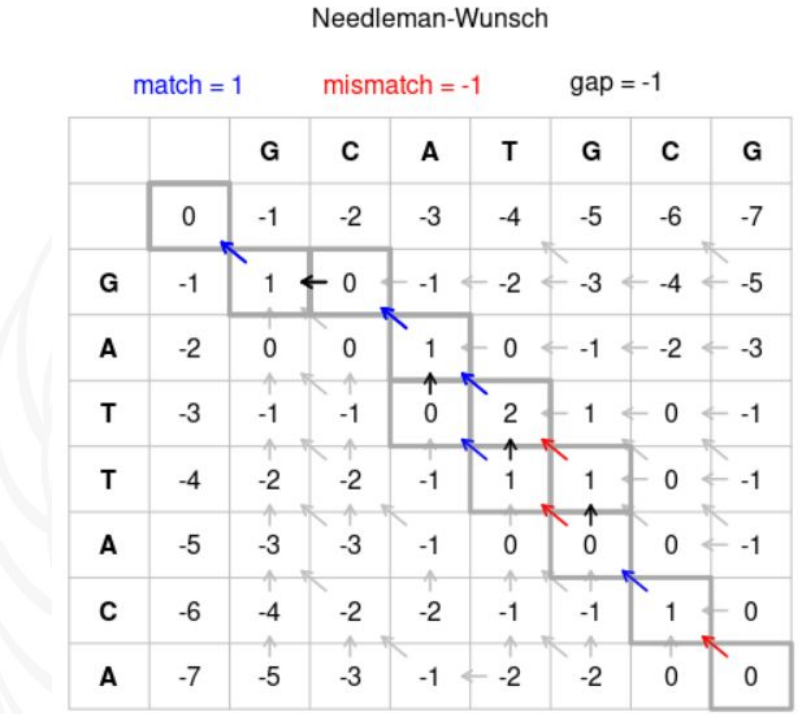
GLOBAL SEQUENCE ALIGNMENT VIA A PARALLEL-PREFIX BASED NEEDLEMAN WUNSCH ALGORITHM

Max Farrington

University at Buffalo The State University of New York

What is global sequence alignment?

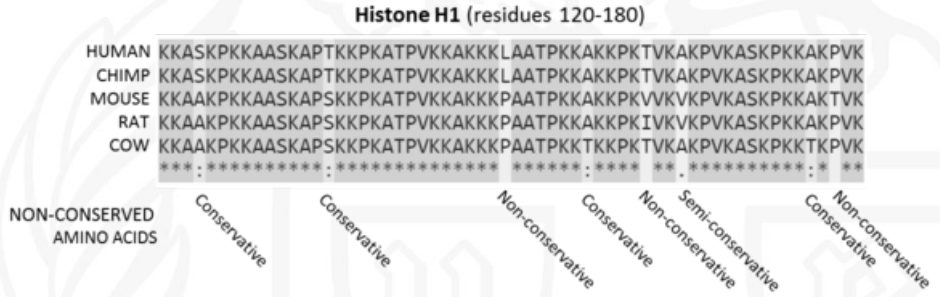
- Global sequence alignment is a bioinformatics technique for aligning two [or more] protein sequences with respects to the whole sequence.
- Every alignment is evaluated by maintaining a scoring matrix.
- Positive and negative scores are granted based on matches or mismatches.
    - Based on the use case, you can change the scoring scheme (ex: +1 match, -1 insertion/deletion, -1 mismatch)
- The best alignment is then found by backtracing from the bottom right



https://en.wikipedia.org/wiki/Needleman%E2%80%93Wunsch_algorithm

What is it actually used for?

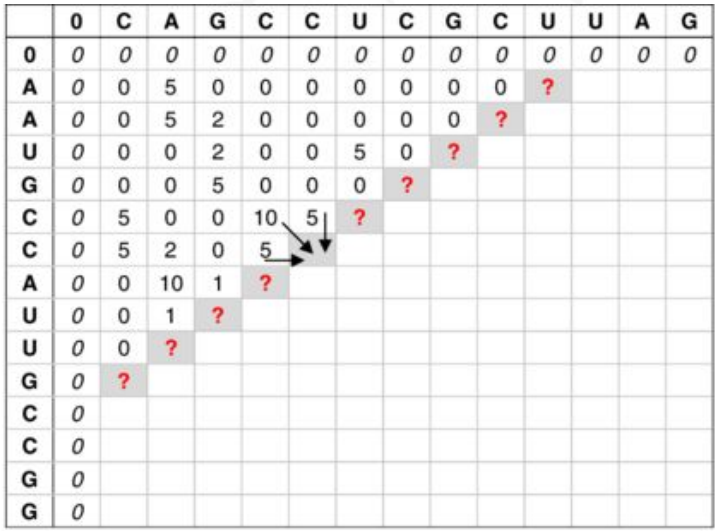
- When comparing the sequences of two subjects that share a common ancestor, you can view the mismatches, insertions, and deletions as mutations from that ancestor.
- You can then derive the importance of specific subsequences by how they are preserved in descendants of that ancestor.
- Millions of subsequences have also been tagged/identified for specific behavior.
    - You can find similarities between untagged/tagged sequences to find known genes in a sequence.



https://en.wikipedia.org/wiki/Sequence_alignment

Why is it a good parallel programming problem?

- To compute the running score, there are minimal data dependencies, allowing for computation to be done in parallel either row/column-wise, or along the anti-diagonal.
- These methods have tradeoffs in terms of efficiency and space complexity.
  - For anti-diagonal solutions, you only need to store the current and previous anti-diagonal, which changes in size as you fill the matrix.
  - For row/column wise solutions you need to store the current and previous row, but the size stays fixed.

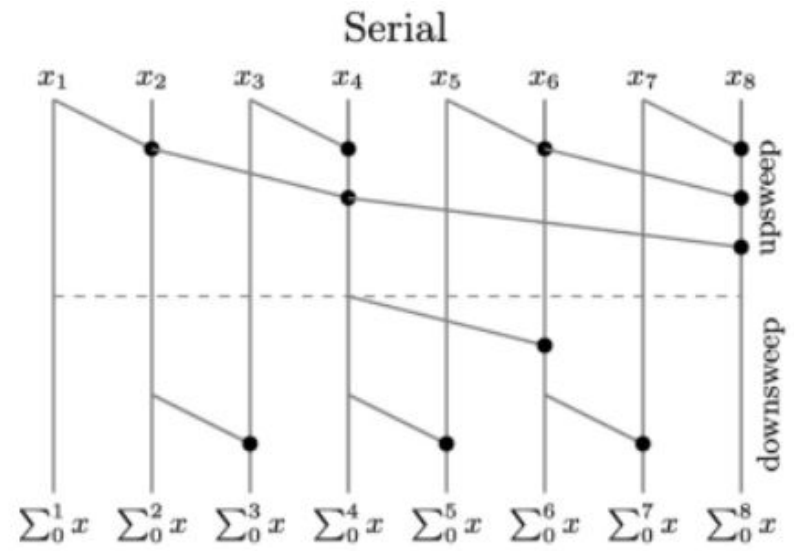


	0	C	A	G	C	C	U	C	G	C	U	U	A	G
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	5	0	0	0	0	0	0	0	?			
A	0	0	5	2	0	0	0	0	0	?				
U	0	0	0	2	0	0	5	0	?					
G	0	0	0	5	0	0	0	?						
C	0	5	0	0	10	5	?							
C	0	5	2	0	5									
A	0	0	10	1	?									
U	0	0	1	?										
U	0	0	?											
G	0	?												
C	0													
C	0													
G	0													
G	0													

https://www.researchgate.net/figure/Anti-diagonal-method-and-dependency-of-the-cells_fig11_222408669

How the algorithm relates to parallel prefix

- Parallel prefix takes a binary associative operator (such as +, -, *, MAX(), etc.) and an array of n elements, and for each element, computes and stores a running total based on the chosen operator.
- In the case of the Needleman Wunsch algorithm, we are effectively keeping a running total, but the value in each spot depends on the max value of its neighbors that have already been computed.
- Needleman Wunsch also allows for negative values in the matrix, so the work can very easily be split into chunks with the preceding values communicated via parallel prefix.



https://link.springer.com/chapter/10.1007/978-3-031-12597-3_21

$$T[i,j] = \max\begin{cases} T[i-1,j-1] + f(a_i, b_j), \\ T[i-1,j] - g, \\ T[i,j-1] - g. \end{cases}$$

$$w[j] = \max\begin{cases} T_1[i,j-1] - (g+h), \\ T_3[i,j-1] - (g+h). \end{cases}$$

Then,

$$T_2[i,j] = \max\begin{cases} w[j], \\ T_2[i,j-1] - g. \end{cases}$$

Let

$$x[j] = T_2[i,j] + jg$$
$$= \max\begin{cases} w[j] + jg \\ T_2[i,j-1] + (j-1)g, \end{cases}$$
$$= \max\begin{cases} w[j] + jg \\ x[j-1]. \end{cases}$$

Aluru et all.

Parallel example

Gap penalty: -2, mismatch: -1, match 1
A = AACTGGAA
B = CATG

		0	1	2	3	4	5	6	7	8
			A	A	C	T	G	G	A	A
	0	0	-2	-4	-6	-8	-10	-12	-14	-16
1	C	-2	**-1 (1)**	**-3 (1)**			**-9* (1)**			
2	A	-4								
3	T	-6								
4	G	-8								

$$w[j] = \max\begin{cases} T_1[i,j-1] - (g+h), \\ T_3[i,j-1] - (g+h). \end{cases} \qquad T_2[i,j] = x[j] - jg.$$

$$x[j] = T_2[i,j] + jg$$
$$= \max\begin{cases} w[j] + jg \\ T_2[i,j-1] + (j-1)g, \end{cases}$$
$$= \max\begin{cases} w[j] + jg \\ x[j-1]. \end{cases}$$

w[1] = Max(T[0,0] + Match(B[0],A[0]) ,T[0,1] - 2) = -1

x[j] = max(-1 + 1(2), -∞) = 1

T[1,1] = 1 - 1(2) = -1

w[2] = Max(T[0,1] + Match(B[0],A[1]) ,T[0,2] - 2) = -3

x[j] = max(-3 + 2(2), 1) = 1

T[1,2] = 1 - 2(2) = -3

w[5] = Max(T[0,4] + Match(B[0],A[4]) ,T[0,5] - 2) = -9

x[j] = max(-9 + 5(2), -∞) = 1

T[1,5] = 1 - 5(2) = -9

Parallel example

Gap penalty: -2, mismatch: -1, match 1
A = AACTGGAA
B = CATG

$$w[j] = \max\begin{cases} T_1[i,j-1] - (g+h), \\ T_3[i,j-1] - (g+h). \end{cases} \qquad T_2[i,j] = x[j] - jg.$$

$$\begin{aligned} x[j] &= T_2[i,j] + jg \\ &= \max\begin{cases} w[j] + jg \\ T_2[i,j-1] + (j-1)g, \end{cases} \\ &= \max\begin{cases} w[j] + jg \\ x[j-1]. \end{cases} \end{aligned}$$

	0	1	2	3	4	5	6	7	8
		A	A	C	T	G	G	A	A
	0	-2	-4	-6	-8	-10	-12	-14	-16
C	-2	-1 (1)	-3 (1)	-3* (3)	-5* (3)	-9* (1)	-11* (1)	-15* (1)	-15* (1)
A	-4								
T	-6								
G	-8								

Parallel example

Gap penalty: -2, mismatch: -1, match 1
A = AACTGGAA
B = CATG

		0	1	2	3	4	5	6	7	8
			A	A	C	T	G	G	A	A
	0		-2	-4	-6	-8	-10	-12	-14	-16
C	-2		-1 (1)	-3 (1)	-3* (3)	-5* (3)	-9* (1)	-11* (1)	-15* (1)	-15* (1)
A	-4									
T	-6									
G	-8									

$$w[j] = \max\begin{cases} T_1[i,j-1] - (g+h), \\ T_3[i,j-1] - (g+h). \end{cases} \qquad T_2[i,j] = x[j] - jg.$$

$$\begin{aligned} x[j] &= T_2[i,j] + jg \\ &= \max\begin{cases} w[j] + jg \\ T_2[i,j-1] + (j-1)g, \end{cases} \\ &= \max\begin{cases} w[j] + jg \\ x[j-1]. \end{cases} \end{aligned}$$

Prefix time!

Binary associative operator - Max(x)

Parallel example

Gap penalty: -2, mismatch: -1, match 1
A = AACTGGAA
B = CATG

$$w[j] = \max\begin{cases} T_1[i,j-1] - (g+h), \\ T_3[i,j-1] - (g+h). \end{cases} \qquad T_2[i,j] = x[j] - jg.$$

$$x[j] = T_2[i,j] + jg$$
$$= \max\begin{cases} w[j] + jg \\ T_2[i,j-1] + (j-1)g, \end{cases}$$
$$= \max\begin{cases} w[j] + jg \\ x[j-1]. \end{cases}$$

	0	1	2	3	4	5	6	7	8
		A	A	C	T	G	G	A	A
	0	-2	-4	-6	-8	-10	-12	-14	-16
C	-2	-1 (1)	-3 (1)	-3 (3)	-5 (3)	-7 (3)	-9 (3)	-11 (3)	-13 (3)
A	-4								
T	-6								
G	-8								

		A	A	C	T	G	G	A	A
	0	-2	-4	-6	-8	-10	-12	-14	-16
C	-2	-1	← -3	-3	← -5	← -7	← -9	← -11	← -13

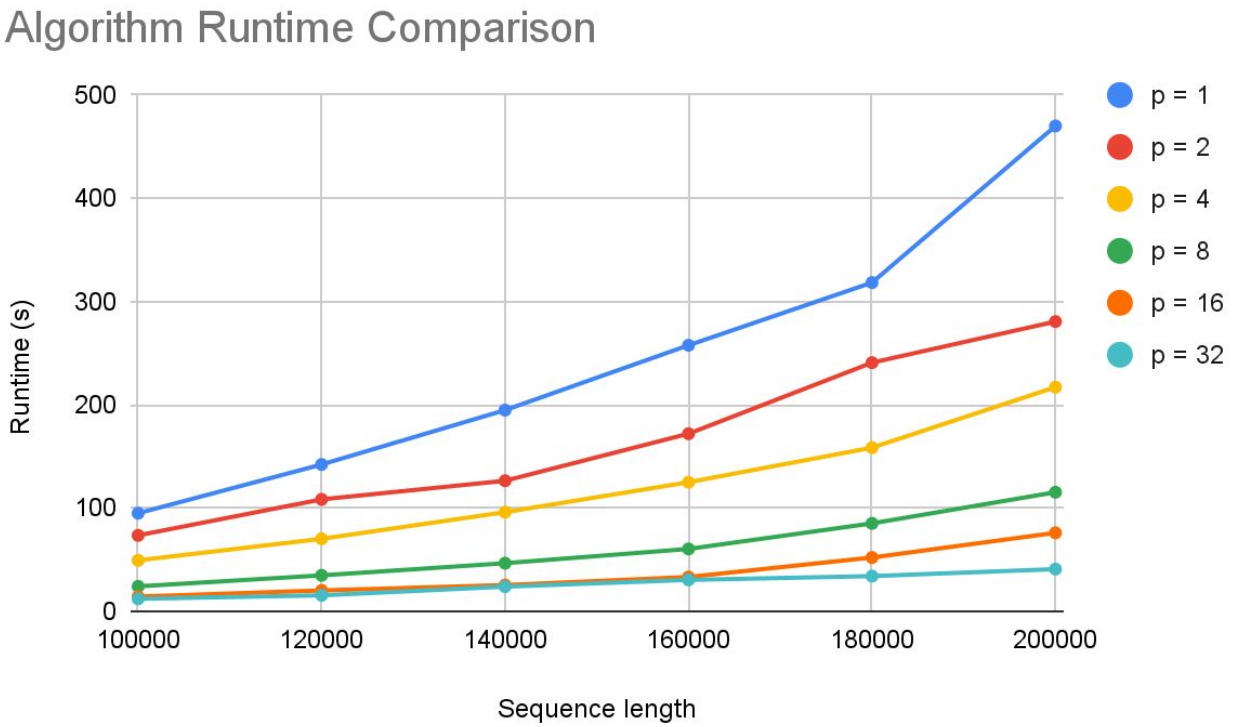
Now, recompute x[j] using the value received during the prefix scan, and use for calculating T[i,j]

Additionally, share the last value in your local row with the processor next to you

Benchmarking and scalability

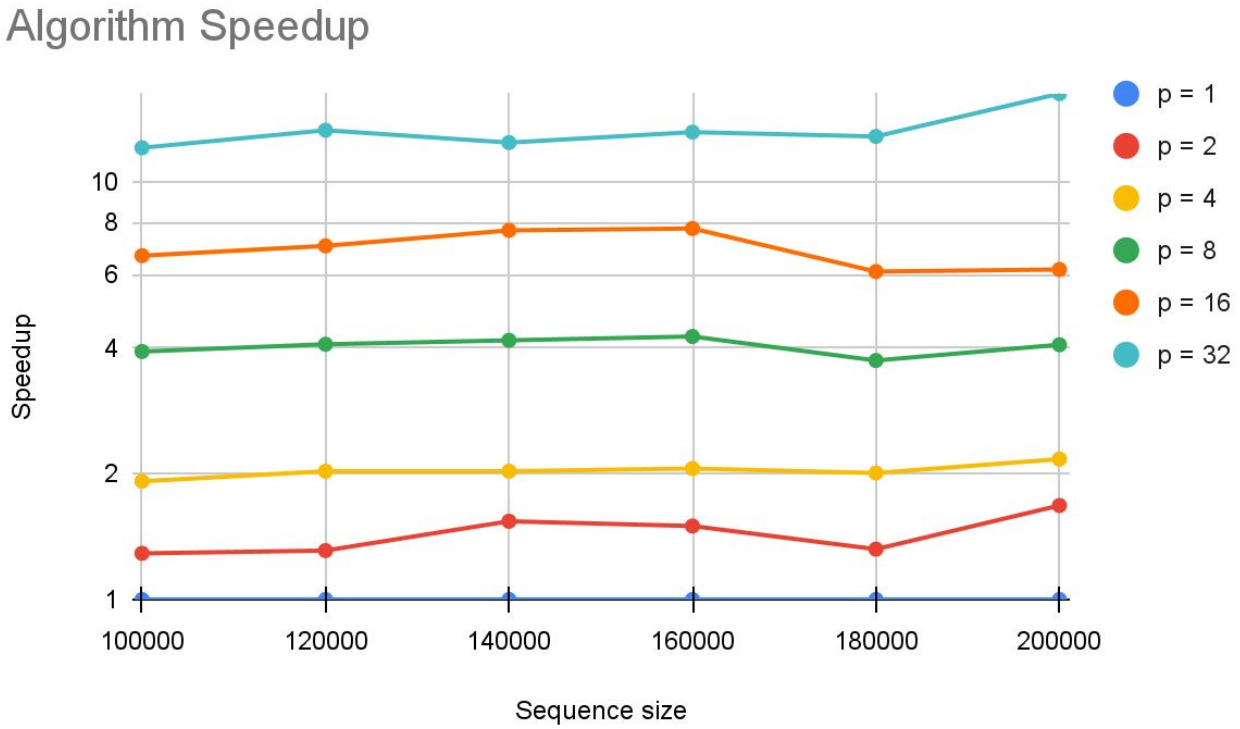
University at Buffalo The State University of New York

Runtime analysis



n/p	100000	120000	140000	160000	180000	200000
1	94.972	142.176	194.823	257.642	318.329	470.006
2	73.5367	108.38	126.419	171.986	240.79	280.597
4	49.3603	70.1906	95.9919	125	158.477	217.044
8	24.1509	34.7976	46.5727	60.3988	85.1384	115.308
16	14.2428	20.195	25.385	33.2346	52.0935	75.9925
32	7.85093	10.6655	15.6509	19.5389	24.7102	28.8214

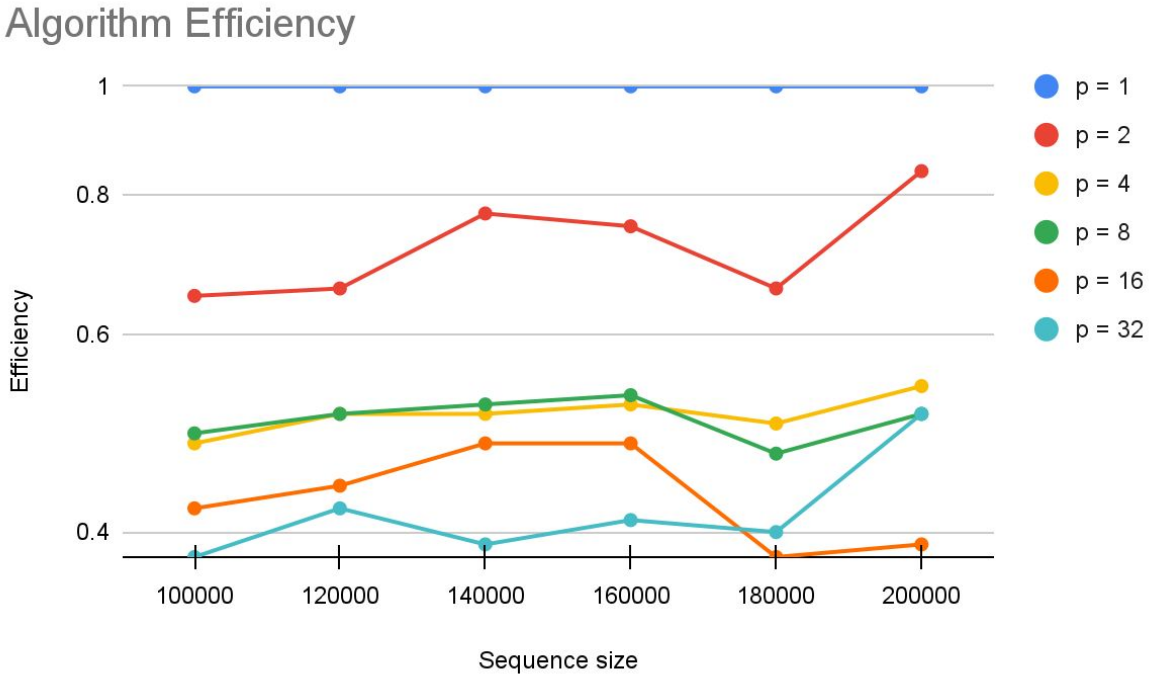
Speedup



	100000	120000	140000	160000	180000	200000
1	1	1	1	1	1	1
2	1.29	1.31	1.54	1.5	1.32	1.68
4	1.92	2.03	2.03	2.06	2.01	2.17
8	3.93	4.09	4.18	4.27	3.74	4.08
16	6.67	7.04	7.67	7.75	6.11	6.18
32	12.1	13.33	12.45	13.19	12.88	16.31

$$\text{Speedup} = \frac{T_1}{T_p}$$

Efficiency



	100000	120000	140000	160000	180000	200000
1	1	1	1	1	1	1
2	0.65	0.66	0.77	0.75	0.66	0.84
4	0.48	0.51	0.51	0.52	0.5	0.54
8	0.49	0.51	0.52	0.53	0.47	0.51
16	0.42	0.44	0.48	0.48	0.38	0.39
32	0.38	0.42	0.39	0.41	0.4	0.51

$$\text{Efficiency} = \frac{T_1}{pT_p}$$

Conclusions

- Strong scaling is shown for p=2 (efficiency >= 65%) according to amdahl's law for n = 100000 through n = 200000.
- No weak scaling according to Gustafson's law with fixed ratio of problem size to processors
- Problem size during testing may not have been high enough, as performance showed some promise for higher problem size.
    - Ran into bottleneck with ram, but further testing development is required for implementing a more space efficient solution
- Better utilization of on-chip parallelism would have likely greatly increased the performance of the algorithm with MPI.

References

- https://en.wikipedia.org/wiki/Needleman%E2%80%93Wunsch_algorithm
- https://en.wikipedia.org/wiki/Sequence_alignment
- https://www.researchgate.net/figure/Anti-diagonal-method-and-dependency-of-the-cells_fig11_222408669
- https://link.springer.com/chapter/10.1007/978-3-031-12597-3_21
- Srinivas Aluru, Natsuhiko Futamura, Kishan Mehrotra, Parallel biological sequence comparison using prefix computations,
- https://bioboot.github.io/bimm143_W20/class-material/nw/