Efficient Parallel Algorithms for String Comparison

Nikita Mishin, Daniil Berezun, Alexander Tiskin

JetBrains Research, Programming Languages and Tools Lab
Saint Petersburg State University, Russia

11.08.2021

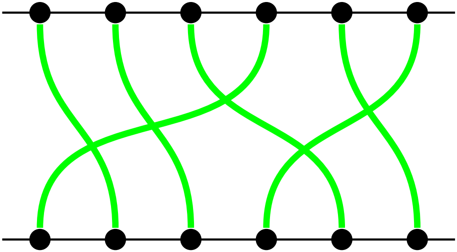
LCS

- $a = a_1 a_i ... a_m, \; b = b_1 b_2 ... b_n$
- $LCS(a, b) = |longest\ common\ subsequence|$
- $a = CIPR, \; b = ICPP \rightarrow LCS(a, b) = LCS(CIPR, ICPP) = 2$
- $a = BAABCBCA, \; b = BAABCABCABACA \rightarrow$
  $LCS(a, b) = LCS(BAABCBCA, BAABCABCABACA) = 8$
- $O(nm)$

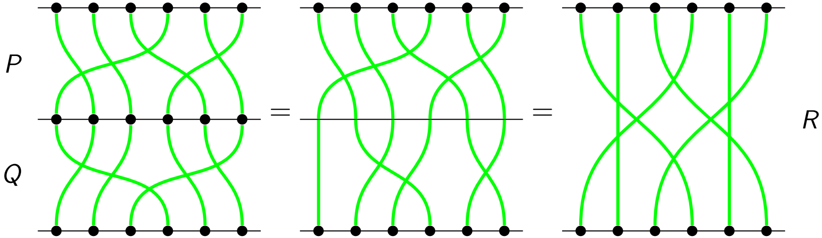
Preliminaries::sticky braid

Informal definition:

- $m + n$ monotone curves (called strands)
- Neighboring strands can form a crossing
- Neighboring strands can cross at most once
- $\{0 \mapsto 1, 1 \mapsto 2, 2 \mapsto 4, 3 \mapsto 0, 4 \mapsto 5, 5 \mapsto 3\}$



- Multiplication $O((m+n)\log(m+n))$ — place one braid under another and untangle strands



Introduction::semi-local LCS

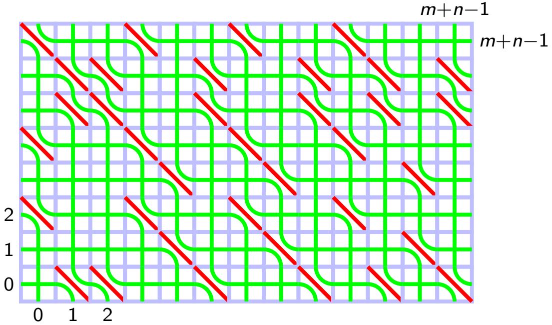
8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
7	7	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
6	6	7	7	7	7	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
5	6	7	7	7	7	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
4	5	6	6	6	6	7	7	7	8	8	8	8	8	8	8	8	8	8	8	8	8
3	4	5	5	5	6	7	7	7	8	8	8	8	8	8	8	8	8	8	8	8	8
2	3	4	4	4	5	6	6	7	8	8	8	8	8	8	8	8	8	8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

$a = BAABCBCA$
$b = BAABCABCABACA$
$H[i, j] = LCS(a, b[i : j])$
$H(4, 11) = 5$

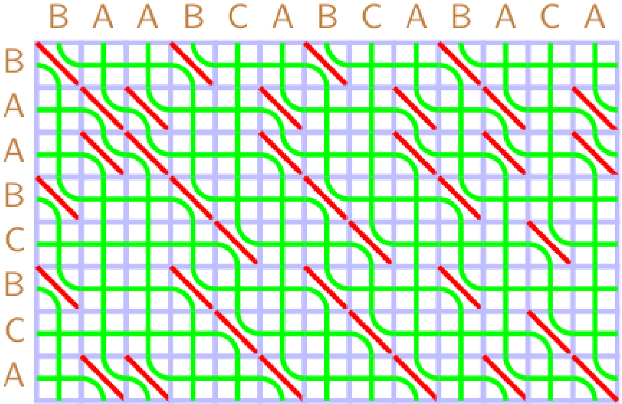
Introduction::semi-local LCS

- Can be expressed via sticky braids objects of size $n + m$
  - Embeddings into LCS grid
    - rotate braid by 45 degrees anti-clockwise
    - symbol matches – barrier for strands to intersect
  - Two approach:
    - Divide-and-conquer: split into smaller braids; to concatenate apply sticky braid multiplication
    - DP: process cell-by-cell and cross strands if needed
  - $O(nm)$

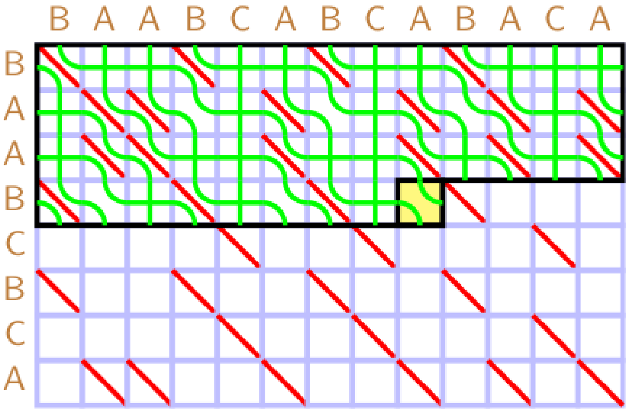
Introduction::semi-local LCS



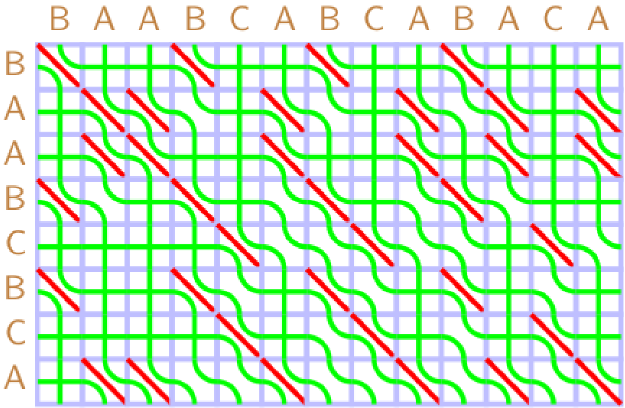
Implementation::DP



$$if(\texttt{a\_symb = b\_symb}) \mid\mid (\texttt{h\_strand > v\_strand}))$$
$$swap(h\_strand, v\_strand)$$



Implementation::DP



Implementation::DP

- $\leftarrow$ and $\uparrow$ — cell dependency

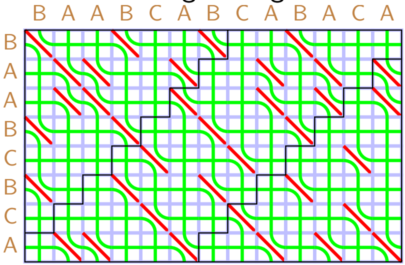
- $\mathit{if}(\texttt{a\_symb = b\_symb}) \mid\mid (\texttt{h\_strand > v\_strand}))$
  $\mathit{swap}(h\_strand, v\_strand)$ — inside cell computation

Implementation::DP

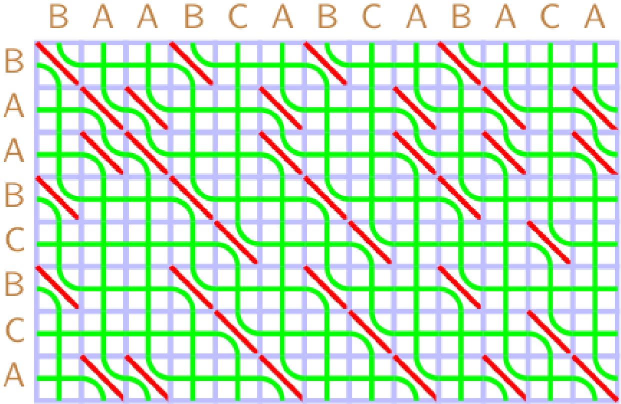
- Thread-level parallelization via antidiagonal pattern
- SIMD parallelization via branch elimination:

$$\texttt{h\_strand}' = (\texttt{h\_strand} \ \& \ (\texttt{p}-1)) \mid ((-\texttt{p}) \ \& \ \texttt{v\_strand})$$
$$\texttt{v\_strand}' = (\texttt{v\_strand} \ \& \ (\texttt{p}-1)) \mid ((-\texttt{p}) \ \& \ \texttt{h\_strand})$$

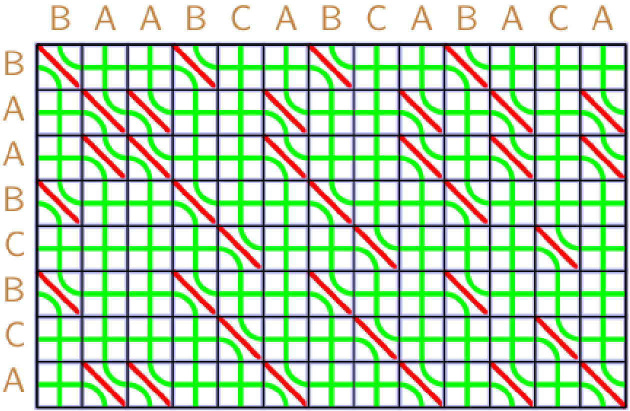
- Bonus №1: for $m + n < 2^t$ t bits per strand sufficient
- Bonus №2: possible load balancing through braid multiplication:



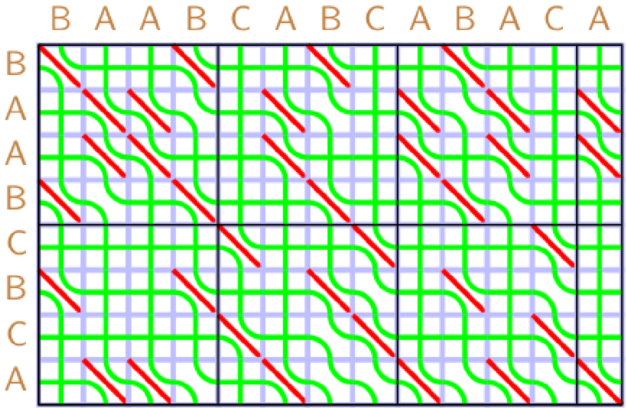
Implementation::Recursive



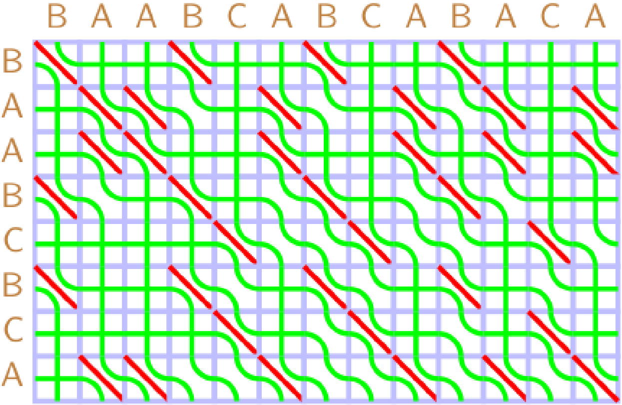
Implementation::Recursive



Implementation::Recursive



Implementation::Recursive



Implementation::Recursive

- core unit — steady ant algorithm:
  - Fast matrix multiplication $O(n \log n)$ (also divide-and-conquer)

- Deep recursion

Implementation::Recursive

- Processor-level parallelism
- Efficient memory management:
  - No malloc inside function
  - Reuse of space from outer levels
- Precalc product of permutations up to some N:
  - Small permutations fit to one machine word
  - $N! * N!$ pairs for N
  - Lookup to map $pair(p, q)$

Implementation::Combine two approaches

- Eliminate outer recursion:
  - Split into fixed-size subproblem: $m_i + n_i < 2^{16}$
  - One thread per problem
  - Then apply sticky braid multiplication in parallel fashion

Implementation::Bit-parallel prefix LCS

- Bit-parallel prefix LCS for binary strings:
  - Hyyrö, Crochemore et al.
  - Integer addition
  - Therefore, carry propagation

Implementation::Bit-parallel prefix LCS for binary strings

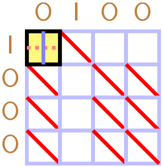
Idea

- 1 for horizontal strands, 0 for vertical
- Most significant bit first for *a* and horizontal strands
- Least significant bit first for *b* and vertical strands
- Shifts for word alignment, Boolean operators for cell logic
- Process in antidiagonal tiles
- $LCS(a, b) = |a| -$ set bits in h

Example

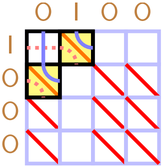
- $w = 4$
- $a = 1000,\ b = 0100$
- Encoding:
  - $a' = 1000_2,\ b' = 0010_2$
  - $h = 1111_2,\ v = 0000_2$

Implementation::Bit-parallel prefix LCS for binary strings



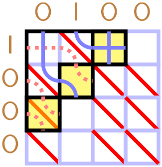
$h = 1111$

$v = 0000$



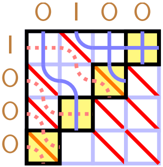
$h = 0011$

$v = 1100$



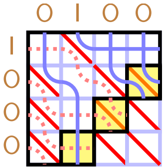
$h = 0111$

$v = 1000$



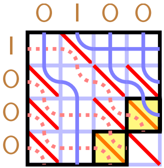
$h = 0011$

$v = 1010$



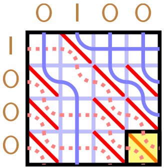
$h = 0011$

$v = 1010$



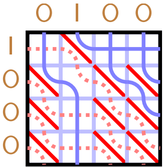
$h = 0001$

$v = 1011$



$h = 0001$

$v = 1011$



$h = 0001$

$v = 1011$

Implementation::Bit-parallel prefix LCS for binary strings

Processing of second antidiagonal (18 op):

- Compare characters: $s = !((a' \gg 2) \oplus b)$
- Active bits: $mask = 0011_2$ (compile time)
- Combing condition: $c = mask \; \& \; (s \mid (!(h \gg 2) \; \& \; v))$
- save $v' = v$
- update $v = (!c \; \& \; v) \mid (c \; \& \; (h \gg 2))$
- update $c = c \ll 2$
- update $h = (!c \; \& \; h) \mid (c \; \& \; (v' \ll 2))$

Implementation::Bit-parallel prefix LCS for binary strings

Processing of second antidiagonal (18 op):

- Compare characters: $s =!((1000_2 \gg 2) \oplus 0010_2) = 0011_2$
- Active bits: $mask = 0011_2$ (compile time)
- Condition: $c = 0011_2 \,\&\, (0011_2 \mid (!(1111_2 \gg 2) \,\&\, 0000_2)) = 0011_2$
- $v' = 0000_2$
- $v = (!0011_2 \,\&\, 0000_2) \mid (0011_2 \,\&\, (1111_2 \gg 2)) = 0011_2$ (1100)
- $c = 0011_2 \ll 2 = 1100_2$
- $h = (!1100_2 \,\&\, 1111_2) \mid (1100_2 \,\&\, (0000_2 \ll 2)) = 0011_2$ (0011)

Implementation::Bit-parallel prefix LCS for binary strings

- Optimizations
  - Register usage
  - Update Rule optimization:
  - $!a$

Implementation::Bit-parallel prefix LCS for binary strings

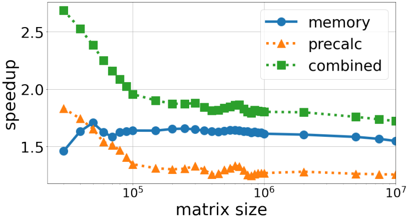
Processing of second antidiagonal (11 op):

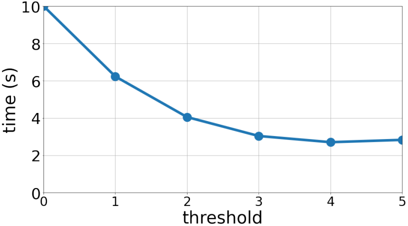
- Compare string characters: $s = ((a'' \gg 2) \ \oplus \ b)$
- $v' = v$
- $v = ((h \gg 2) \mid !mask) \ \& \ (v \mid (s \ \& \ mask))$
- $h = h \oplus (v \ll 2) \oplus (v' \ll 2)$.

Evaluation::Main results

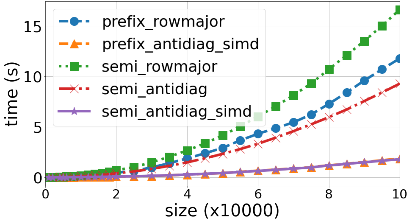
- `AMD Ryzen-7-3800X`, 8 cores and 16 threads, `C++`, `G++ 10.2.0`
- Synthetic dataset for different matching frequency:
  - $\sigma = 1$ — High
  - $\sigma = 5$ — Medium
  - $\sigma = 26$ — Low
- Real-data: Genome of viruses from NCBI

Evaluation::Main results

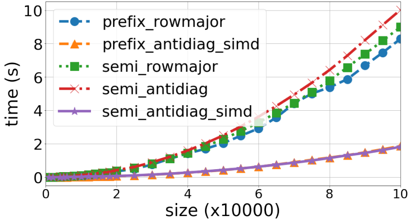




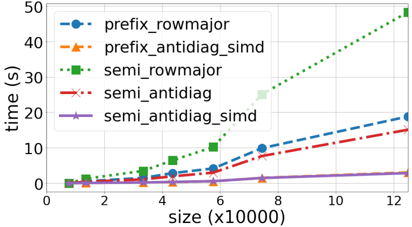
Evaluation::Main results



strings of equal lengths $\sigma = 1$

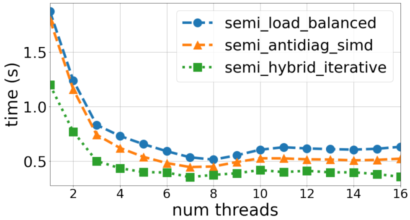


strings of equal lengths $\sigma = 26$

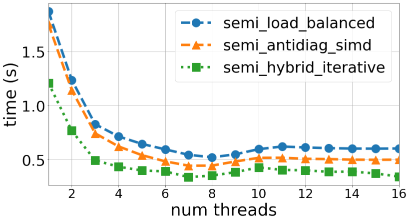


Viruses of appx. equal lengths

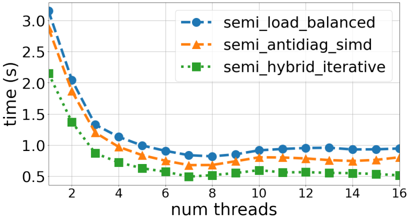
Evaluation::Main results



$\sigma = 1, \ m = n = 100000$

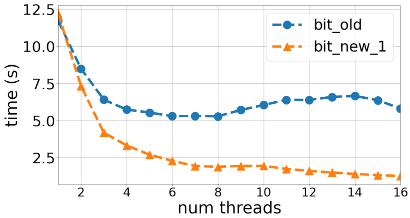


$\sigma = 26, \ m = n = 100000$

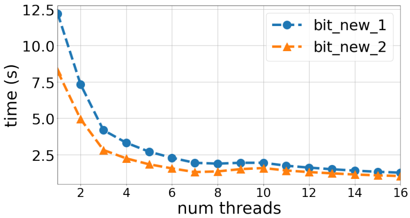


Viruses, $m = 124884, \ n = 134226$

Evaluation::Main results

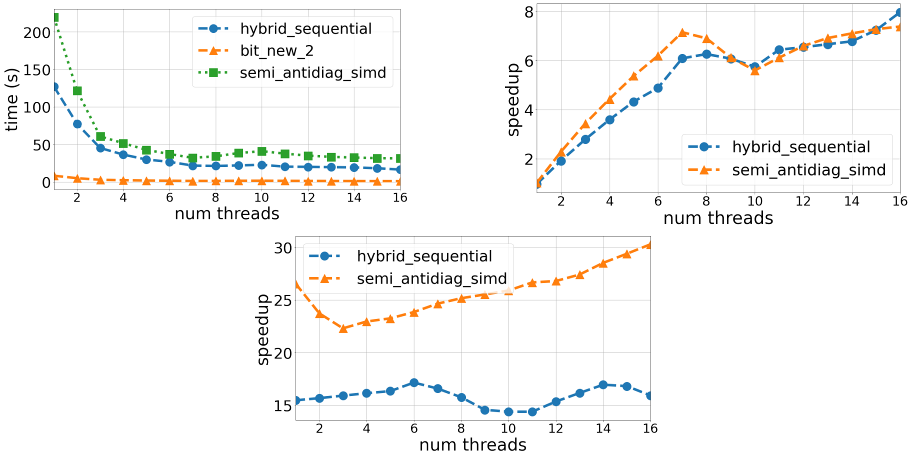


Memory access optimization



Boolean formula optimization

Evaluation::Main results



Relative performance of bit-parallel algorithm against semi-local LCS

Conclusion::Recap

- Semi-local LCS (theory works in practice!)
- Hybrid approach for semi-local LCS
- Bit-parallel prefix LCS without adders based on sticky braid

Conclusion::Takeaway

Semi-local LCS is cool
Let's study it!