Эффективные параллельные алгоритмы сравнения строк

Никита Мишин
Научный руководитель: к. ф.-м. н., доцент Березун Д. А.
Консультант: DPhil, доцент, Тискин А. В.
Рецензент: инженер-исследователь, Корнилова А. В.

Санкт-Петербургский государственный университет

07.06.2022

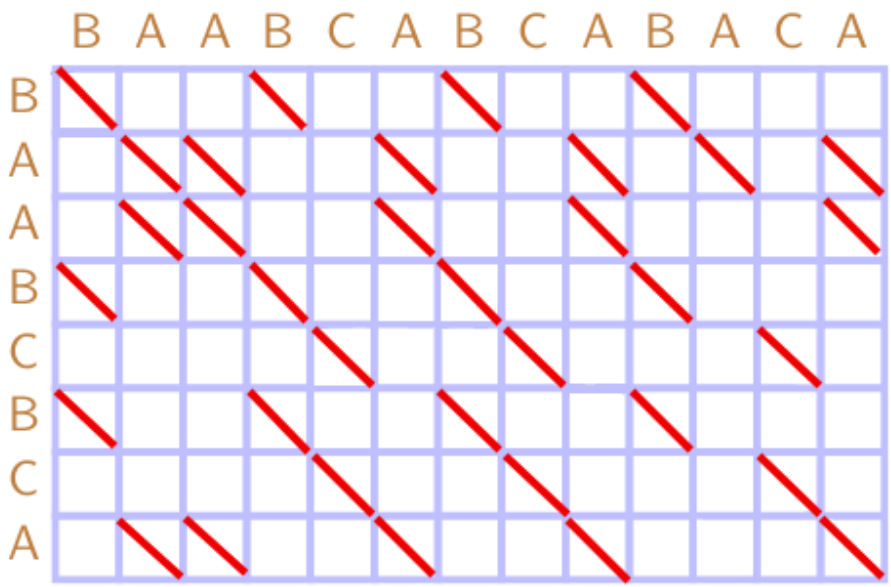
Поиск наибольшей общей подпоследовательности

Задача поиска наибольшей общей подпоследовательности (LCS)

- $a = a_1 a_i ... a_m$, $b = b_1 b_2 ... b_n$
- $LCS(a, b) = |\textit{наибольшая общая подпоследовательность}|$
- Пример:
  - $a = BAABCBCA$, $b = BAABCABCABACA \rightarrow$
    $LCS(a, b) = LCS(BAABCBCA, BAABCABCABACA) = 8$

- $O(m \times n)$
- Применение в биоинформатике, текстовый анализ, ...
- Оценка глобальной похожести

Поиск наибольшей общей подпоследовательности



LCS-сетка

Полулокальный поиск наибольшей общей подпоследовательности

Полулокальная задача поиска наибольшей общей подпоследовательности (semi-local LCS)

- Обобщение над LCS, больше информации для сравнения строк:
  - (string-substring) a против всех подстрок b (и наоборот)
  - (prefix-suffix) все префиксы a против всех подстрок b (и наоборот)
  - Решение в явном виде:

$$H_{a,b} = \begin{bmatrix} \textit{suffix-prefix} & \textit{substring-string} \\ \textit{string-substring} & \textit{prefix-suffix} \end{bmatrix}$$

Полулокальный поиск наибольшей общей подпоследовательности

- Пример подматрицы *string-substring* для $a = BAABCBCA, b = BAABCABCABACA$:

0	1	2	3	4	5	6	6	7	8	8	8	8	8
-1	0	1	2	3	4	5	5	6	7	7	7	7	7
-2	-1	0	1	2	3	4	4	5	6	6	6	6	7
-3	-2	-1	0	1	2	3	3	4	5	5	6	6	7
-4	-3	-2	-1	0	1	2	2	3	4	4	**5**	5	6
-5	-4	-3	-2	-1	0	1	2	3	4	4	5	5	6
-6	-5	-4	-3	-2	-1	0	1	2	3	3	4	4	5
-7	-6	-5	-4	-3	-2	-1	0	1	2	2	3	3	4
-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	3	4
-9	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4
-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	0	1
-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	0

Мотивация

- Немногие существующие решения направлены на применимость в производных задачах
- В теории все хорошо: $O(m \times n)$, а на практике?
  - Скрытые константы
  - Всем нужны эффективные (быстрые) алгоритмы
- Структура алгоритмов позволяет применить паралеллизацию на разных уровнях
  - Динамическое программироание
  - Разделяй и властвуй
  - Сужение задачи и возможность использования битовой паралеллизации
- Многие задачи могут быть сведены к semi-local LCS:
  - Bounded length smith-waterman
  - Dynamic time warping
  - Window substring
  - Periodic LCS
  - Compressed string comparison
  - ...

Цель и задачи

Цель

Реализация библиотеки параллельных алгоритмов решения semi-local LCS

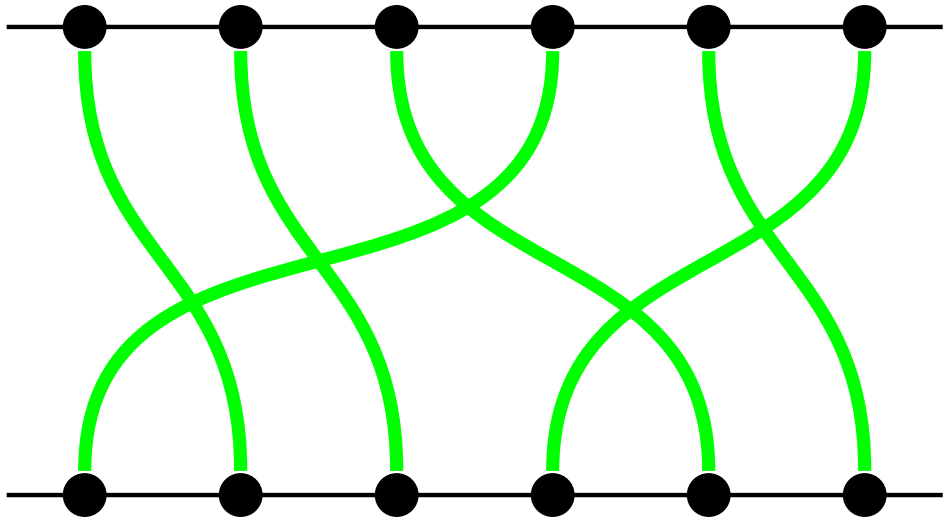
Задачи:

- Изучить теорию в основе semi-local LCS, известные алгоритмы для semi-local LCS и их узкие места
- Реализовать последовательные и параллельные версии алгоритмов с устранением их слабых сторон
- Разработать и реализовать новый гибридный алгоритм для решения задачи semi-local LCS
- Разработать и реализовать новый битовый подход для решения задачи LCS с ограниченным алфавитом на основе задачи semi-local LCS
- Провести сравнение алгоритмов из библиотеки на реальных и синтетических данных

Липкая коса (моноид Гекке)

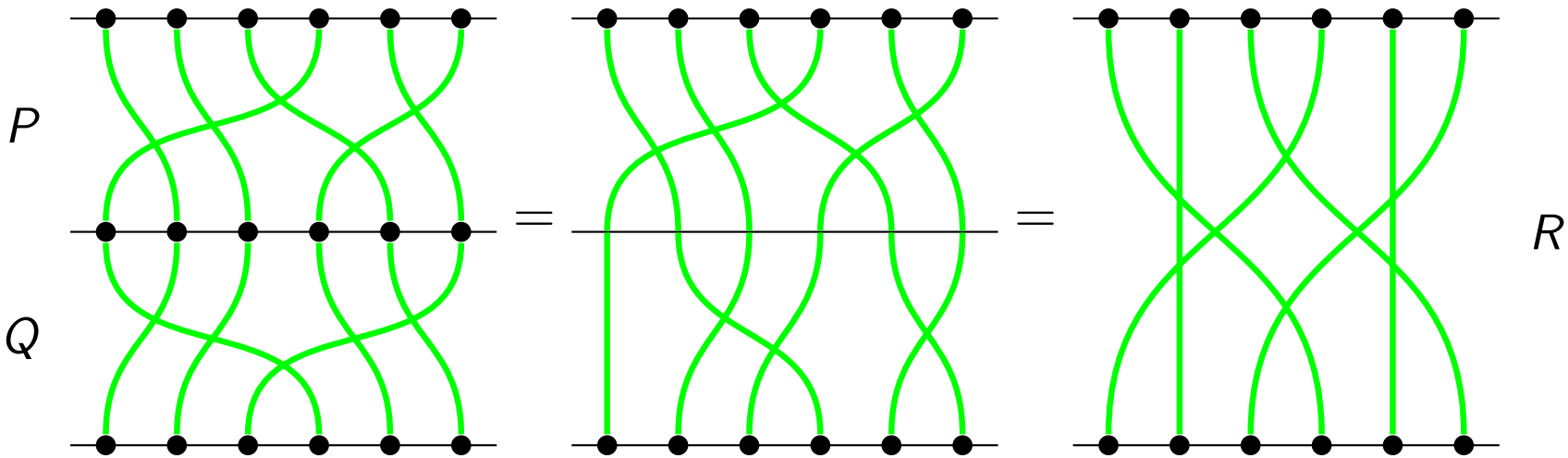
Геометрическая интерпретация:

- $m + n$ нитей
- Нити могут пересекаться друг с другом
- Два состояния косы: *сокращенная* и *несокращенная*
- *Сокращенная* коса представима в виде перестановочной матрицы
  - $\{0 \mapsto 1, 1 \mapsto 2, 2 \mapsto 4, 3 \mapsto 0, 4 \mapsto 5, 5 \mapsto 3\}$:



Липкая коса (моноид Гекке)

- Склейка липких кос $O((m+n)\log(m+n))$:
  - Расположить косу под косой, распутать нити, где необходимо для приведения к сокращенной форме

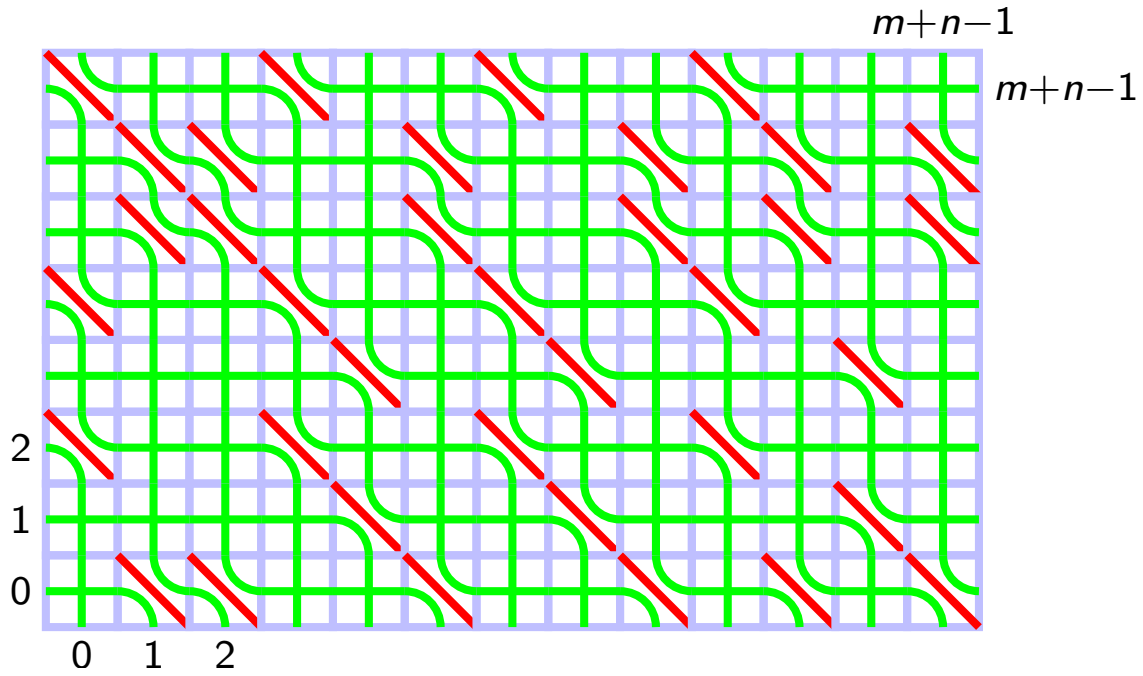


Связь semi-local LCS и липких кос

Сильная связь задачи с липкой косой размера $m + n$:

- Решение в терминах поиска сокращенной липкой косы:
  - ★ в LCS сетку встраивается липкая коса (45 градусов)
  - ★ $a_i = b_j \rightarrow$ внутри клетки нити не могут пересечься
  - ★ номера нитей от 0 до $m + n - 1$
  - ★ Поиск решения в неявном виде (перестаночная матрица,не нужно хранить $O((m + n)^2)$)
- Два существующих подхода:
  - ★ Разделяй-и-властвуй: разбить большую косу на маленькие, распутать, потом склеить.
  - ★ Динамическое программирование: пройти по всей сетке, распутать там, где необходимо.

Связь semi-local LCS и липких кос



Встраивание косы, коса несокращенная

Алгоритм с динамическим программированием

- $(\leftarrow, \uparrow)$-зависимость
- *if* (*ранее пересекались или символы равны*) *пересечь нити*

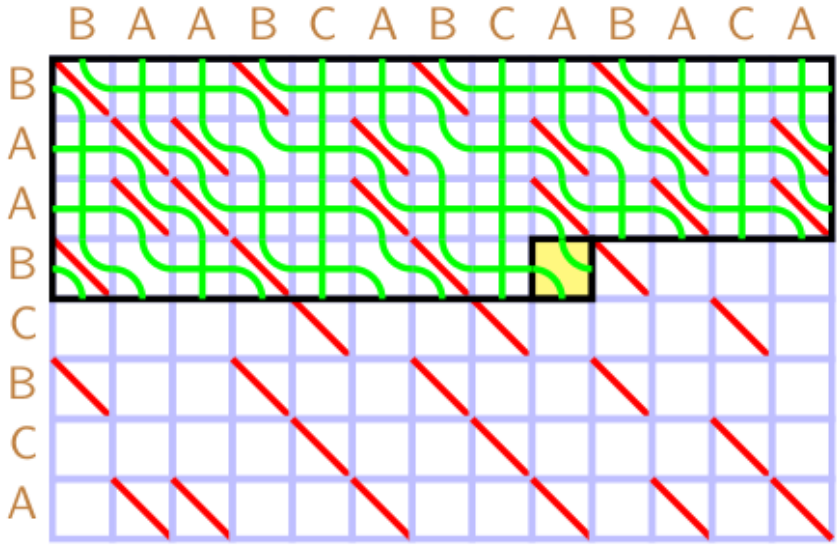


Figure: Итеративный процесс распутывания нитей построчно (row-major)

Иллюстрация взята из книги А. Тискина *Semi-local string comparison: Algorithmic techniques and applications*

Алгоритм с динамическим программированием

- $(\leftarrow, \uparrow)$-*зависимость* $\rightarrow$ антидиагональный(antidiagonal) шаблон обхода для параллелизации на уровне потоков $\rightarrow$
- Несбалансированные вычисления при обходе антидиагонально $\rightarrow$

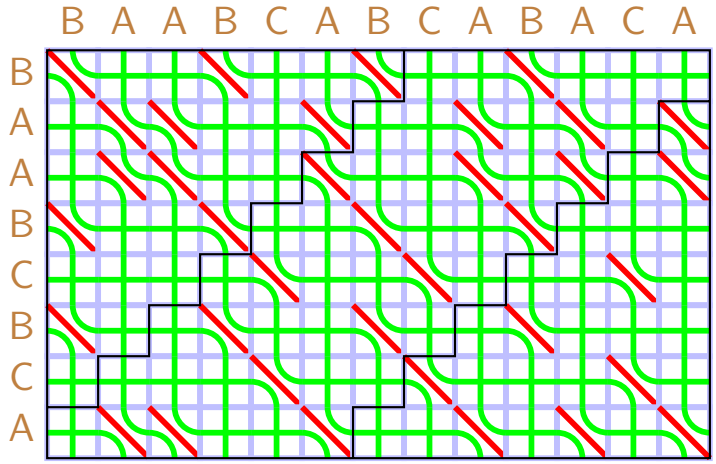


Figure: Реорганизация вичислений, в конце склейка треугольников

- SIMD-параллелизм:
  - Хранение a в обратном порядке для поддержания последовательного доступа
  - Устранение *if* $=$ арифметика $+$ постоянная запись в память
  - $m + n \leq 2^{16} \rightarrow$ 16 битные машинные слова

Алгоритм с разделяй-и-властвуй

- Рекурсивно бьем на мелкие подзадачи
- Конкатенация мелких нитей в более крупную с помощью склейки кос

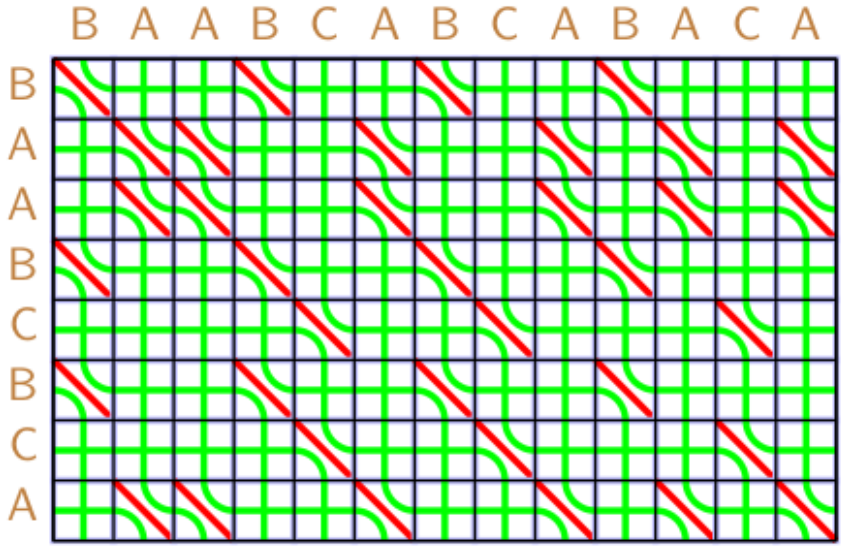


Figure: База рекурсии, косы с 2 нитями

Иллюстрация взята из книги А. Тискина *Semi-local string comparison: Algorithmic techniques and applications*

Алгоритм с разделяй-и-властвуй

- Конкретно для умножения кос:
  - Перестановочных матриц конечное число $\rightarrow$ предподсчет произведения кос $\rightarrow$ $:(w_0, w_1, , ..w_i.., w_{N-1}), w_i \in [0, N-1]$
  - Аккуратное управление памятью $\rightarrow$:
    - ★ Переиспользование памяти с верхних уровней рекурсии
    - ★ Единоразовое выделение памяти
- В обоих случаях рекурсия $\rightarrow$ паралеллизм на уровне задач

Гибридный алгоритм

- Исходная сетка разбивается на подсетки: $m_{i,j} + n_{i,j} \leq 2^k$, $i \in [0, I-1], j \in [0, J-1]$
- В каждой подсетке нити распутываются через итеративный алгоритм
- Далее $\log I + \log J$ паралелльных склеек (склеиваем вертикально/горизонтально):
  - Склейка по длинной строне, чтобы размеры подсеток были примерно одинаковые
- Нивелируем проблему с двойной рекурсией, k-битные машинные слова для базовых подзадач (более гранулярный SIMD-паралеллизм)

LCS для строк с ограниченным алфавитом

- Битовый алгоритм Hyrro и Crorchemore:
  - 4-5 операций
  - carry propagation
  - precalc
- Новый подход на основе липких кос $\downarrow$

LCS для строк с ограниченным алфавитом

- Общая идея:
  - Номера горизонтальных и вертикальных нитей либо 1 либо 0
  - Разная кодировка строк и нитей (big и little endian)
  - Обработка антидиагональным паттерном
  - Операторы булевы логики + сдвиги (11 битовых операций)
  - Также придумано и реализовано обобщение на алфавит размера 2^N

Сравнение алгоритмов[1]

- `AMD Ryzen-7-3800X`, 8 ядер and 16 потоков, `C++`, `G++10.2.0`, $w = 32$, *OpenMP*
- Синтетический датасет, числовые последовательности, эмулирование разных сценариев
  - $\sigma = 1$ — высокая частота
  - $\sigma = 5$ — средняя частота
  - $\sigma = 26$ — слабая частота
- Реальные данные: геномы вирусов из NCBI базы данных

[1] Детальное описание может быть найдено в статье Efficient Parallel Algorithms for String Comparison'ICPP21, часть результатов опущена

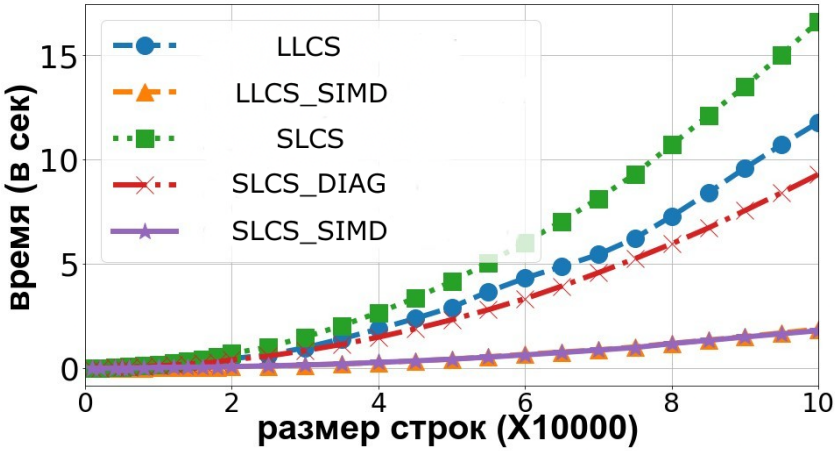
Сравнение алгоритмов[1]

Легенда:

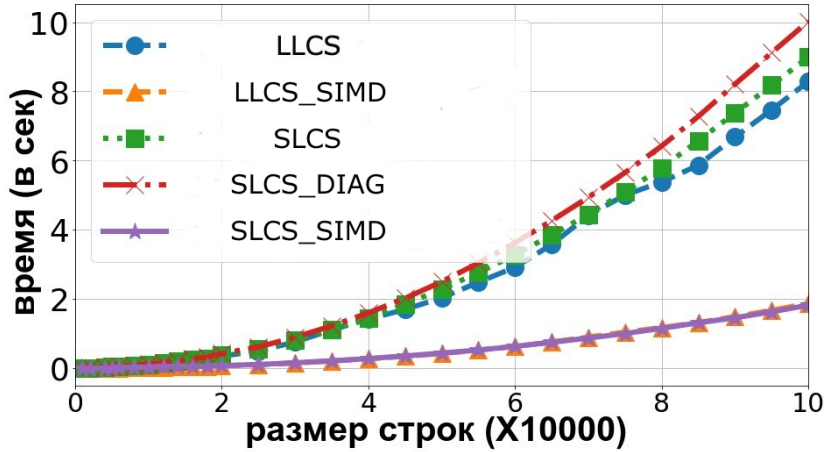
- *LLCS* – классический алгоритм решения LCS с линейным потреблением памяти
- *LLCS_SIMD* — *LLCS* с антидиагональным паттерном
- *SLCS* — итеративный алгоритм для решения semi-local LCS
- *SLCS_DIAG* — *SLCS* с антидиагональным паттерном
- *SLCS_SIMD* — *SLCS_DIAG* с оптимизациями
- *hybrid* — гибридный алгоритм
- *bitwise* — битовый алгоритм для бинарных строк

[1] Детальное описание может быть найдено в статье Efficient Parallel Algorithms for String Comparison'ICPP21, часть результатов опущена

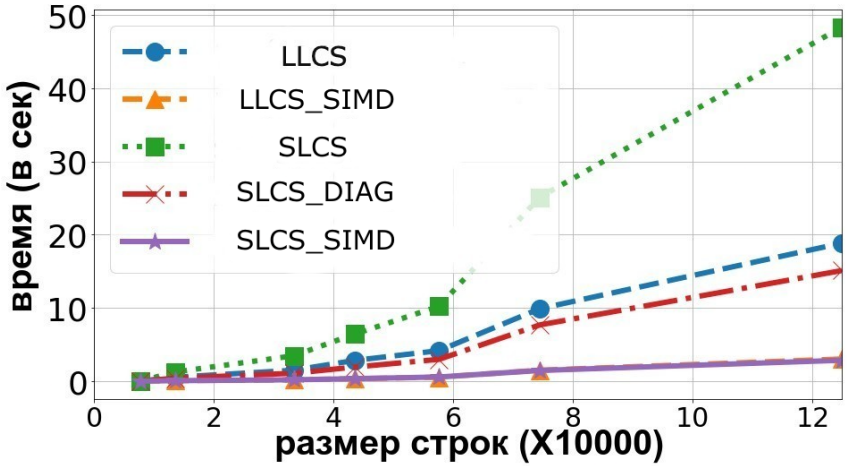
Сравнение алгоритмов[1]



Строки одинаковой длины $\sigma = 1$



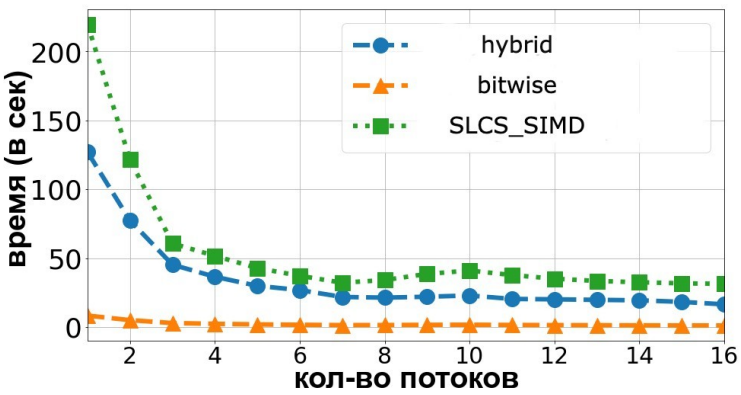
Строки одинаковой длины $\sigma = 26$

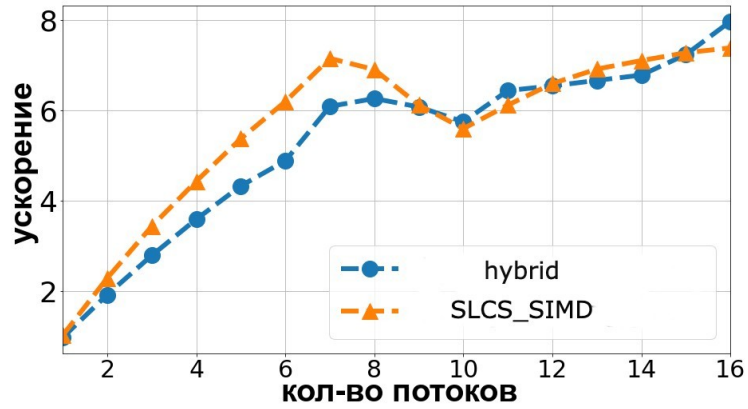


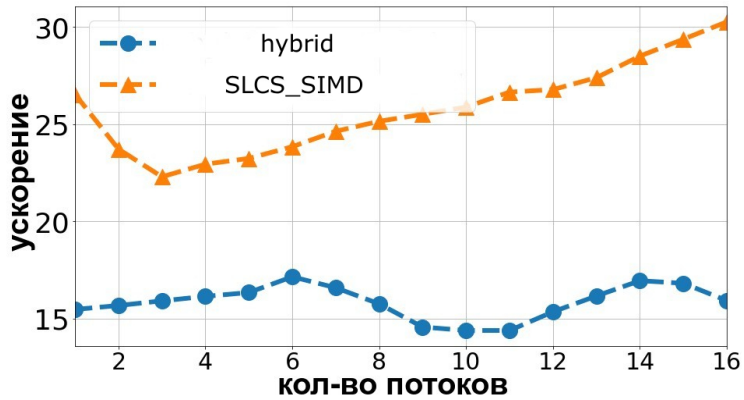
Геномы вирусов примерно одинаковой длины

[1] Детальное описание может быть найдено в статье Efficient Parallel Algorithms for String Comparison'ICPP21, часть результатов опущена

Сравнение алгоритмов[1]







Производительность бит-параллельного алгоритма относительно полулокальных алгоритмов на бинарных строках, $m = n = 10^6$

[1] Детальное описание может быть найдено в статье Efficient Parallel Algorithms for String Comparison'ICPP21, часть результатов опущена

Результаты

- Изучена предметная область вокруг semi-local LCS, выявлены слабые места существующих подходов
- Разработана библиотека паралелльных алгоритмов решения задачи semi-local LCS:
  - Библиотека написана на языке C++, исходный код доступен по ссылке: https://github.com/NikitaMishin/semilocal
  - Предложены эффективные последовательные и параллельные алгоритмы для semi-local LCS
  - Разработан новый гибридный алгоритм сочетающий преимущества предыдущих для semi-local LCS
  - Разработан новый битовый подход без сумматоров для вычисления LCS двух строк с ограниченным алфавитом
- Проведена апробация алгоритмов из библиотеки:
  - Алгоритмы при применении оптимизаций применимы на практике к большим данным
  - Большая часть результатов опубликована на конференции: Efficient Parallel Algorithms for String Comparison'ICPP21 Nikita Mishin, Daniil Berezun, Alexander Tiskin