frame alignment involves a dynamic programming matrix with strong inter-cell data dependencies, performing two such passes significantly increases runtime—making the six-frame variant especially suitable for parallel acceleration.

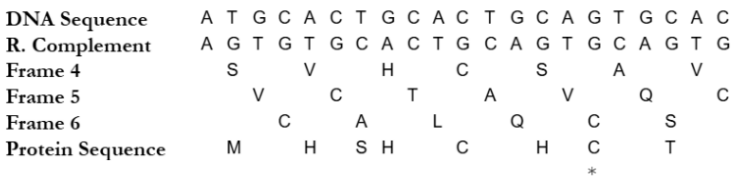

Fig. 2: Six-Frame DNA-Protein Alignment

3 SIMT & CUDA

CUDA, specifically designed for NVIDIA GPUs, offers a full programming environment for GPGPU processing [12].

CUDA's interface enables high-performance computing tasks that would otherwise be inefficient in traditional CPU implementations. CUDA's design allows several threads to execute identical instructions on different data streams, resulting in a massively parallel computation. Unlike CPU-based vectorization (SIMD), which processes fixed-width data lanes, SIMT on GPUs dynamically schedules thousands of lightweight threads, allowing irregular or dependency-constrained workloads—like dynamic programming—to be parallelized across wavefronts of independent cells. CUDA utilizes two forms of parallelism: data parallelism, and task parallelism. Data parallelism is concurrently running the same operation across multiple data streams, which is typically used in large-scale matrix or vector operations. In contrast, task parallelism is concurrently running various operations on different threads [13].

4 System Overview

4.1 Dataset & Inputs

The datasets used in this research consist of reference proteomes and DNA reads for *Drosophila melanogaster* and *Arabidopsis thaliana* from UniProt and ENA [14–16]. The program accepts two inputs: the DNA read sequences and the reference proteome dataset.

4.2 Setup Configuration

The implementation uses the BLOSUM62 substitution matrix [17] with gap-open ($gop = 2$), gap-extend ($gep = 3$), and frameshift ($\pi = 4$) penalties.

4.3 Matrix Initialization

Before matrix scoring can be done, the C, I, and D matrices, along with their corresponding traceback matrices, must be initialized. The first few rows and columns of the matrices are initialized, as shown in Equations 1-4, 6-10, and 15. This preparation ensures that the algorithm's data dependencies are fulfilled before the matrix scoring recurrence is performed.

4.4 Matrix Scoring

Due to Zhang's data dependencies, both implementations must respect computation order. The Sequential Implementation processes the matrix in a straightforward row-major order, computing each cell one at a time from left to right and top to bottom. However, for the CUDA Implementation, a hybrid multi-level anti-diagonal wavefront technique was implemented (further discussed in Section 5).

4.5 Traceback

After scoring, traceback is performed to obtain the optimal alignment. It will start at the index where the highest score in the scoring matrix is located, and traces back from that position based on the values of the traceback matrix until a zero score is encountered.

5 Matrix Scoring Parallelization with CUDA

Zhang's recurrence relations impose strong data dependencies: computing cell (i, j) requires values from $(i-2, j-1)$, $(i-3, j-1)$, and $(i-4, j-1)$. This prevents row- or column-wise parallelization but permits computation along anti-diagonals (where $i+j$ is constant), as all cells on the same anti-diagonal are mutually independent. To exploit this, we use a hybrid multi-level anti-diagonal wavefront: – The host iterates over anti-diagonals of 32×32 tiles, launching one kernel per tile anti-diagonal. – Each kernel computes cells within its assigned tiles along internal anti-diagonals, using block synchronization to respect intra-tile dependencies. This two-tiered design enables maximal concurrency while preserving correctness.

5.1 CUDA Configuration

Given the hybrid nature of the implementation, the block size was set to 1024 (32x32) to fully leverage in-kernel block synchronization across the largest feasible submatrix. The grid size, in constrast, is dynamically determined within each iteration of the host-side loop based on the number of submatrices in the current antidiagonal.

5.2 Matrix Parallelization

The implemented hybrid anti-diagonal wavefront technique divides the overall problem matrices into submatrix tiles that can be processed concurrently along the macro-level matrix anti-diagonals.

Host Side The implementation utilizes a host-side loop that iterates through the macro-level anti-diagonals to manage the launching of kernels, as seen in Figure 3. In each iteration of the loop, a CUDA kernel is launched to compute one full anti-diagonal of 32×32 tiles across the matrices. This host-driven orchestration ensures that all tiles within a macro anti-diagonal are ready for concurrent execution,

respecting the inter-tile data dependencies imposed by Zhang's recurrence. The implementation also utilizes CUDA streams to enable the concurrent launching of the forward strand and reverse complement alignment kernels.

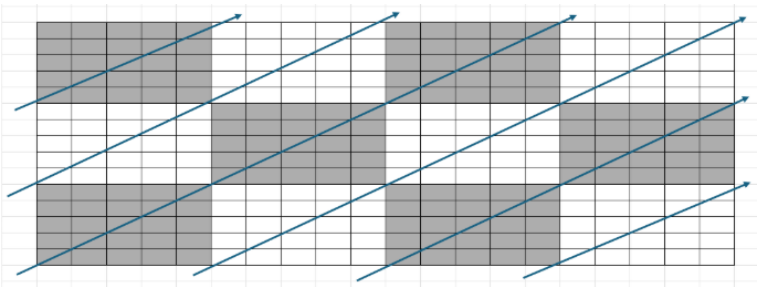

Fig. 3: Macro-Level Matrix Anti-Diagonals

CUDA Kernel The CUDA kernel is responsible for calculating the insertion, deletion, and substitution scores and updating the respective matrices in place. In addition to the scores, traversal values that will be used in the traceback are computed and stored. Within each assigned 32×32 tile, the kernel processes cells along internal anti-diagonals, using block-level synchronization to enforce intra-tile dependencies. The kernel utilizes a conditional anti-diagonal loop to ensure that only cells belonging to the current micro anti-diagonal are computed at each step. Block synchronization calls are performed in every iteration of the loop to guarantee that threads wait for required values from the previous anti-diagonal, preventing race conditions. Grid-stride calculations are performed within the kernel to allow a single kernel launch to process all tiles in the current macro anti-diagonal. A visualization of this implementation is seen in Figure 4.

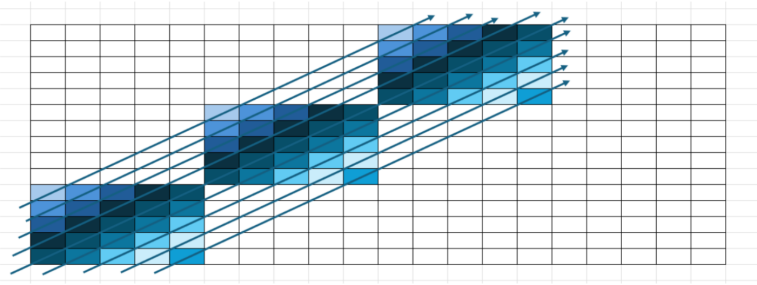

Fig. 4: Visualization of Grid-Stride Anti-Diagonal Computation in CUDA Kernel

6 Results & Discussion

This section presents the results and analysis from various tests conducted on the implemented system. The evaluation includes correctness validation (Section 6.1), execution time performance (Section 6.2), and scalability (Section 6.3).

Tests were conducted on a system with an Intel i7-12700F and an NVIDIA RTX 4070 Super with 32 GB of RAM. The source code is available at https://github.com/Shiray427/Zhangs-six-frame-in-CUDA.

6.1 Correctness

The correctness of the implementation was confirmed through manual inspection and external validation using BLASTX [18]. Both tests showed that the outputs, traceback outputs, scores, and matched protein IDs were consistent and accurate.

Manual Checking To verify the correctness of the implementation, DNA reads were aligned to known protein sequences using Excel. The matrices were manually computed based on the recurrence relations above. The computed alignments, including start and end indices, scores, and selected protein IDs, matched the implementation outputs. Several test cases, such as perfect matches, insertions, deletions, and combinations, were tested and returned correct results.

Sequential							Parallel							Correctness Test		
Protein	Length	Score	s_index	e_index	ime(in sec)		Protein Inde	Length	Score	s_Index	e_index	Time (in ms)		Score	s_Index	e_index
0	774	48	561	596	558.071	Top Score	0	774	48	561	596	37.59821	Top Score	TRUE	TRUE	TRUE
1	88	38	56	76	61.0324	198	1	88	38	56	76	32.1792	198	TRUE	TRUE	TRUE
2	2718	48	1092	1131	1909.944	Found in	2	2718	48	1092	1131	107.6316	Found in	TRUE	TRUE	TRUE
3	1866	47	850	879	1300.69	30	3	1866	47	850	879	80.23142	30	TRUE	TRUE	TRUE
4	965	45	806	852	668.9748		4	965	45	806	852	63.36205		TRUE	TRUE	TRUE
5	780	42	714	753	542.543		5	780	42	714	753	56.43776		TRUE	TRUE	TRUE
6	343	46	26	60	237.2557		6	343	46	26	60	40.67533		TRUE	TRUE	TRUE
7	834	36	110	138	576.932		7	834	36	110	138	56.05376		TRUE	TRUE	TRUE
8	2036	53	334	366	1411.246		8	2036	53	334	366	87.3943		TRUE	TRUE	TRUE
9	400	40	293	334	274.6718		9	400	40	293	334	59.59885		TRUE	TRUE	TRUE
10	2176	62	1919	1960	1509.137		10	2176	62	1919	1960	96.27853		TRUE	TRUE	TRUE
11	446	38	58	101	306.3408		11	446	38	58	101	48.13926		TRUE	TRUE	TRUE

Fig. 5: Correctness Verification of Alignment Results

External Cross Checking To further ensure correctness, the results of the implementation were compared with the results of a certified gold standard algorithm [19], BLASTX. Using the same dataset and inputs, both the CUDA implementation and BLASTX produced similar alignment paths and identified the same matched proteins. Differences in alignment scores were attributed to BLASTX's early termination heuristics, but the output regions and protein IDs remained consistent, confirming the accuracy of the implementation.

6.2 Execution Time

To evaluate the performance of the implementation, the runtime of both sequential and CUDA (SIMT) implementations were tested. Results were averaged over 30 runs of the program to eliminate variability and observe consistent behavior. The inputs used were two 150 length DNA read sequences and the entire reference proteome datasets, with the fruit fly dataset containing 21,449 protein sequences ranging from lengths 11 to 22,949, and the mouse-ear cress dataset containing 39,116 protein sequences ranging from lengths 5 to 5,400.

Drosophila melanogaster For the fruit fly dataset, the program obtained a total runtime of 153.15 s for sequential and 37.10 s for SIMT, which results in a total speedup of 4.13x. The average runtimes obtained are 7.14 ms and 1.73 ms respectively, and has a mean speedup of 3.51x and a median speedup of 3.48x. Results are summarized in Tables 1-2 and visualized in Figures 6-7.

The results show that the CUDA implementation offers significant speedups, especially for longer sequences; however, for shorter sequences, it can become slower. This performance behavior is expected in GPU-based systems, where parallel overhead favors batch processing of larger, more complex workloads. For shorter sequences, this overhead may overtake any potential speedups by the implemented parallelism, resulting in slowdowns.

Table 1: Summary Statistics for Sequential and SIMT Implementation (for Fruit Fly Dataset)

	Sequential (in ms)	**SIMT (in ms)**	**Speedup**
Mean	7.1402	1.7300	3.5104x
Median	4.5557	1.3112	3.4797x
Standard Deviation	10.8383	1.9551	–
Min	0.1606	0.2903	0.5468x
Max	245.6083	48.7752	6.0252x

Table 2: Total Runtime and Speedup for Sequential and SIMT Implementation (for Fruit Fly Dataset)

Sequential (in ms)	**SIMT (in ms)**	**Speedup**
153150.4721	37106.7840	4.1273x

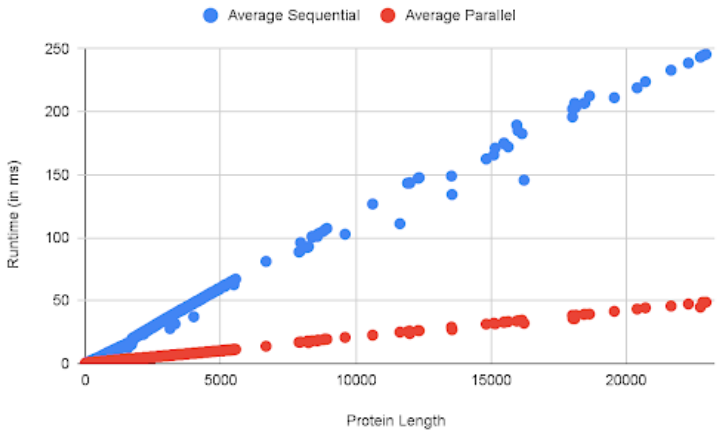

Fig. 6: Runtime vs. Sequence Length for Fruit Fly Dataset

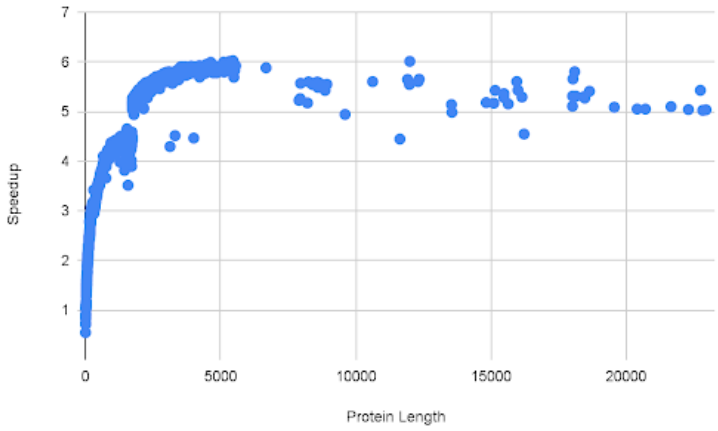

Fig. 7: Speedup vs. Sequence Length for Fruit Fly Dataset

Arabidopsis thaliana For the mouse-ear cress dataset, the program obtained a total runtime of 134.91 s for sequential and 35.29 s for SIMT, which results in a total speedup of 3.8224x. The average runtimes obtained were 3.45 ms and 0.90 ms, respectively. Similar results for the mean and median speedups were also obtained, 3.44x and 3.41x, respectively. Results are summarized in Tables 3-4 and visualized in Figures 8-9.

Table 3: Summary Statistics for SIMT and Sequential Implementation (for Mouse Ear Cress Dataset)

	Sequential (in ms)	**SIMT (in ms)**	**Speedup**
Mean	3.4489	0.9023	3.4412x
Median	2.5876	0.7459	3.4098x
Standard Deviation	3.1978	0.6162	–
Min	0.0822	0.1450	0.2954x
Max	65.0062	10.9524	6.4371x

Table 4: Total Runtime and Speedup for Sequential and SIMT Implementation (for Mouse Ear Cress Dataset)

Sequential (in ms)	**SIMT (in ms)**	**Speedup**
134908.8647	35293.6498	3.8224x

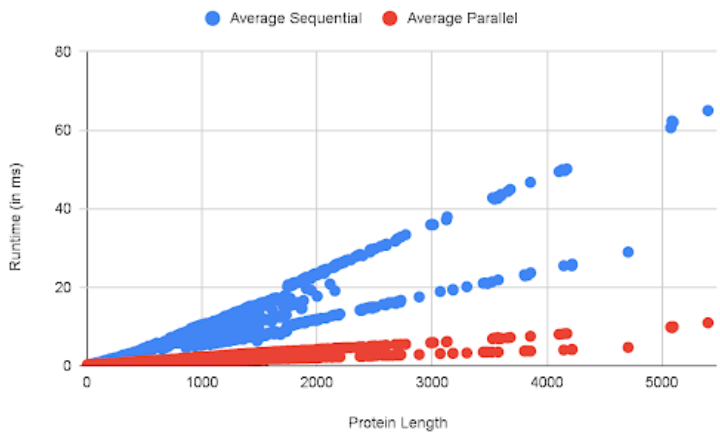

Fig. 8: Runtime vs. Sequence Length for Mouse Ear Cress Dataset

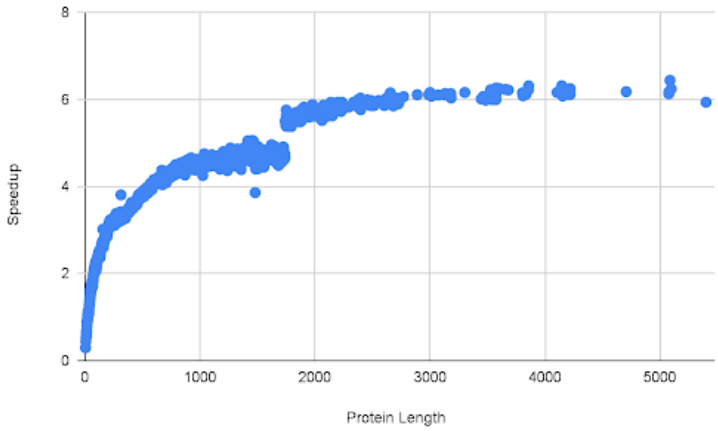

Fig. 9: Speedup vs. Sequence Length for Mouse Ear Cress Dataset

These results further demonstrate the consistent behavior of the CUDA implementation across datasets: very short sequences incur GPU overhead, leading to reduced speedups or slowdowns, while longer sequences increasingly benefit from parallel execution.

6.3 Scalability

In theory, the sequential implementation of Zhang's six-frame Alignment has a time complexity of $\mathcal{O}(nm)$, where n is the DNA length and m is the protein length, due to nested cell dependencies. The CUDA-based parallel version, using a hybrid multi-level anti-diagonal wavefront technique, reduces this to $\mathcal{O}(n+m-1)$ in practice by enabling concurrent computation along anti-diagonals. To assess this scalability, a $2{\times}2^k$ test was conducted, aligning two DNA inputs against increasing numbers of reference sequences, from 2^8 to 2^{14} for Fruit Fly, and 2^8 to 2^{15} for Mouse-Ear Cress. See Figures 10 and 11 for results.

These results align well with theoretical expectations, where the scalability of the parallel approach shows reduced complexity. Results show that the CUDA implementation achieves significantly better performance as input sizes grow, validating the effectiveness of the implementation. Initially, at smaller sequence counts, the difference in execution times is also relatively small. As the sequence count increases, however, the absolute difference between the two implementations becomes significantly larger. This indicates that the parallel implementation scales more favorably as datasets become larger. While both implementations curve upward, the parallel execution time curve's slope is noticeably less steep. Thus, the results clearly show that the parallel implementation exhibits significantly better scalability as compared to the sequential one, and grows at a much slower rate.

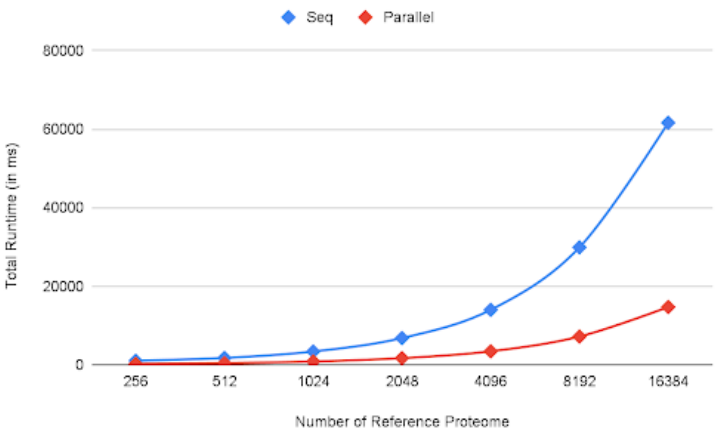

Fig. 10: Scalability of Fruit Fly Dataset

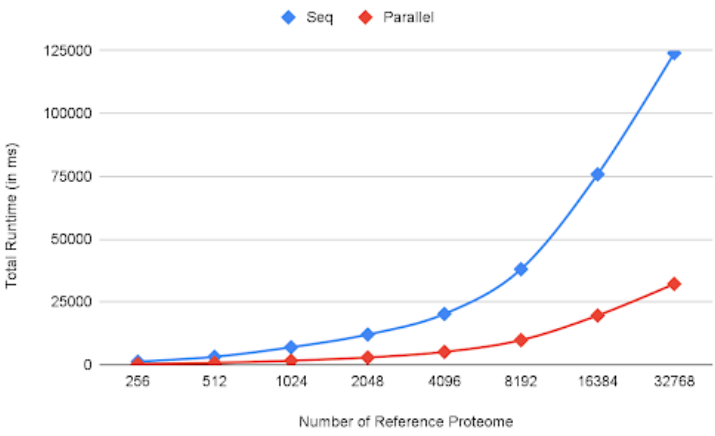

Fig. 11: Scalability of Mouse Ear Cress Dataset

7 Conclusion

This study demonstrated the capabilities of CUDA, a SIMT execution model, to speed up the execution of Zhang's six-frame algorithm that possesses inherent data dependencies. This work demonstrates that the anti-diagonal paradigm can be effectively implemented in CUDA SIMT, and achieved this by leveraging a hybrid multi-level anti-diagonal wavefront technique, a modified version of the anti-diagonal technique to fit the parallel framework of the SIMT model. The implementation achieves a mean speedup of approximately 3.5x, validating the effectiveness of the hybrid wavefront design.

Future work includes a purely device-side implementation to eliminate host-side overhead and enable deeper GPU optimization using tools like NVIDIA Nsight, helping to analyze performance bottlenecks in the researchers' implementations. Speedups can also be explored in the other phases of the program, such as initialization and setup.

Finally, this study serves as proof of concept for the utilization of GPUs in the field of bioinformatics, especially for more complex algorithms such as Zhang's Six Frame. This will enable faster processing of data in the field of bioinformatics, which will allow faster downstream analysis in the field of genetics.

References

1. A. E. E. -D. Rashed, H. M. Amer, M. El-Seddek and H. E. -D. Moustafa, "Sequence Alignment Using Machine Learning-Based Needleman–Wunsch Algorithm," in IEEE Access, vol. 9, pp. 109522-109535, 2021, doi: 10.1109/ACCESS.2021.3100408.
2. M. Pop and S. F. Altschul, "Sequence Alignment," in Handbook of Discrete and Combinatorial Mathematics, 2nd ed., K. H. Rosen, D. R. Shier, and W. Goddard, Eds. Boca Raton, 2017, ch. 20.1. Available: https://www.ncbi.nlm.nih.gov/books/NBK464187/
3. R. Stark, M. Grzelak, and J. Hadfield, "RNA sequencing: the teenage years," Nature Reviews Genetics, vol. 20, no. 11, pp. 631–656, Jul. 2019, doi: https://doi.org/10.1038/s41576-019-0150-2.
4. M.S. Yu, C.Y. Yuan, T.L. Chen, and J.-K. Lee, "Case Study: Optimization Methods with TVM Hybrid-OP on RISC-V Packed SIMD," ResearchGate, Jan. 2024. https://doi.org/10.1109/ACCESS.2024.3397195
5. S. V. Lim, S. E. Lim, C. L. Ting, A. E. Wong, and R. L. Uy, "SIMD Implementation of Local Alignment-Based six-frame Alignment Algorithm," Philippine Computing Journal, vol. 17, no. 1, 2023.
6. S. V. Lim, S. E. Lim, C. L. Ting, A. E. Wong, and R. L. Uy, "SIMD Implementation of Modified Zhang's Three-Frame Alignment Algorithm," 2021 IEEE 13th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment, and Management (HNICEM), pp. 1–6, Nov. 2021, doi: https://doi.org/10.1109/hnicem54116.2021.9731997.
7. Y. Yao and M. C. Frith, "Improved DNA-Versus-Protein Homology Search for Protein Fossils," IEEE/ACM Transactions on Computational Biology and Bioinformatics, vol. 20, no. 3, pp. 1691–1699, May 2022, doi: https://doi.org/10.1109/tcbb.2022.3177855.
8. S. Sarkar, Gaurav Ramesh Kulkarni, Partha Pratim Pande, and Ananth Kalyanaraman, "Network-on-Chip Hardware Accelerators for Biological Sequence Alignment," I.E.E.E.

transactions on computers/IEEE transactions on computers, vol. 59, no. 1, pp. 29–41, Jan. 2010, doi: https://doi.org/10.1109/tc.2009.133.
9. N. J. Curtis, K. E. Niemeyer, and C.-J. Sung, "Using SIMD and SIMT vectorization to evaluate sparse chemical kinetic Jacobian matrices and thermochemical source terms," Combustion and flame, vol. 198, pp. 186–204, Dec. 2018, doi: https://doi.org/10.1016/j.combustflame.2018.09.008.
10. NVIDIA, "Whitepaper NVIDIA's Next Generation CUDA TM Compute Architecture: Fermi TM V1.1.", 2009, Available: https://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf
11. Z. Zhang, W. R. Pearson, and W. Miller, "Aligning a DNA Sequence with a Protein Sequence," ResearchGate, Feb. 1997. https://www.researchgate.net/publication/13942672_Aligning_a_DNA_Sequence_with_a_Protein_Sequence.Œ
12. Nvidia, "CUDA Toolkit Documentation," Nvidia.com, Jun. 05, 2025. https://docs.nvidia.com/cuda/ (accessed Jul. 29, 2025).
13. A. Chakraborty, "Data parallelism vs Task parallelism," Tutorialspoint, Oct. 11, 2019. https://www.tutorialspoint.com/data-parallelism-vs-task-parallelism (accessed Jul. 29, 2025).
14. UniProt, "Drosophilia melanogaster (Fruit Fly)," UniProt, 2025. https://www.uniprot.org/proteomes/UP000000803 (accessed Jul. 29, 2025).
15. UniProt, "Arabidopsis thaliana (Mouse-ear cress)," UniProt, 2025. https://www.uniprot.org/proteomes/UP000006548 (accessed Jul. 29, 2025).
16. European Molecular Biology Laboratory, "European Nucleotide Archive," EMBL-EBI, 2025. https://www.ebi.ac.uk/ena/browser/home (accessed Jul. 29, 2025).
17. W. Li, B. Ma, and K. Zhang, "Amino Acid Classification and Hash Seeds for Homology Search," Lecture Notes in Computer Science, pp. 44–51, 2009, doi:https://doi.org/10.1007/978-3-642-00727-9_6.
18. National Library of Medicine. "blastx: search protein databases using a translated nucleotide query," BLAST, 2022. https://blast.ncbi.nlm.nih.gov/Blast.cgi?LINK_LOC=blasthome&PAGE_TYPE=BlastSearch&PROGRAM=blastx (accessed May 21, 2025).
19. B. Buchfink, C. Xie, and D. H. Huson, "Fast and sensitive protein alignment using DIAMOND," Nature Methods, vol. 12, no. 1, pp. 59–60, Nov. 2014, doi:https://doi.org/10.1038/nmeth.3176.