Proposal - Parallelization Strategies for DNA Alignment: A Comparative Study of Multiple String-Matching Algorithms

Kuan-Yi Lee (B10901091), Wei-Chi Wu (B12902080), Yu-Chen Hsu (B10901158)

November 16, 2025

Abstract

DNA sequence alignment is very important in bioinformatics, used to detect patterns in genetic sequences for applications such as mutation analysis and genomic screening. In this proposal, we aim to investigate and compare multiple DNA sequence alignment algorithms from a parallel programming perspective. Our goal is to identify the different parallelization strategies each algorithm enables, analyze their parallelizability on HPC , and evaluate how algorithmic structure affects performance, scalability, and hardware suitability. By comparing brute-force, KMP, Boyer–Moore, Rabin–Karp, we seek to characterize the strengths and limitations of each algorithm under parallel execution.

1 Motivation

1.1 Importance in Biomedical Informatics

DNA sequence alignment is very important in biomedical informatics. Many essential tasks all depend on it. (Ex. comparing different viral strains, finding cancer-related mutations, checking conserved protein regions, and running large-scale sequence). Currently, there are many DNA alignment algorithms implementations. This naturally raises the question of how these algorithms differ when parallelized. Because their computational structures vary, we expect their parallel behavior and suitable optimization strategies to differ as well. Motivated by this, we aim to study multiple alignment algorithms, analyze their parallelization characteristics, and determine effective strategies for each by comparing with official implementations.

1.2 Importance in Computer Engineering

From a computer engineering perspective, DNA pattern matching is a strong platform for studying parallel programming because different string-matching algorithms expose very different forms of parallelism. Algorithms such as brute force, KMP, Boyer–Moore, and Rabin–Karp vary in control flow, data dependencies, and memory behavior, making them ideal for comparing parallelization strategies.

The workload is easily scalable by adjusting sequence or pattern sizes, and the diversity of algorithmic structures highlights hardware affinities. For example, GPUs favor data-parallel methods like brute force or Rabin–Karp, while CPUs may better support branch-heavy algorithms such as KMP or Boyer.

Overall, DNA sequence matching provides a compact yet rich testbed for evaluating how algorithm design affects parallel performance across different architectures.

2 Target Problem Definition and Implement

Our project strictly follows the problem definition from a DNA sequence alignment study (*Accepted in EduHPC-24: Workshop on Education for High-Performance Computing*). For the brute-force baseline, we directly adopt the official implementation provided in the EduHPC'24 code release. For all other algorithms, such as KMP, Boyer–Moore, and Rabin–Karp, we rely on their original reference implementations from standard libraries or official sources (see References). This ensures that all algorithmic behaviors remain correct and faithful to their canonical definitions.

2.1 Alignment Algorithms Considered

Each algorithm exhibits different structural properties that affect parallelism:

- **Brute-force**: highly data-parallel, independent comparisons across patterns and positions; serves as our baseline implementation.
- **Knuth–Morris–Pratt (KMP)**: efficient skipping mechanism but strong sequential dependency during both prefix-table construction and scanning.
- **Boyer–Moore (BM)**: uses heuristic-based jumps; irregular control flow limits fine-grained parallelism.
- **Rabin–Karp (RK)**: hash-based matching exposes coarse-grained and position-level parallelism.

Each algorithm is evaluated under identical input sequences to ensure fair comparison.

2.2 Correctness Evaluation

To maintain consistency with the EduHPC'24 baseline, we validate correctness by applying each algorithm to the same DNA sequences and verifying that the detected match positions and alignment results are identical to their respective reference implementations. When parallelized, each method must preserve the exact output semantics.

3 Proposed Parallelization Approach

We parallelize the program in stages, focusing first on the most effective sources of parallelism.

3.1 Parallelization Strategy

The algorithm naturally supports parallelism because patterns are independent. We mainly parallelize over patterns, assigning different patterns to different threads. For long sequences, we may also explore finer-grained parallelism over positions. Shared data such as coverage counts will be handled using thread-local buffers and reductions to avoid contention.

3.2 Data Structures and Memory Layout

The main sequence and patterns remain stored in simple contiguous arrays for good memory locality. Patterns may be reordered (e.g., by length or type) to improve cache behavior. Coverage updates are accumulated in thread-local arrays and merged at the end for correctness and performance.

3.3 CUDA or OpenMP / MPI

Depending on course requirements, we plan to implement the main version using OpenMP / MPI and CUDA to evaluate scalability on different architectures.

4 Implementation Plan

4.1 Schedule

Our schedule is showed in Table 1.

4.2 Evaluation Methodology

We evaluate our parallel implementation in terms of both **correctness** and **performance**.

For **correctness**, we will run multiple input configurations, and verify that the number of matched patterns, the coverage array, and all final checksums exactly match the official sequential baseline.

For **performance**, we will measure kernel execution time, speedup, and scalability across different thread counts and problem sizes. All results will be directly compared against sequential version.

We will test several representative scenarios by varying sequence length, the number of patterns, and the distributions of pattern lengths and starting positions.

Week	Milestone	Description
10-11 (End)	Survey and understand concept	Survey topic and related algorithms for DNA alignment.
12 (Now)	Sequential code	Understand sequential code in various papers, and try to refine to meet our requirement.
13	First parallel version (CPU)	Implement pattern-level parallelism using OpenMP (or MPI) for 4 algorithms; ensure correctness via checksum comparison; add basic timing instrumentation.
14	Optimization and tuning	Experiment with different schedules, pattern orderings, and strategies for updating coverage; collect performance data for multiple scenarios and thread counts.
15	Extended models	Implement GPU versions of the kernel for algorithms; compare performance against the CPU-only version and official implement.
16	Final evaluation and report	Analysis parallel methods for each algorithm, and prepare the final report and presentation.

Table 1: Tentative project schedule and milestones.

5 Expected Outcomes

By the end of the project, we expect to **identify and evaluate the most effective parallelization strategies for several DNA alignment algorithms**. For each method, we will analyze its parallel behavior, determine suitable optimization techniques, and document the algorithm-specific challenges encountered during implementation. The final results will provide a comparative view of how different algorithms respond to parallel execution and which strategies yield the best performance on modern hardware.

References

1. Linus Zwaka, Parallel DNA Sequence Alignment on High-Performance Systems with CUDA and MPI, Arxiv, 2024.

2. Y. T. L. Yu, "Genetic Algorithms and Evolutionary Computation," Course Materials, Department of Electrical Engineering, National Taiwan University, 2024.

3. DNA Sequence Aligment Assigment Design, EduHPC-24: Workshop on Education for High-Performance Computing, 2024.

4. Xiangyu Lu. The Analysis of KMP Algorithm and its Optimization, Journal of Physics Conference Series, 2019.