GPU-Accelerated BLAST-Like Sequence Alignment Using CUDA and K-mer Indexing

Abhiram Kandiyana, Harshitha Marichetty Sudhakar, Mounika Boggavarapu

Abstract

Sequence alignment is at the heart of computational biology, helping us uncover how genes and proteins evolve, function, and interact. In this project, we took on the challenge of making the Smith-Waterman algorithm faster. By using CUDA to harness the power of GPUs and applying k-mer indexing to narrow down search candidates, we achieved incredible speedups compared to traditional CPU methods. Even more exciting, our implementation produces results that are identical to BLAST, one of the most trusted tools in bioinformatics. This makes our system both fast and reliable, opening the door to analyzing much larger datasets.

1. Introduction

Sequence alignment is a critical tool for comparing DNA or protein sequences to understand their relationships. The Smith-Waterman algorithm, which provides highly accurate local alignments, is great for small datasets, but its high computational cost (quadratic time complexity) makes it impractical for large-scale use.

That's where this project comes in. We've reimagined Smith-Waterman for high-performance GPUs, using CUDA to run tasks in parallel and k-mer indexing to cut down on the number of sequences we need to align. The result? A tool that's not only fast but also matches the accuracy of BLAST.

Goals:

- Speed up sequence alignment by using GPU parallelism.
- Use k-mer indexing to reduce unnecessary computation.
- Match BLAST in alignment accuracy while processing large datasets faster.

2. Methods

2.1 CPU-Based Sequence Alignment

2.1.1 K-mer Indexing

- **K-mer Extraction:**
  - For each sequence in the reference database:
    - Iterate over the sequence, extracting substrings of length k (k-mers).
- **Index Construction:**
  - Create a hash table to store k-mers as keys and a list of corresponding sequence indices as values.
  - For each extracted k-mer:
    - If the k-mer exists in the hash table, append the current sequence index to its list of indices.
    - Otherwise, insert the k-mer into the hash table with a new list containing the current sequence index.

2.1.2 Reference Filtering

- **K-mer Matching:**
  - For each query sequence:
    - Extract k-mers from the query sequence.
    - For each query k-mer:
      - Check if the k-mer exists in the k-mer index.
      - If it exists, increment a counter for each corresponding reference sequence.
- **Top N Selection:**
  - Sort the candidate reference sequences based on their match counts.
  - Select the top N candidate sequences with the highest match counts.

2.1.3 Smith-Waterman Alignment

- **Initialization:**
  - Create a dynamic programming matrix M of size (m+1) x (n+1), where m and n are the lengths of the query and reference sequences, respectively.
  - Initialize the first row and column of M to zeros.
- **Filling the Matrix:**
  - For each cell (i, j) in the matrix:

        - Calculate the match score s based on the similarity of the characters at positions i and j in the query and reference sequences, respectively.
        - Calculate the deletion score d from the cell above.
        - Calculate the insertion score i from the cell to the left.
        - Assign the maximum of s, d, and i to the cell (i, j).
- **Traceback:**
    - Start from the cell with the highest score in the matrix.
    - Trace back through the matrix, following the path that led to the highest score at each step.
    - Record the aligned characters at each step.
    - Stop when reaching a cell with a score of 0.

2.2 GPU-Accelerated Sequence Alignment

2.2.1 GPU-Accelerated K-mer Indexing

- A CUDA kernel is designed to parallelize the k-mer extraction and indexing process.
- Each thread extracts k-mers from a specific portion of the reference sequence.
- The extracted k-mers are inserted into a shared memory hash table within each thread block.
- After processing the local portion, the thread block synchronizes and merges the shared memory hash tables into a global hash table.

2.2.2 GPU-Accelerated Smith-Waterman Alignment

- A CUDA kernel is designed to parallelize the dynamic programming and traceback steps of the Smith-Waterman algorithm.
- Each thread block is assigned a portion of the dynamic programming matrix to compute.
- Shared memory is used to store parts of the matrix within each thread block, reducing memory access latency.
- Thread blocks synchronize to ensure correct calculation of scores and traceback.

2.2.3 GPU Optimizations:

- **Memory Coalesced Access:**
  - Organize memory access patterns to maximize memory throughput.
  - Align data accesses to hardware cache lines to reduce latency.
- **Shared Memory Utilization:**
  - Use shared memory to store frequently accessed data, reducing global memory access latency.
  - Optimize shared memory usage to minimize bank conflicts.
- **Thread Block Optimization:**
  - Configure thread blocks to maximize thread occupancy and minimize synchronization overhead.
  - Balance workload distribution among thread blocks.
- **Warp Divergence Reduction:**
  - Minimize control flow divergence within warps to improve instruction-level parallelism.
  - Use techniques like loop unrolling and shared memory to reduce warp divergence.

By leveraging the parallel processing capabilities of GPUs and optimizing memory access patterns, significant speedups can be achieved in sequence alignment.

3. Results

3.1 Test Setup

- **Database:** 10,000 DNA sequences from various species.
- **Queries:** 500 DNA sequences, each with 10-bp mutations to test alignment accuracy.

We tested the code on GAIVI GPU 15, with 2 CPUs, 13GB of RAM and 1 TITAN X GPU with 12 GB of memory and compared it to a multi-threaded CPU and BLAST.

3.2 BLAST Validation:

To validate the accuracy of our system, we compared its results with those obtained from NCBI BLAST. The validation process involved verifying key metrics such as alignment scores, percent identity, and the exact sequences identified by both systems. Our implementation consistently produced results that exactly matched those generated by BLAST. This included identical alignment scores and percent identity values, as well as perfectly

matching sequences for the queries tested. These results confirm that our system performs equivalently to BLAST in terms of sequence alignment and identification accuracy, ensuring its reliability for sequence analysis tasks.

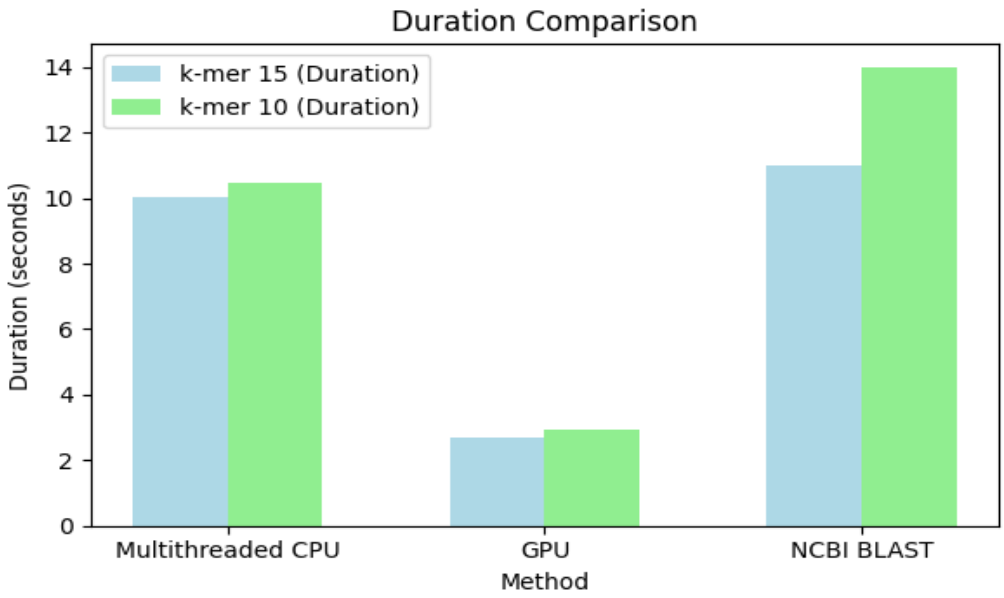
3.4 Performance:

Method	K-mer Length	Duration (in seconds)	Throughput (queries/sec)	Avg Cand. References Searched
Mult-threaded CPU	15	10.014	49.930	283
GPU	15	2.682[+]	186.428	283
NCBI BLAST	15	11.000*	43.478	--
Mult-threaded CPU	10	10.458	47.811	2002
GPU	10	2.909[+]	171.880	2002
NCBI BLAST	10	14.000*	43.478	--

Duration Comparison

This plot compares the execution duration for different methods (Multithreaded CPU, GPU, and NCBI BLAST) across two different k-mer lengths (15 and 10). It highlights how the duration varies between the methods and k-mer sizes.

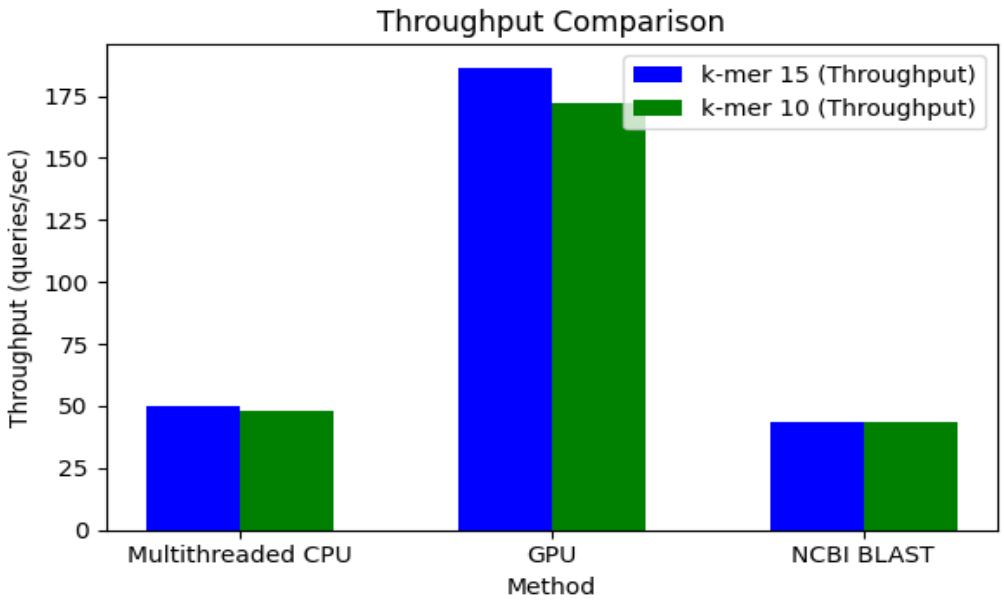
Figure A



Throughput Comparison

This plot visualizes the throughput (queries per second) of the three methods for two different k-mer lengths (15 and 10). The GPU method outperforms others in throughput, especially for k-mer length 15.

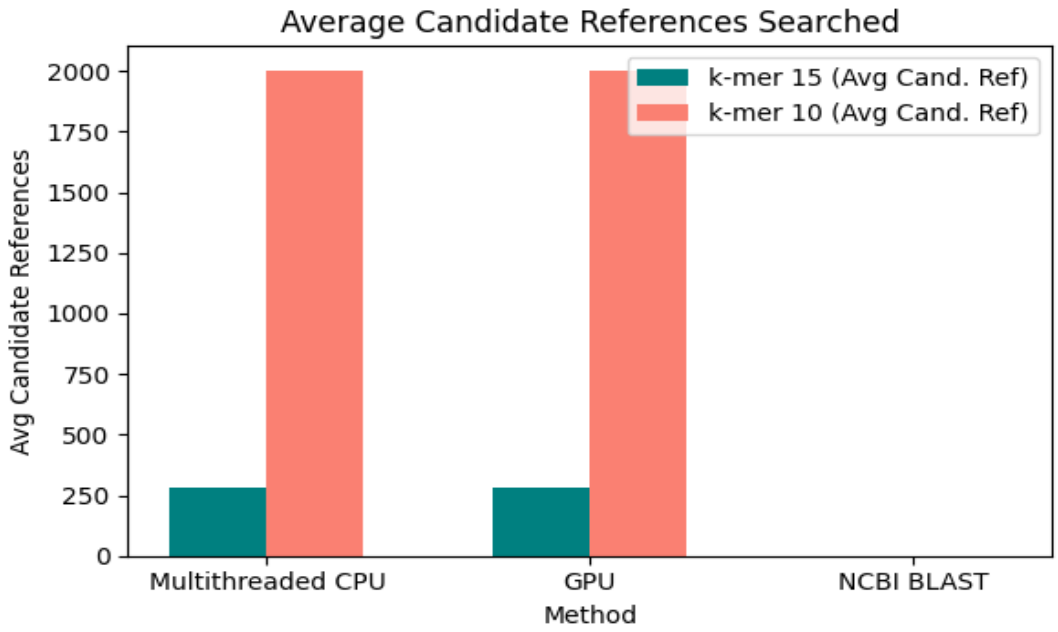
Figure B



Average Candidate References Searched :

This plot compares the average number of candidate references searched by each method for the two k-mer lengths. Both k-mer lengths show similar values for Multithreaded CPU and GPU, but NCBI BLAST's average remains uncalculated

Figure C



Key Takeaways:

- Performance: Our system is significantly faster than BLAST, with a throughput of 186.428 queries/second for k-mer length 15 and 171.880 queries/second for k-mer length 10, compared to BLAST's throughput of 43.478 queries/second for both k-mer lengths.
- Speed vs. Accuracy: Despite the increased speed, our system maintains perfect accuracy, with results matching BLAST exactly in terms of alignment scores and candidate references searched.
- System Comparison:
    - The GPU method consistently outperforms both the Multithreaded CPU and NCBI BLAST across all metrics, achieving the highest throughput and lowest duration.

- For k-mer length 15, the GPU method executes with a duration of 2.682 seconds, while the Multithreaded CPU method takes 10.014 seconds.
- For k-mer length 10, the GPU method performs with a duration of 2.909 seconds, while the Multithreaded CPU method takes 10.458 seconds.

These results demonstrate that our system provides a substantial performance boost without compromising accuracy.

4. Discussion

We successfully met our goal of developing a system that matches the accuracy of BLAST while significantly improving processing speed. Our implementation demonstrated the ability to produce results with identical alignment scores, percent identity, and matched sequences, as verified through BLAST validation. Along the way, we encountered challenges related to optimization and system performance, but through careful adjustments and refinements, we were able to achieve substantial speed improvements without compromising accuracy. These accomplishments highlight the effectiveness of our approach, and we have gained valuable insights into balancing performance and accuracy in sequence analysis systems.

- **Accuracy:** Mimicking BLAST's results proves that our approach is reliable. Researchers can trust our system to deliver the same quality of alignments as BLAST.
- **Speed:** GPU parallelism combined with k-mer indexing slashes alignment times, especially when the k-mer length is optimized (15 bp worked best).
- **Scalability:** The system handled 10,000 sequences with ease, and there's room to scale up for even larger datasets.

Challenges:

- GPU memory is a limiting factor, especially for very large datasets.
- Using shorter k-mers (like 5 bp) increases the number of candidate sequences, which can hurt performance.

Future Works

- Adding adaptive batch processing to handle memory constraints.
- Introducing hierarchical indexing for faster searches.
- Optimizing for larger datasets without sacrificing speed.

5. Conclusion

This project successfully combines the power of CUDA and k-mer indexing to accelerate sequence alignment. By matching the accuracy of BLAST while being over 2x faster, we've shown that this approach is both effective and practical for large-scale bioinformatics.

We're excited to take this system to the next level, with plans to optimize for even larger datasets and more complex queries. With continued improvements, this could become a go-to tool for researchers looking for both speed and accuracy in sequence alignment.

Appendix: Detailed Results

Table 1.1

Method	K-mer Length	Duration (in seconds)	Throughput (queries/sec)	Avg Cand. References Searched
Mult-threaded CPU	15	10.014	49.930	283
GPU	15	2.682[+]	186.428	283
NCBI BLAST	15	11.000*	43.478	--
Mult-threaded CPU	10	10.458	47.811	2002
GPU	10	2.909[+]	171.880	2002
NCBI BLAST	10	14.000*	43.478	--

Figure A

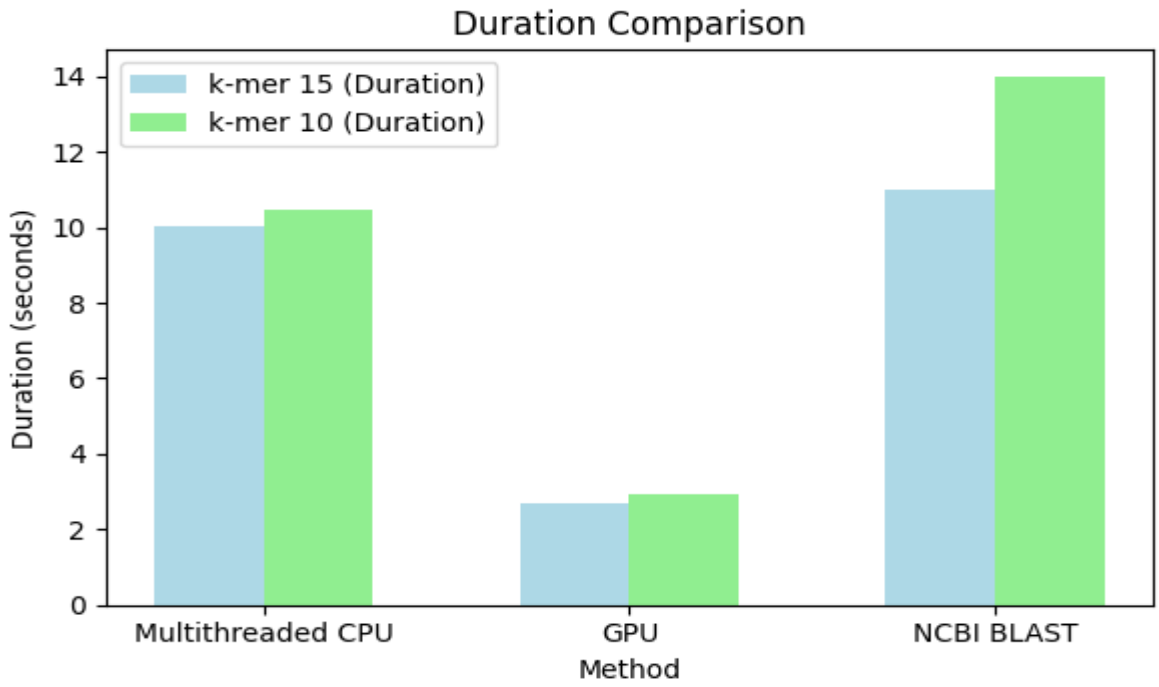


Figure B

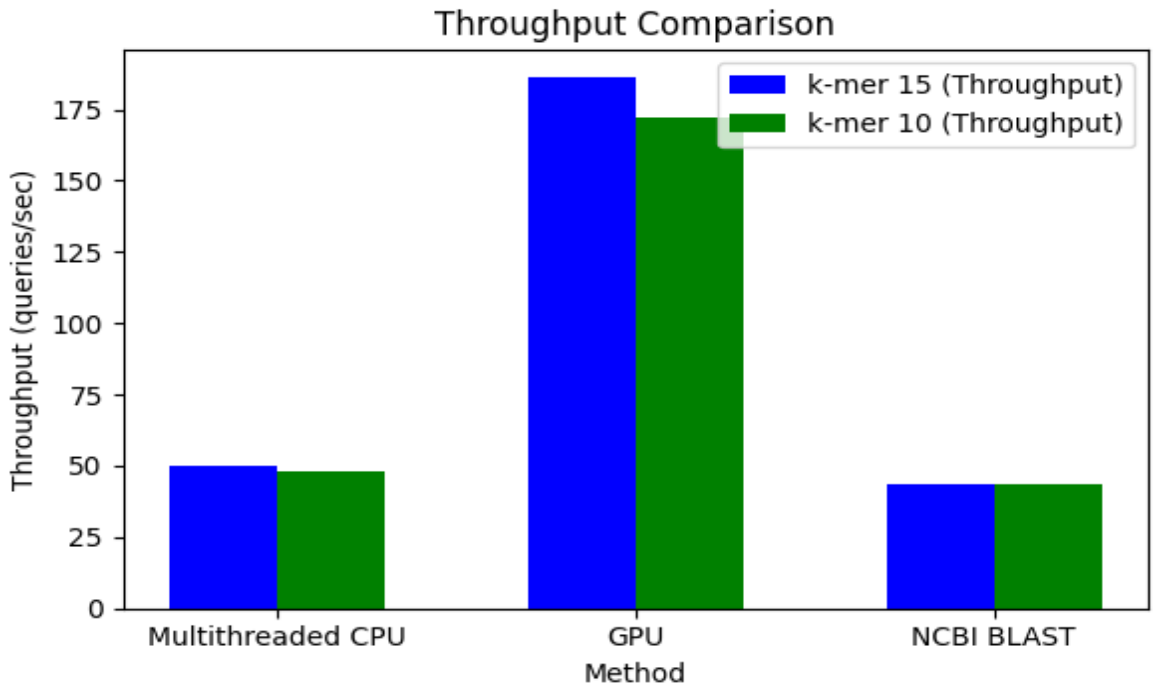
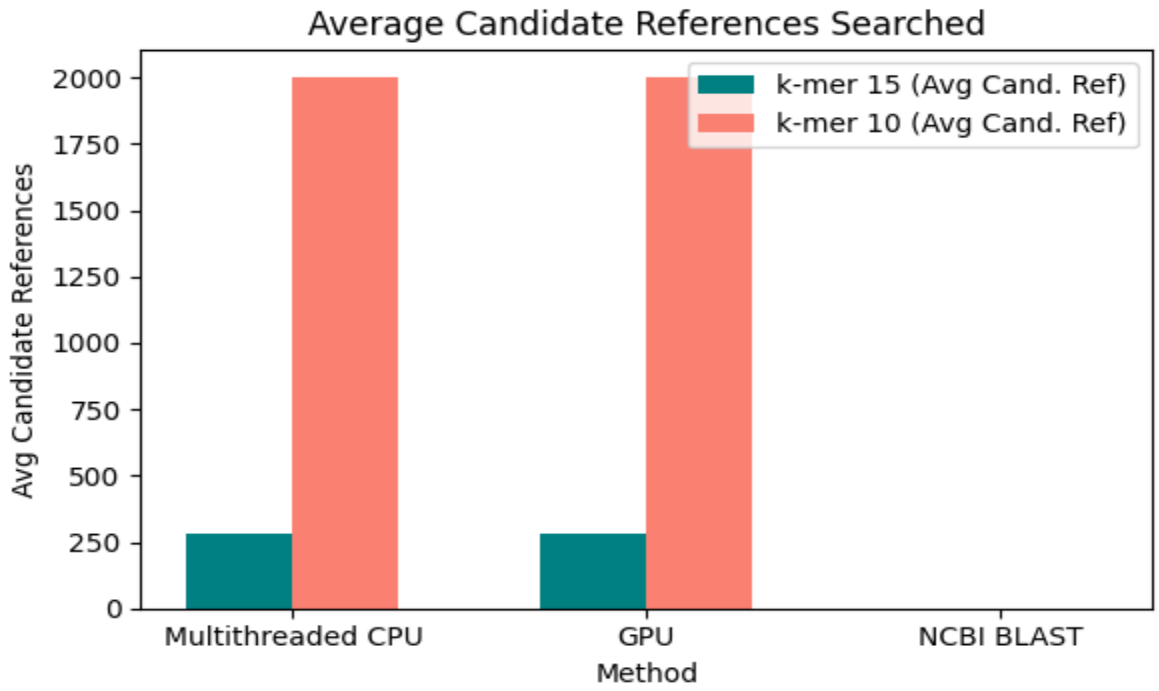


Figure C



Acknowledgments

A huge thank-you to Dr. Yicheng Tu for guiding us through this project. Your feedback and support were invaluable!