# Approximate Longest Common Substring of Multiple Strings: Experimental Evaluation

Hamed Hasibi [1] Neerja Mhaskar [1] William F. Smyth [1]

[1]McMaster University

The Prague Stringology Conference 2025
Prague, Czech Republic, August 25–26

# Overview

1. Preliminaries
2. Problem Definition
3. $\mathcal{O}(N^2/p)$ time by CPU
4. Further speedup by GPU
5. Results
6. Future work

# Outline

# Definitions

- $s[1..]$ and $s[..n]$ are ***prefix*** and ***suffix***, respectively.
- For equal-length strings $s_1$ and $s_2$, **Hamming distance** $d_H(s_1, s_2)$ is the number of positions $i$ such that $s_1[i] \neq s_2[i]$, $1 \leq i \leq |s_1|$.
- For $1 \leq i' \leq |s_1|$ and $1 \leq j' \leq |s_2|$, we define $\mathbf{LCP^{H,k}_{(s_1,s_2)}[i',j']} = l$ as the length of the longest common prefix between the suffixes $s_1[i'..|s_1|]$ and $s_2[j'..|s_2|]$, such that $d_h(s_1[i'..i'+l-1], s_2[j'..j'+l-1]) \leq k$.
- $\boldsymbol{MaxLCP^{H,k}_{(s_i,s_j)}}$ is defined as an array of length $|s_i|$, where each entry $MaxLCP^{H,k}_{(s_i,s_j)}[i']$ stores the maximum value of $LCP^{H,k}_{(s_i,s_j)}[i',j']$ over all $1 \leq j' \leq |s_j|$.

# Example

Table: $LCP^{H,1}_{(s_1,s_2)}$ and $MaxLCP^{H,1}_{(s_1,s_2)}$ for $s_1 =$ ACGTA (rows) and $s_2 =$ ACGACA (columns).

| | A | C | G | A | C | A |
|---|---|---|---|---|---|---|
| A | 4 | 1 | 1 | 3 | 1 | 1 |
| C | 1 | 3 | 1 | 1 | 2 | 1 |
| G | 1 | 1 | 2 | 1 | 1 | 1 |
| T | 1 | 1 | 2 | 1 | 2 | 1 |
| A | 1 | 1 | 1 | 1 | 1 | 1 |

| $MaxLCP^{H,1}_{(s_1,s_2)}$ |
|---|
| 4 |
| 3 |
| 2 |
| 2 |
| 1 |

# Outline

# *Rkt*-LCS problem

Given integers $k, t, m \in \mathbb{N}$ with $1 \leq t \leq m$ and a set $\boldsymbol{S} = \{s_1, s_2, \ldots, s_m\}$ of strings,

## Problem

*[Restricted k-t Longest Common Substring (Rkt-LCS) [2]] Find a longest substring u taken from any string in* $\boldsymbol{S}$ *such that there exist t distinct strings* $s'_1, \ldots, s'_t \in \boldsymbol{S}$ *with corresponding substrings* $u_1, \ldots, u_t$ *satisfying* $d_\delta(u, u_j) \leq k$ *for every* $j = 1, \ldots, t$.

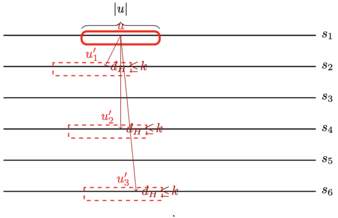


Figure: *Rkt*-LCS for $m = 6$, $t = 4$ , and $\delta = H$ (not necessarily substring of $s_1$)

# Arxiv Results

Parameters: $N = m\ell$, $k$, $t$

**Theorem**

*The k-t LCS problem is NP-hard for $\delta = H$ [2].*

**Theorem**

*The Rkt-LCS for $\boldsymbol{S} = \{s_1, s_2, \ldots, s_m\}$ for $\delta = H$ can be computed in $\mathcal{O}(N^2)$ time and $\mathcal{O}(m\ell^2)$ additional space [2].*

**Theorem**

*The Rkt-LCS problem for $\boldsymbol{S} = \{s_1, s_2, \ldots, s_m\}$ and $t = m$ can be computed in $\mathcal{O}(mN \log^k \ell)$ time with $\mathcal{O}(N)$ additional space, for any $\delta = \{H, L, E\}$ [2].*

# Arxiv Results

**Theorem**

*The Rkt-LCS for $\delta = \{L, E\}$ and $\boldsymbol{S} = \{s_1, s_2, \ldots, s_m\}$ can be computed in $\mathcal{O}(k\ell N^2)$ time [2].*

**Lemma**

*The Strong Exponential Time Hypothesis **(SETH)**: for every $\varepsilon > 0$, there exists an integer q such that SAT on q-CNF formulas with m clauses and n variables cannot be solved in $m^{O(1)}2^{(1-\varepsilon)n}$ time.*

**Theorem**

*Suppose there is a $\varepsilon > 0$ such that Rkt-LCS for any $t = m$ and $\delta = H$ can be solved in $\mathcal{O}(N^{2-\varepsilon})$ time on binary strings for $k = \Omega(\log \ell)$. Then SETH is false [2].*

# LengthStat Data structure

**Definition (LengthStat [2])**

Let $\boldsymbol{S} = \{s_1, s_2, \ldots, s_m\}$ be a set of strings. For every $(i, x)$ pair with $1 \leq i \leq m$ and $1 \leq x \leq |s_i|$, define the $LengthStat^{k}_{(i,x)}$ table as follows:

$$LengthStat^{H,k}_{(i,x)}[l, j] = \begin{cases} 1, & \text{if } MaxLCP^{H,k}_{(s_i,s_j)}[x] \geq l \\ 0, & \text{otherwise} \end{cases}$$

where $1 \leq j \leq m$ indexes the strings $\boldsymbol{S}$ and $1 \leq l \leq |s_i| - x + 1$ is the prefix length.

The matrix is augmented with a final column $LengthStat^{H,k}_{(i,x)}[\,l, m+1\,]$ storing, for each row $l$, the sum of its first $m$ entries, i.e. the number of strings in $\boldsymbol{S}$ that share with $s_i[x..]$ a prefix of length at least $l$ under $k$-mismatch Hamming distance.

# Example

Table: The $lengthStat^{H,1}_{(1,3)}$ table for $\boldsymbol{S} = \{TTGAC, CGAAAT, TGGTA\}$, where $k = 1$. The $lengthStat^{H,1}_{(1,3)}[3, 2] = 1$ indicates the 1-approximate occurrence of the length-3 prefix of $s_1[3..5]$ ($GAC$), somewhere in $s_2$ ($s_2[2..4] = GAA$).

| | 1 ($s_1$) | 2 ($s_2$) | 3 ($s_3$) | 4 (Frequency) |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 3 |
| 2 | 1 | 1 | 1 | 3 |
| 3 | 1 | 1 | 0 | 2 |

# LS key-values and $C_i$

We formulate **last** column of $LengthStat^{H,k}$ in LS key-value (i,p,l),count:

- $i$: the string index of the string $s_i \in \boldsymbol{S}$
- $p$: the starting position of $s_i$
- $l$: the prefix length of the $p$-$th$ suffix of $s_i$
- $count$: the number of the strings in which $s_i[i..i+l-1]$ has k-approximate occurrences.

For instance, the entry ((1,2,4),5) in LS states that the substring $s_1[2..2+4-1]$ occurs with at most $k$ mismatches in five strings of the set $\boldsymbol{S}$.

---

$C_i$, $1 \leq i \leq m$: **Longest** substring of $s_i$ that has $k$-approximate occurrences in $t$ strings of the set $\boldsymbol{S}$.

# Outline

# CPU Computation Model

Suppose we have 2 processors ($P_1$ and $P_2$):

- $P_1$ **sequentially** computes $MaxLCP^{H,k}_{(s_i,s_j)}$, $ls(i,p,l)$ and $C_i$ for $i = \{1,2,3\}$ (first for $i = 1$, then $i = 2$, and finally $i = 3$).
- $P_2$ **sequentially** computes $MaxLCP^{H,k}_{(s_i,s_j)}$, $ls(i,p,l)$ and $C_i$ for $i = \{4,5,6\}$ (first for $i = 4$, then $i = 5$, and finally $i = 6$).

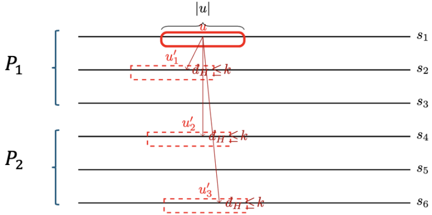


Figure: String set distribution across processors

# Time Complexity & Runtime

$N = m\ell$: $m$ is the number of strings in set $\boldsymbol{S}$, $\ell$ is the length of each string
$P$: number of processors

- Sequential: $\mathcal{O}(N^2)$ [2]
- Parallel: $\mathcal{O}(N^2/P)$

(a) Parallel CPU, $t = 1000$, $\tau = 15$

| Cores | $k = 1$ | | $k = 3$ | | $k = 10$ | |
|---|---|---|---|---|---|---|
| | Time | RSU | Time | RSU | Time | RSU |
| 4 | 138 | 1.00× | 242 | 1.00× | 705 | 1.00× |
| 8 | 71 | 1.94× | 122 | 1.98× | 352 | 2.00× |
| 16 | 40 | 3.45× | 63 | 3.84× | 181 | 3.89× |
| 32 | 19 | 7.26× | 42 | 5.76× | 112 | 6.29× |

Figure: Runtime for $m = 5000$

# Outline

# Introduction to GPU Computing

- **GPU (Graphics Processing Unit)** originally designed for graphics rendering.
- Now widely used for **general-purpose parallel computing**.
- Consists of thousands of lightweight cores optimized for parallel tasks.
- Excellent for data-parallel problems (e.g., matrix multiplication, deep learning).

# CPU vs GPU

**CPU**

- Few powerful cores.
- Optimized for sequential processing.
- Large caches, complex control logic.
- Suited for diverse, branching workloads.

**GPU**

- Thousands of simple cores.
- Optimized for massive parallelism.
- High memory bandwidth.
- Suited for uniform, data-parallel workloads.

# Why Big-O is Not Enough on GPU

- $\mathcal{O}$ captures **asymptotic growth**, but ignores hardware-level factors.
- On GPUs, performance depends on:
  - **Parallelism**: how well the problem maps to thousands of threads.
  - **Warp divergence**: different branches reduce efficiency.
  - **PCIe transfer costs**: moving data CPU $\leftrightarrow$ GPU.
- Two algorithms with the same $\mathcal{O}(N^2)$ complexity may run **orders of magnitude apart** on a GPU.
- Hence, GPU complexity is better described by **work, depth, and parallelism efficiency**, not just $\mathcal{O}$.

# GPU Computation Model

GPU computes $MaxLCP^{H,k}_{(s_i,s_j)}$ of given $i$ and all $j$ in $\mathcal{O}(m\ell)$ kernel call.

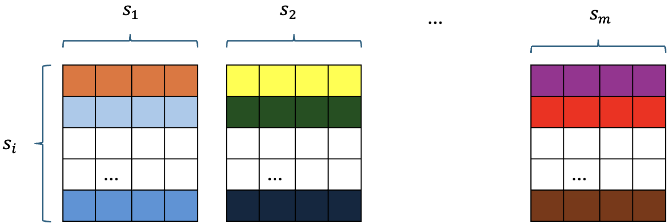


Figure: $MaxLCP^{H,k}_{(s_i,s_j)}$ cells with similar colors are computed by one GPU kernel call.

# GPU Computation Model

$$
\begin{cases}
P_1 \text{ invokes } \mathcal{O}(m\ell) \text{ threads for } MaxLCP^{H,k}_{(s_1, S_{buffer})} \text{ at } t^1_1 \\
P_1 \text{ invokes } \mathcal{O}(m\ell) \text{ threads for } MaxLCP^{H,k}_{(s_2, S_{buffer})} \text{ at } t^1_2 \\
\quad \vdots \\
P_1 \text{ invokes } \mathcal{O}(m\ell) \text{ threads for } MaxLCP^{H,k}_{(s_{m/p}, S_{buffer})} \text{ at } t^1_{m/p}
\end{cases}
$$

$$\vdots$$

$$
\begin{cases}
P_p \text{ invokes } \mathcal{O}(m\ell) \text{ threads for } MaxLCP^{H,k}_{(s_{m-m/p+1}, S_{buffer})} \text{ at } t^p_1 \\
P_p \text{ invokes } \mathcal{O}(m\ell) \text{ threads for } MaxLCP^{H,k}_{(s_{m-m/p+2}, S_{buffer})} \text{ at } t^p_2 \\
\quad \vdots \\
P_p \text{ invokes } \mathcal{O}(m\ell) \text{ threads for } MaxLCP^{H,k}_{(s_m, S_{buffer})} \text{ at } t^p_{m/p}
\end{cases}
$$

# Outline

# System Configuration

- 2x 4.1 GHz 16-core Intel Xeon Gold 6426Y processors
- 250 GB of main memory
- 4x NVIDIA H100 GPUs, each with 80 GB of memory
- REHL 9 operating system
- Dataset consists of two files, each containing 1,077,820 nucleotide sequences ($\Sigma = \{A, T, C, G\}$) of uniform length 51, formatted in `FASTQ`.

# Results

Implementation for *Rkt*-LCS under $\delta = H$:

- GPU implementation: 179$\times$ speed-up

Table 3: Runtime (in seconds) and Relative SpeedUp (RSU — relative to $Cores = 4$) comparison for $m = 5000$

(a) Parallel CPU, $t = 1000$, $\tau = 15$

| Cores | $k=1$ | | $k=3$ | | $k=10$ | |
|---|---|---|---|---|---|---|
| | Time | RSU | Time | RSU | Time | RSU |
| 4 | 138 | 1.00× | 242 | 1.00× | 705 | 1.00× |
| 8 | 71 | 1.94× | 122 | 1.98× | 352 | 2.00× |
| 16 | 40 | 3.45× | 63 | 3.84× | 181 | 3.89× |
| 32 | 19 | 7.26× | 42 | 5.76× | 112 | 6.29× |

(b) Parallel CPU, $t = 100$, $\tau = 30$

| Cores | $k=1$ | | $k=3$ | | $k=10$ | |
|---|---|---|---|---|---|---|
| | Time | RSU | Time | RSU | Time | RSU |
| 4 | 82 | 1.00× | 146 | 1.00× | 358 | 1.00× |
| 8 | 44 | 1.86× | 75 | 1.94× | 182 | 1.96× |
| 16 | 30 | 2.73× | 39 | 3.74× | 94 | 3.80× |
| 32 | 17 | 4.94× | 26 | 5.61× | 66 | 5.42× |

(c) Parallel GPU, $t = 1000$, $\tau = 15$

| Cores | $k=1$ | | $k=3$ | | $k=10$ | |
|---|---|---|---|---|---|---|
| | Time | RSU | Time | RSU | Time | RSU |
| 4 | 4 | 1.00× | 3 | 1.00× | 72 | 1.00× |
| 8 | 5 | 0.80× | 4 | 0.75× | 37 | 1.94× |
| 16 | 6 | 0.66× | 6 | 0.50× | 22 | 3.27× |
| 32 | 12 | 0.33× | 11 | 0.27× | 21 | 3.42× |

(d) Parallel GPU, $t = 100$, $\tau = 30$

| Cores | $k=1$ | | $k=3$ | | $k=10$ | |
|---|---|---|---|---|---|---|
| | Time | RSU | Time | RSU | Time | RSU |
| 4 | 2 | 1.00× | 2 | 1.00× | 2 | 1.00× |
| 8 | 3 | 0.66× | 3 | 0.66× | 2 | 1.00× |
| 16 | 4 | 0.50× | 5 | 0.40× | 5 | 0.40× |
| 32 | 9 | 0.22× | 9 | 0.22× | 9 | 0.22× |

# Results

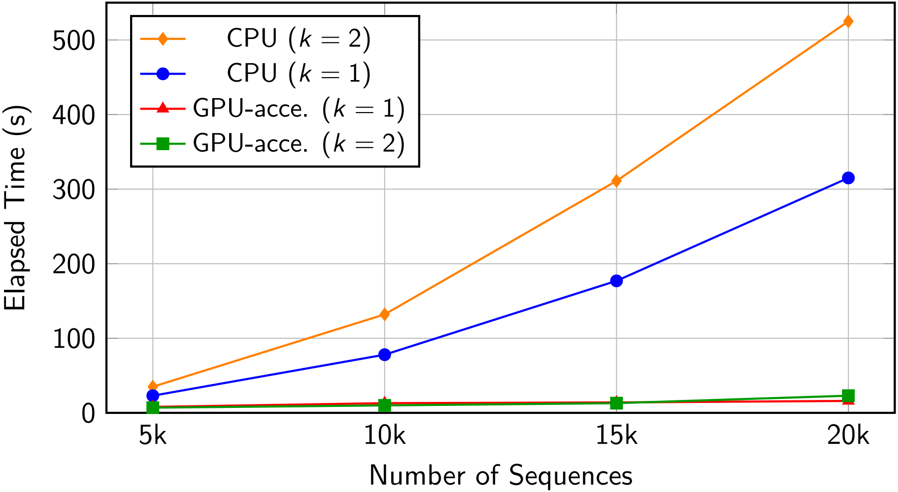


Figure: Runtime comparison for CPU and GPU-accelerated implementations with varying $k$ on different sequence set sizes, $t = 1000$, $\tau = 15$, and $p = 32$

# Results

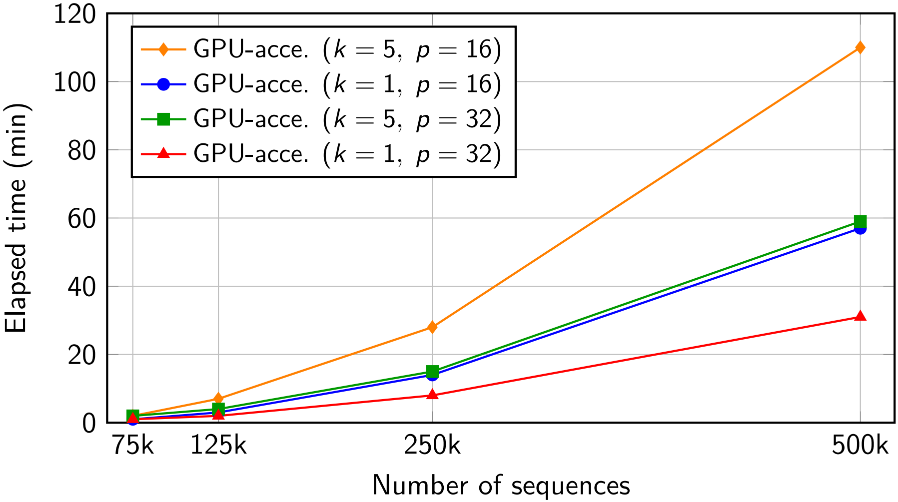


Figure: GPU-accelerated implementation runtime (in whole minutes) for different $(k, p)$ settings, $t = 1000$, and $\tau = 15$

# Outline

# Future work

- Implementation for other distance metrics (like affine gap edit distance)
- Further speed up using FFT (Fast Fourier Transform)
- Adapting Flouri et el. [1] $LCP^{H,k}$ computation to GPU

# Thank You!

# References I

[1] T. Flouri, E. Giaquinta, K. Kobert, and E. Ukkonen. Longest common substrings with k mismatches. Inf. Process. Lett., 115(6-8):643–647, 2015.

[2] H. Hasibi, N. Mhaskar, and W. F. Smyth. On the complexity of finding approximate LCS of multiple strings, 2025. https://arxiv.org/abs/2505.15992.