A new algorithm for the alignment of long genomic sequences

Internship supervised by Dominique Lavenier, GenScale team

Igor Martayan

September 20, 2021

Introduction

DNA sequencing is more and more accessible

```
ATCGGGCAAT  TAAAAGGATC
TGAAGCGAAG  ACACCGTACC
AGACGTAGCG  AGCCCTATTT
ATCGGAGCAA  TAAAAGGATC
CGAAGCGAGA  GACCACCGTA
CAGGACGTAG  GGAGCCCTAT
```

Introduction

DNA sequencing is more and more accessible
but still presents many errors

```
ATCGGGCAAT  TAAAAGGATC
TGAAGCGAAG  ACACCGTACC
AGACGTAGCG  AGCCCTATTT
ATCGGAGCAA  TAAAAGGATC
CGAAGCGAGA  GACCACCGTA
CAGGACGTAG  GGAGCCCTAT
```

How do we correct these errors?

Correcting errors

3 types of errors

- insertion: `GCA -> GCTA`
- deletion: `GCA -> GA`
- substitution: `GCA -> GTA`

Correcting errors

3 types of errors

- insertion: `GCA -> GCTA`
- deletion: `GCA -> GA`
- substitution: `GCA -> GTA`

several reads from the same sequence

```
TTGAC_TCAAGGGCCA_TCATG
T _ACAT_A _GGCTAATTATG
T GACATTACGGGCCAGTAATG
TCGACA_CA GGGTAAA_AATT
----------------------
T GACATCA GGGCCAATAATG
```

Aligning two sequences

Computing an alignement score

- $+\alpha$ by match
- $-\beta$ by substitution
- $-\delta$ by insertion/deletion

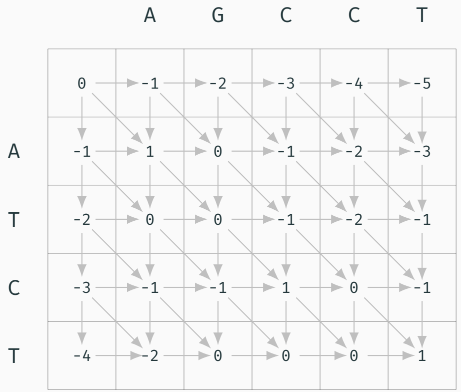
Aligning two sequences

Computing an alignement score

- $+\alpha$ by match
- $-\beta$ by substitution
- $-\delta$ by insertion/deletion

Needleman—Wunsch algorithm

Compute score matrix with dynamic programming, $\mathcal{O}(nm)$ complexity



$$\begin{array}{c|cccccc}
 & & A & G & C & C & T \\
\hline
 & 0 & -1 & -2 & -3 & -4 & -5 \\
A & -1 & 1 & 0 & -1 & -2 & -3 \\
T & -2 & 0 & 0 & -1 & -2 & -1 \\
C & -3 & -1 & -1 & 1 & 0 & -1 \\
T & -4 & -2 & 0 & 0 & 0 & 1
\end{array}$$

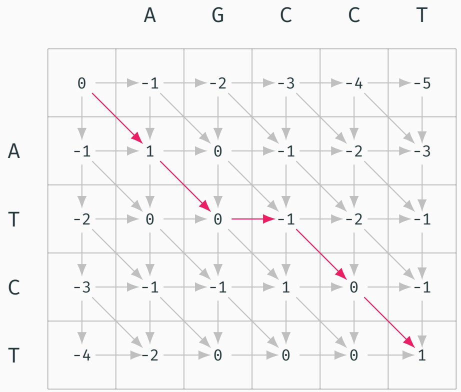
Aligning two sequences

Computing an alignement score

- $+\alpha$ by match
- $-\beta$ by substitution
- $-\delta$ by insertion/deletion

Needleman—Wunsch algorithm

Compute score matrix with dynamic programming, $\mathcal{O}(nm)$ complexity



```
AGCCT
|  ||
AT_CT
```

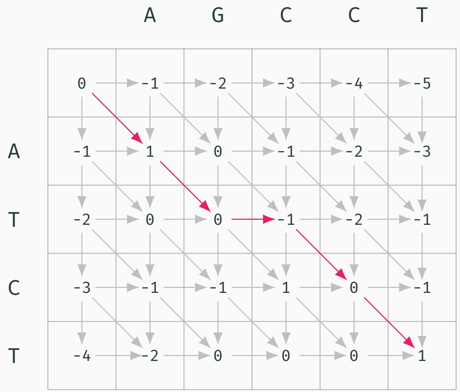
Aligning two sequences

Computing an alignement score

- $+\alpha$ by match
- $-\beta$ by substitution
- $-\delta$ by insertion/deletion

Needleman—Wunsch algorithm

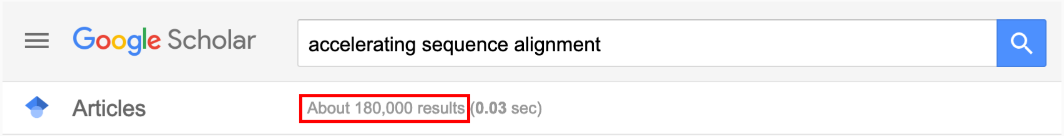
Compute score matrix with dynamic programming, $\mathcal{O}(nm)$ **complexity**



```
AGCCT
|  ||
AT_CT
```

How can we go faster?

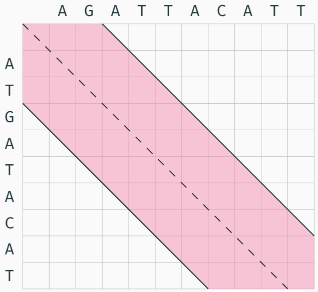
Accelerating sequence alignment


accelerating sequence alignment

About 180,000 results (**0.03** sec)

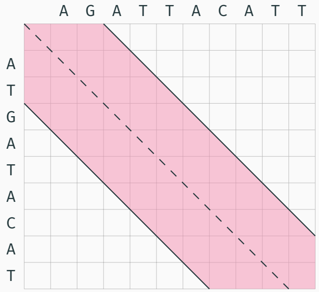
Accelerating sequence alignment

Restricting the score matrix:



Accelerating sequence alignment

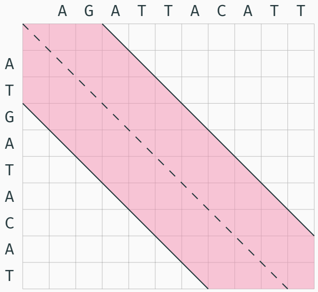
Restricting the score matrix:



Vectorizing operations:
SSE / AVX instruction set

Accelerating sequence alignment

Restricting the score matrix:



Vectorizing operations:
SSE / AVX instruction set

Hardware acceleration:
using GPU / FPGA

My contribution

Constraints of the algorithm

Constraints we have to deal with:

- very long sequences (20000 bases)
- errors can be agglomerated

Constraints of the algorithm

Constraints we have to deal with:

- very long sequences (20000 bases)
- errors can be agglomerated

but sequences are very similar

Constraints of the algorithm

Constraints we have to deal with:

- very long sequences (20000 bases)
- errors can be agglomerated

but sequences are very similar

Desired properties:

- subquadratic time complexity
- low memory footprint

Constraints of the algorithm

Constraints we have to deal with:

- very long sequences (20000 bases)
- errors can be agglomerated

but sequences are very similar

Desired properties:

- subquadratic time complexity
- low memory footprint
- easy parallelization

Constraints of the algorithm

Constraints we have to deal with:

- very long sequences (20000 bases)
- errors can be agglomerated

but sequences are very similar

Desired properties:

- subquadratic time complexity
- low memory footprint
- easy parallelization
- reduced instruction set

Overview of the algorithm

ATCGGGCAATTAAAAGGATCTGAAGCGAAGACACCGTACCAGACGTAGCGAGCCCTATTT

ATCGGAGCAATAAAAGGATCCGAGCGAGAGACCACCGTACAGGACTTAGGGAGCCCTATTT

take advantage of the similarity between sequences

Overview of the algorithm

ATCGGGCAATTAAAAGGATCTGAAGCGAAGACACCGTACCAGACGTAGCGAGCCCTATTT

ATCGGAGCAATAAAAGGATCCGAGCGAGAGACCACCGTACAGGACTTAGGGAGCCCTATTT

1. mark the words of size k common to both sequences

Overview of the algorithm

```
ATCGG GCAAT TAAAAGG ATCTGAAGCGA AGA  CACCGTA CCA GACGTAGC GAGCCCT ATTT

ATCGGAGCAA  TAAAAGG ATCCGA GCGAGAGAC CACCGTA  CAGGACTTAGG GAGCCCT ATTT
```

1. mark the words of size k common to both sequences

Overview of the algorithm

ATCGG GCAAT **TAAAAGG** ATCTGAAGCGA AGA **CACCGTA** CCA GACGTAGC **GAGCCCT** ATTT

ATCGGAGCAA **TAAAAGG** ATCCGA GCGAGAGAC **CACCGTA** CAGGACTTAGG **GAGCCCT** ATTT

1. mark the words of size k common to both sequences
2. align sequences between the anchor points

Overview of the algorithm

```
ATCGG_GCAAT TAAAAGG ATCTGAAGCGA_AGA_ CACCGTA CCA_GACGTAGC GAGCCCT ATTT
||||| ||||          ||| || |||| |||           || ||| |||          ||||
ATCGGAGCAA_ TAAAAGG ATCCGA_GCGAGAGAC CACCGTA _CAGGACTTAGG GAGCCCT ATTT
```

1. mark the words of size k common to both sequences
2. align sequences between the anchor points

Overview of the algorithm

```
ATCGG_GCAAT TAAAAGG ATCTGAAGCGA_AGA_ CACCGTA CCA_GACGTAGC GAGCCCT ATTT
||||| ||||          ||| || |||| |||           || ||| |||          ||||
ATCGGAGCAA_ TAAAAGG ATCCGA_GCGAGAGAC CACCGTA _CAGGACTTAGG GAGCCCT ATTT
```

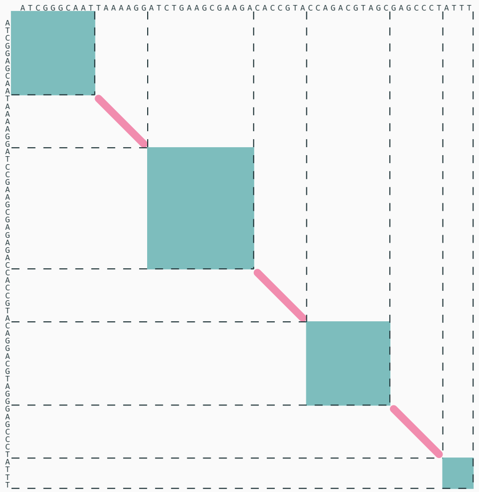
1. mark the words of size k common to both sequences
2. align sequences between the anchor points
3. merge the results

Overview of the algorithm

```
ATCGG_GCAATTAAAAGGATCTGAAGCGA_AGA_CACCGTACCA_GACGTAGCGAGCCCTATTT
||||| |||| |||||||||| || |||| ||| ||||||| || ||| ||| |||||||||||
ATCGGAGCAA_TAAAAGGATCCGA_GCGAGAGACCACCGTA_CAGGACTTAGGGAGCCCTATTT
```

1. mark the words of size k common to both sequences
2. align sequences between the anchor points
3. merge the results

Fragmented alignment


A T C G G G C A A T T A A A A G G A T C T G A A G C G A A G A C A C C G T A C C A G A C G T A G C G A G C C C T A T T T
A T C G G A G C A A T A A A A G G A T C C G A A G C G A G A G A C C A C C G T A C A G G A C G T A G G G A G C C C T A T T T

Aligning subsequences

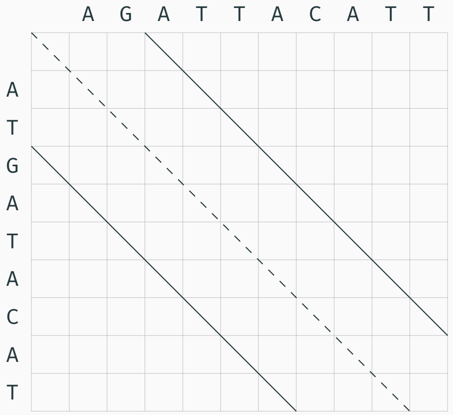
Apply existing optimizations:

		A	G	A	T	T	A	C	A	T	T
A											
T											
G											
A											
T											
A											
C											
A											
T											

Aligning subsequences

Apply existing optimizations:

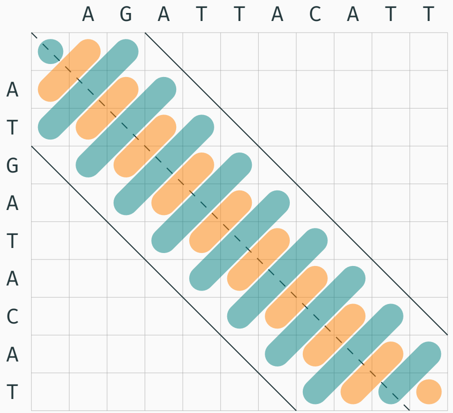
- restricting the matrix



Aligning subsequences

Apply existing optimizations:

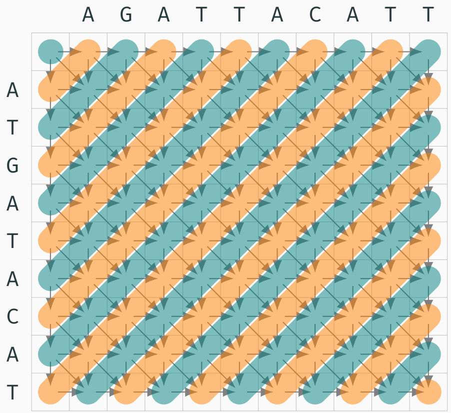
- restricting the matrix
- vectorizing operations



Vectorizing operations

$$S_{i,j} = \max\left\{S_{i-1,j} - \delta, S_{i,j-1} - \delta, S_{i-1,j-1} + m_{i,j}\right\}$$

$$\text{with } m_{i,j} = \begin{cases} \alpha \text{ if } u_i = v_j \\ -\beta \text{ otherwise} \end{cases}$$

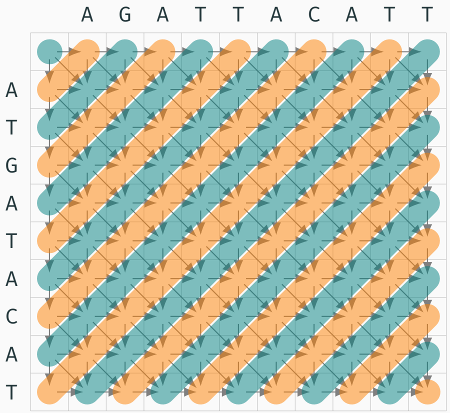


Vectorizing operations

$$S_{i,j} = \max\{S_{i-1,j} - \delta, S_{i,j-1} - \delta, S_{i-1,j-1} + m_{i,j}\}$$

with $m_{i,j} = \begin{cases} \alpha \text{ if } u_i = v_j \\ -\beta \text{ otherwise} \end{cases}$

1. `sum` (match, substitution, indel)

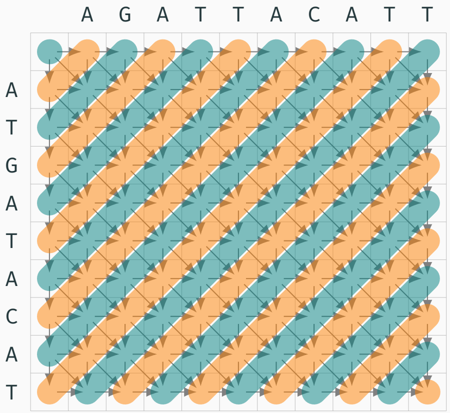


Vectorizing operations

$$S_{i,j} = \max\{S_{i-1,j} - \delta, S_{i,j-1} - \delta, S_{i-1,j-1} + m_{i,j}\}$$

with $m_{i,j} = \begin{cases} \alpha \text{ if } u_i = v_j \\ -\beta \text{ otherwise} \end{cases}$

1. `sum` (match, substitution, indel)
2. `compare`

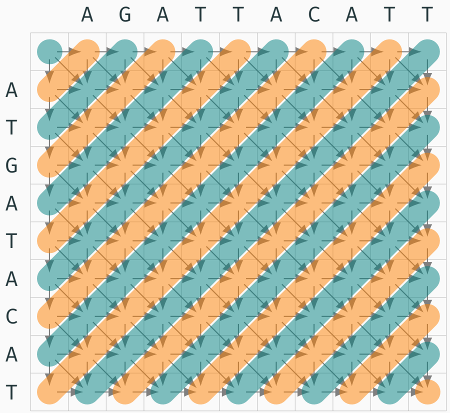


Vectorizing operations

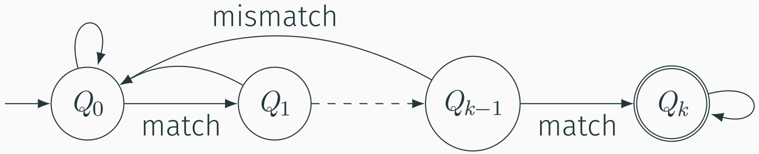
$$S_{i,j} = \max\{S_{i-1,j} - \delta, S_{i,j-1} - \delta, S_{i-1,j-1} + m_{i,j}\}$$

with $m_{i,j} = \begin{cases} \alpha \text{ if } u_i = v_j \\ -\beta \text{ otherwise} \end{cases}$

1. `sum` (match, substitution, indel)
2. `compare`
3. `blend` (keep the maximum)



Probabilistic modeling

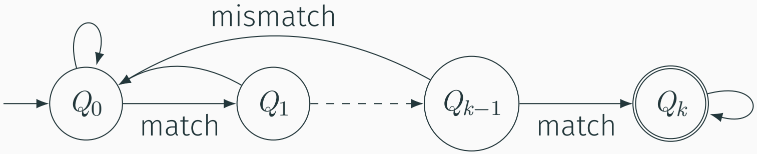


Probabilistic modeling

$$\begin{tikzcd}
{} \arrow[r] & Q_0 \arrow[loop above] \arrow[r, "\text{match}"'] & Q_1 \arrow[l, bend right] \arrow[r, dashed] & Q_{k-1} \arrow[ll, bend right, "\text{mismatch}"'] \arrow[r, "\text{match}"'] & Q_k \arrow[loop right]
\end{tikzcd}$$

$$X_n = \begin{bmatrix} p & \cdots & p & 0 \\ 1-p & & (0) & \vdots \\ & \ddots & & 0 \\ (0) & & 1-p & 1 \end{bmatrix}^n \times \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Probabilistic modeling



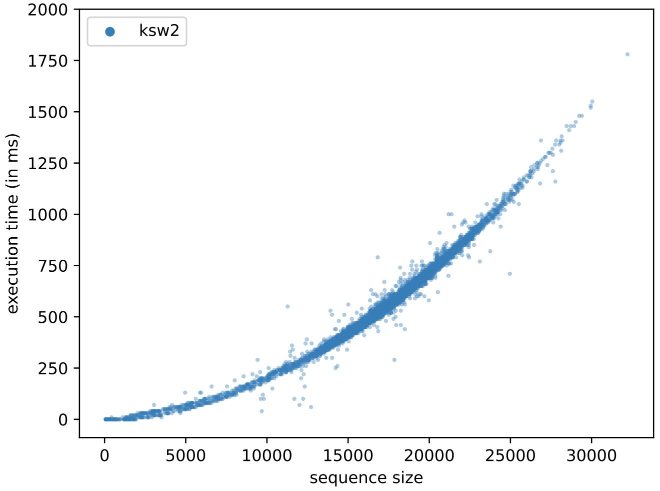
$$X_n = \begin{bmatrix} p & \cdots & p & 0 \\ 1-p & & (0) & \vdots \\ & \ddots & & 0 \\ (0) & & 1-p & 1 \end{bmatrix}^n \times \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

with $k = 16$ and $p = 0.07$,

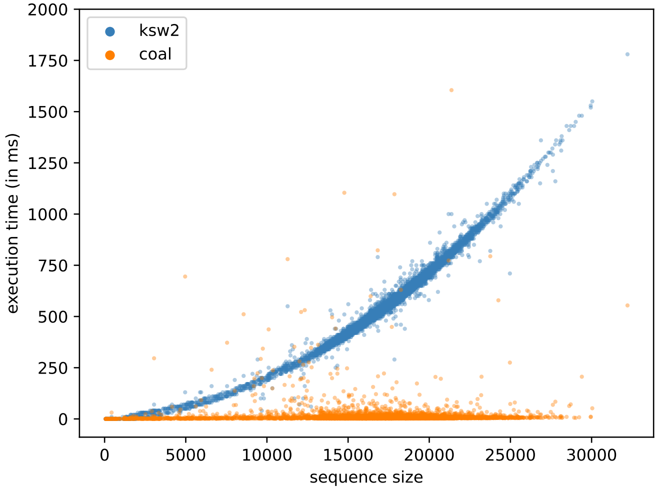
$$\mathbb{E}\,(\text{\# steps to reach } Q_k) \approx 31$$

Results

Run time measurement



Run time measurement



Run time comparison

	sequence size			
	14000–16000	16000–18000	18000–20000	20000–22000
	average run time (in ms)			
`ksw2`	430	531	645	776
`coal`	10.2	8.0	7.2	10.8
acceleration	$\times$ 42	$\times$ 74	$\times$ **90**	$\times$ 72

Score comparison

	sequence size			
	14000–16000	16000–18000	18000–20000	20000–22000
	average score			
`ksw2`	18967	21955	24543	26878
`coal`	18304	21359	23931	26071
relative gap	3.5%	2.7%	2.5%	3.0%

Next steps

Next steps

Improve scores:

- refine the choice of anchor points
- try different alignment methods

Next steps

Improve scores:

- refine the choice of anchor points
- try different alignment methods

Push performances further:

- parallelize subalignments
- improve hashing methods
- port on other architectures (*processing-in-memory*)

Processing-in-memory

processors directly integrated into the DRAM

Processing-in-memory

processors directly integrated into the DRAM

minimize data access time,
speed up data intensive calculations

many applications in genomics

Processing-in-memory

processors directly integrated into the DRAM

minimize data access time,
speed up data intensive calculations

many applications in genomics

starting in 2022:

- CIFRE PhD
- Inria GenoPIM project

Conclusion

Conclusion

- (+) better performances ($\times$ 90)
- (+) low memory footprint
- (+) adaptable with different alignment methods
- (-) lower scores (-3%)

Conclusion

- (+) better performances ($\times$ 90)
- (+) low memory footprint
- (+) adaptable with different alignment methods
- (-) lower scores (-3%)

`https://github.com/imartayan/coal`