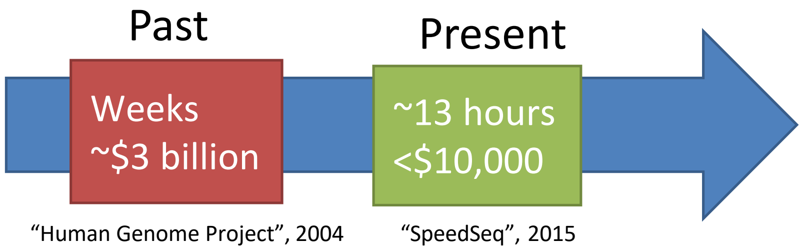
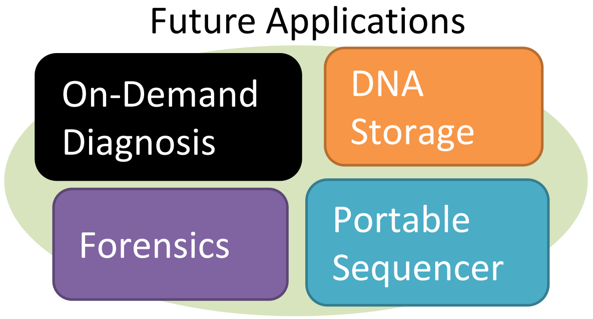
Accelerating Genomic Sequence Alignment Workload with Scalable Vector Architecture

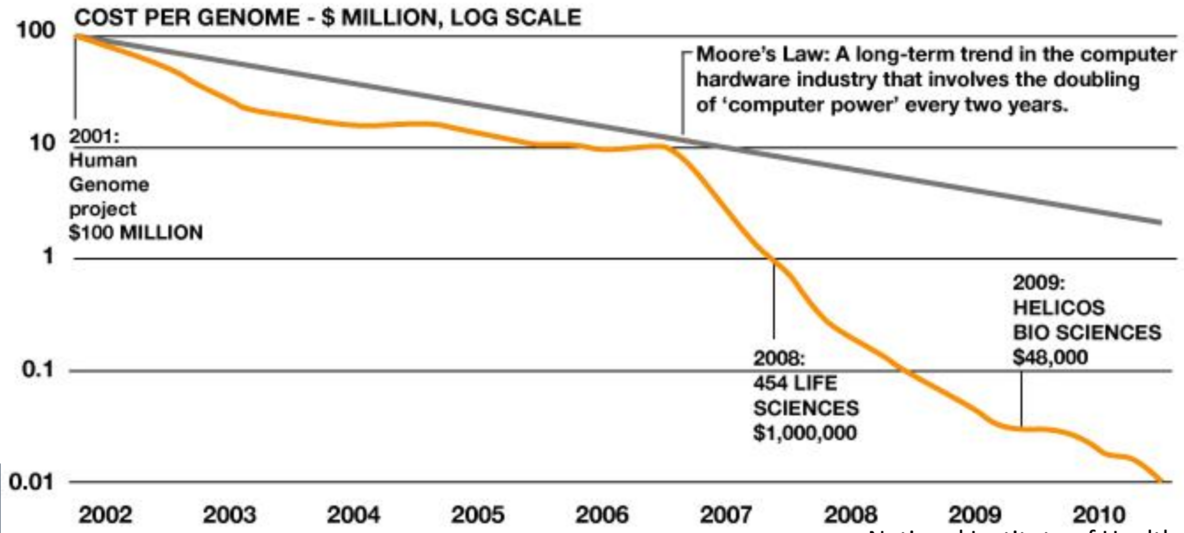
Dong-hyeon Park, Jon Beaumont, Trevor Mudge

University of Michigan, Ann Arbor

Genomics





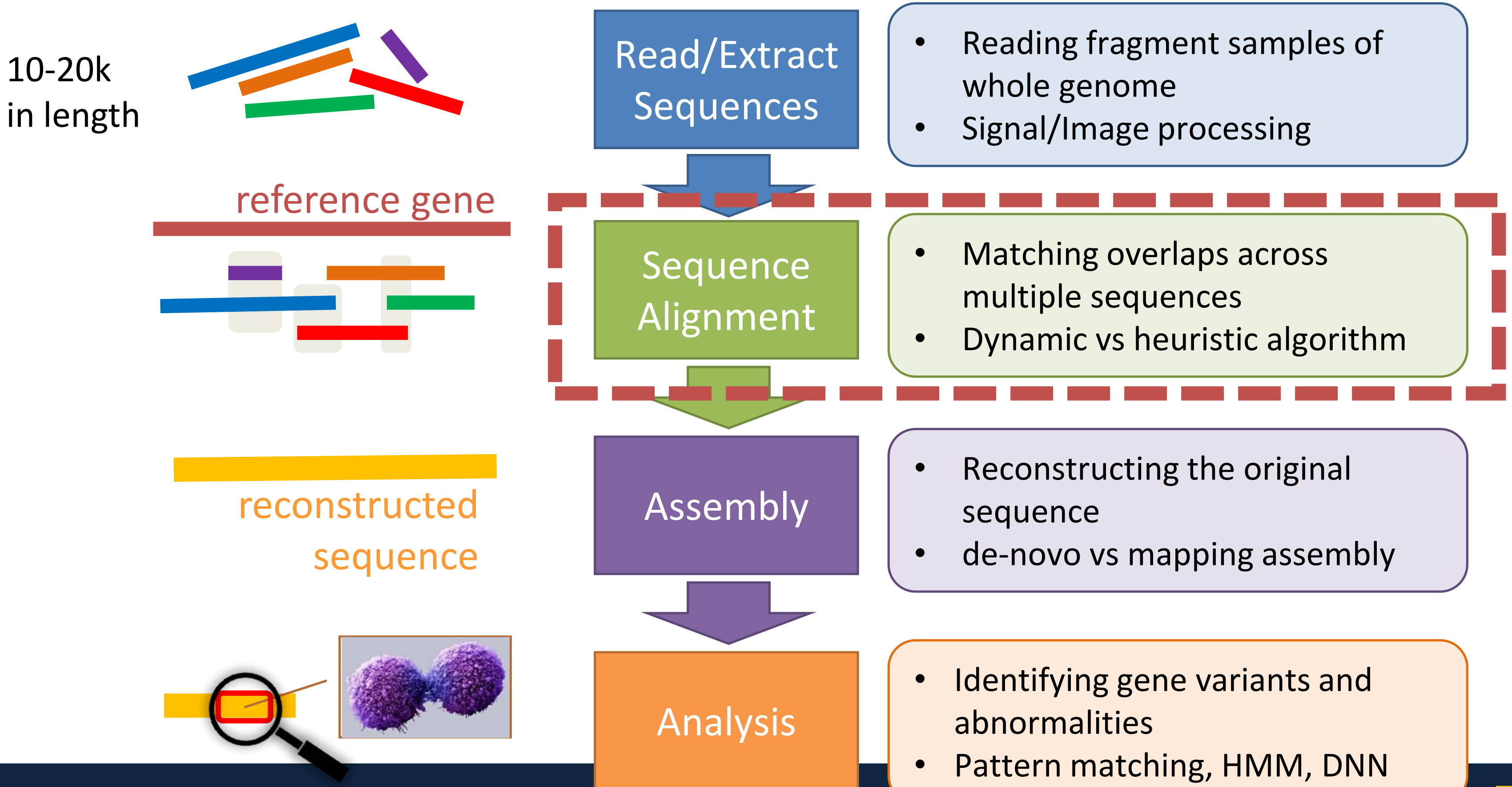


source: National Institute of Health

Human Genome:
3.2 billion base pairs

Need to sample at 30-50x coverage

Whole Genome Sequencing Pipeline



- Reading fragment samples of whole genome
- Signal/Image processing

- Matching overlaps across multiple sequences
- Dynamic vs heuristic algorithm

- Reconstructing the original sequence
- de-novo vs mapping assembly

- Identifying gene variants and abnormalities
- Pattern matching, HMM, DNN

Target Architecture:

Scalable Vector Extension (SVE)

ARM’s Scalable Vector Extension (SVE)

- Designed to complement existing SIMD architecture (NEON)

- Key Features:
    - Scalable Vector Length (128 - 2048-bits)
    - Per-lane Predication (32 SIMD Reg. + 16 Predicate Reg.)
    - Gather-load and scatter-store
    - Horizontal vector operations

→ **Vector Length Agnostic Code**

ARM’s Scalable Vector Extension (SVE)

- Genomic sequences are sampled at different lengths depending on the device used for sampling:
  - Illumina HiSeq System: 30-300 bps
  - Sanger 3730xl: 400-900 bps

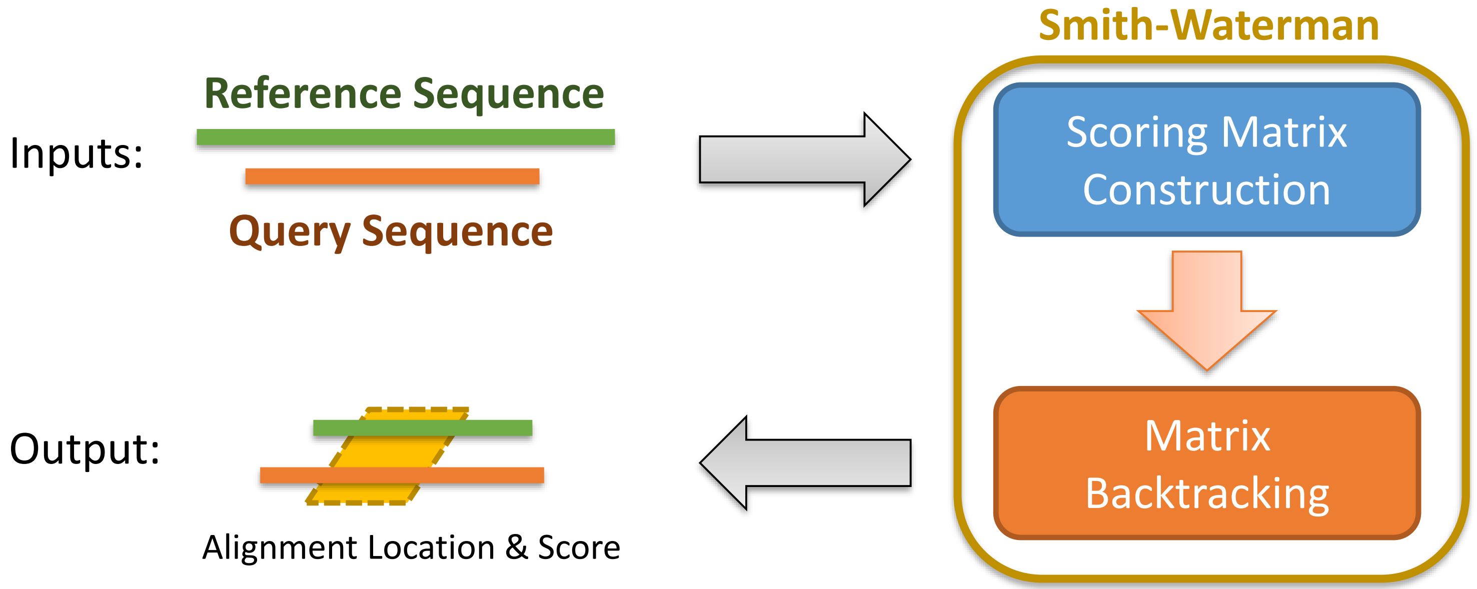
Vector-Length Agnostic Code can be used to **Dynamically Choose the Optimal SIMD Width**

Target Algorithm:

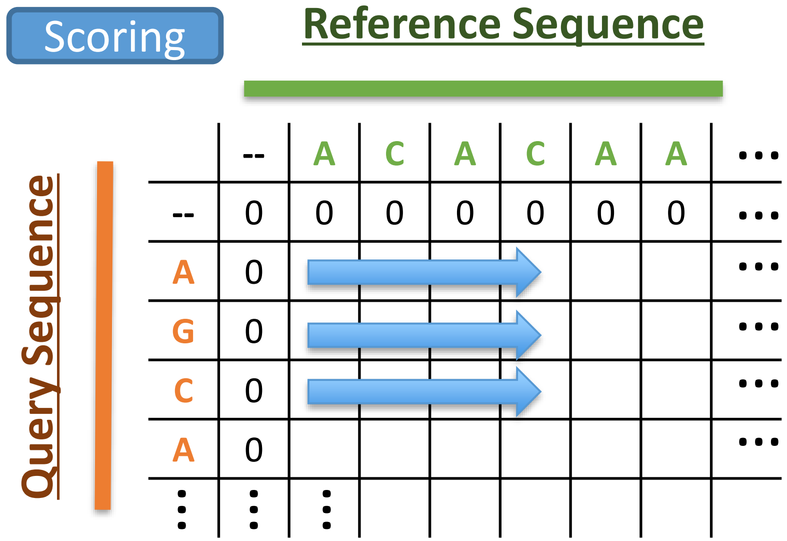
Smith-Waterman
Sequence Alignment

Smith-Waterman Algorithm

Local sequence alignment algorithm developed in 1981



Scoring Matrix Construction



	--	A	C	A	C	A	A	⋯
--	0	0	0	0	0	0	0	⋯
A	0							⋯
G	0							⋯
C	0							⋯
A	0							⋯
⋮	⋮	⋮						

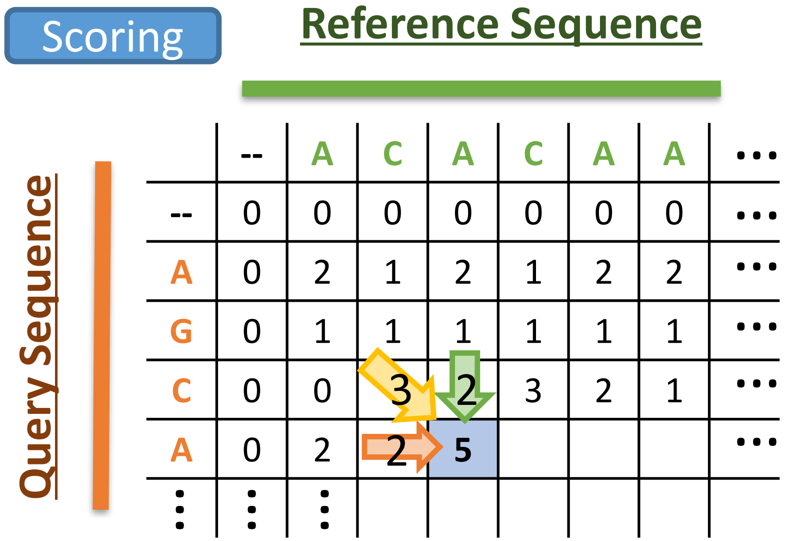
$$H(m,n) = max \begin{cases} E(m,n) \\ F(m,n) \\ H(m-1,n-1) + S(a_m, b_n) \end{cases}$$

$$E(m,n) = max \begin{cases} H(m,n-1) - g_o \\ E(m,n-1) - g_e \end{cases}$$

$$F(m,n) = max \begin{cases} H(m-1,n) - g_o \\ F(m-1,n) - g_e \end{cases}$$

Scoring Matrix Construction

Scoring



	--	A	C	A	C	A	A	…
--	0	0	0	0	0	0	0	…
A	0	2	1	2	1	2	2	…
G	0	1	1	1	1	1	1	…
C	0	0	3	2	3	2	1	…
A	0	2	2	5				…
⋮	⋮	⋮						

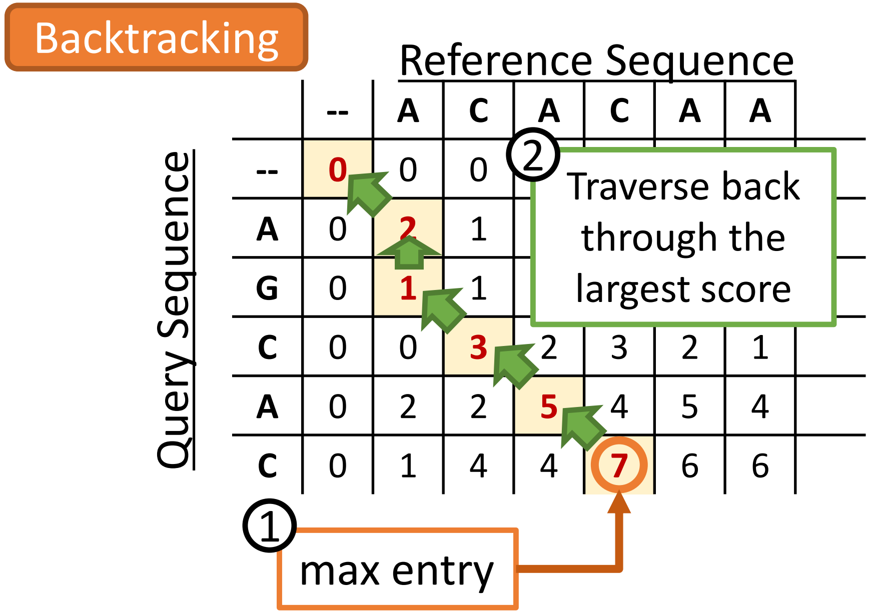
$$H(m,n) = max \begin{cases} E(m,n) \\ F(m,n) \\ H(m-1,n-1) + S(a_m, b_n) \end{cases}$$

$$E(m,n) = max \begin{cases} H(m,n-1) - g_o \\ E(m,n-1) - g_e \end{cases}$$

$$F(m,n) = max \begin{cases} H(m-1,n) - g_o \\ F(m-1,n) - g_e \end{cases}$$

Backtracking

Finds the best local alignment from the scoring matrix



	--	A	C	A	C	A	A
--	0	0	0				
A	0	2	1				
G	0	1	1				
C	0	0	3	2	3	2	1
A	0	2	2	5	4	5	4
C	0	1	4	4	7	6	6

Step 2.

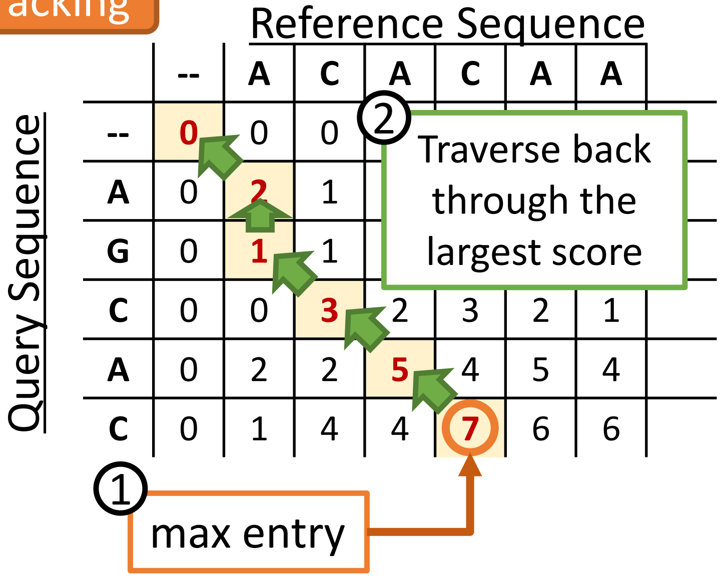
Check the adjacent entries for the next largest score

Move to the entry with the largest score and continue the path

Backtracking

Finds the best local alignment from the scoring matrix

Backtracking



Step 3.

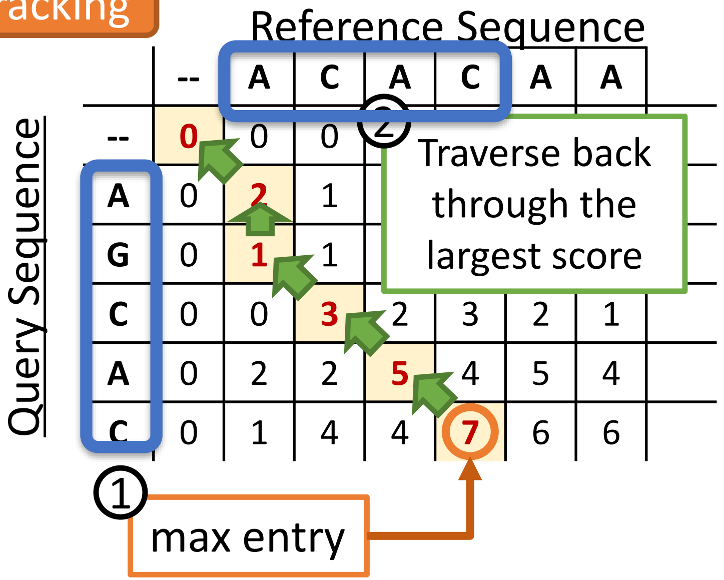
Get the resulting alignment

Path Direction	Alignment
Horizontal	Deletion
Vertical	Insertion
Diagonal	Match

Backtracking

Finds the best local alignment from the scoring matrix

Backtracking



	--	A	C	A	C	A	A
--	**0**	0	0				
A	0	**2**	1				
G	0	**1**	1				
C	0	0	**3**	2	3	2	1
A	0	2	2	**5**	4	5	4
C	0	1	4	4	**7**	6	6

Step 3.

Get the resulting alignment

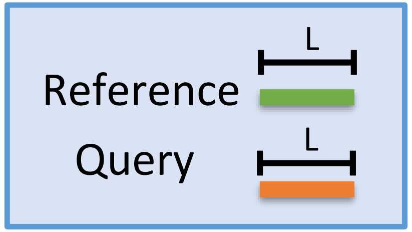
3

Reference:	`A-CAC`
Query:	`AGCAC`

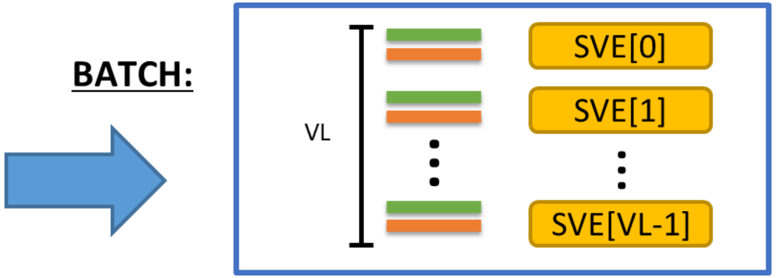
Insertion

Alignment Score: 7

Vectorization

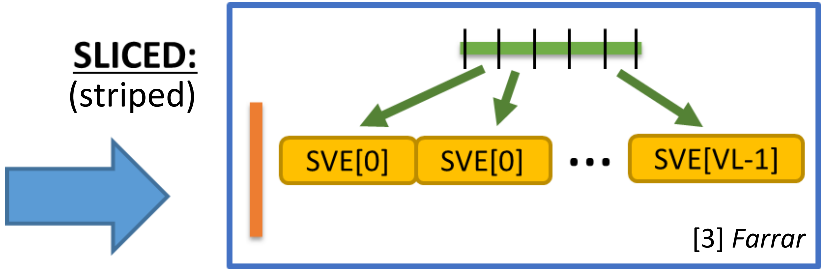


BATCH:

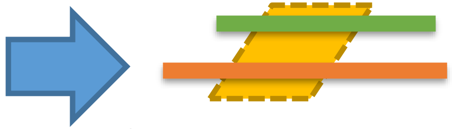


[4] *Rognes*

SLICED:
(striped)

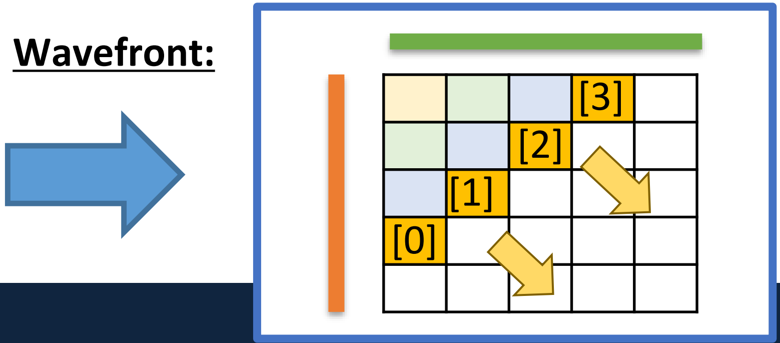


[3] *Farrar*



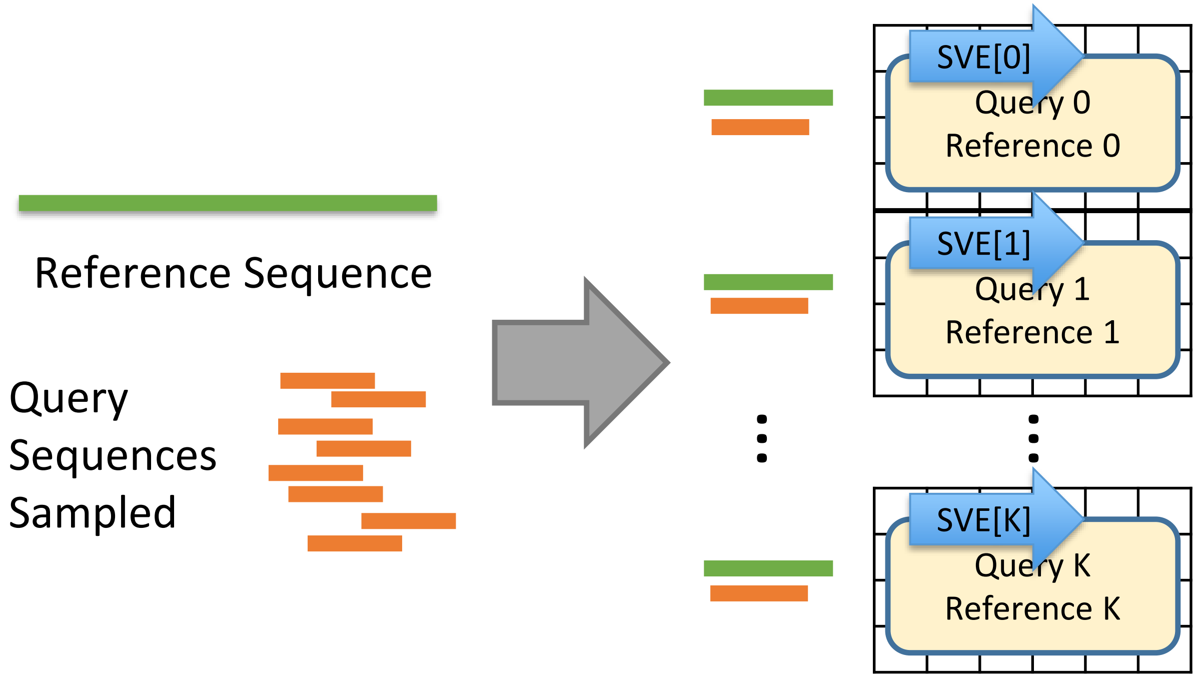
Alignment Location & Score

Wavefront:



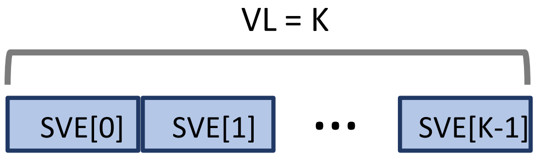
[2] *Wozniak* et al

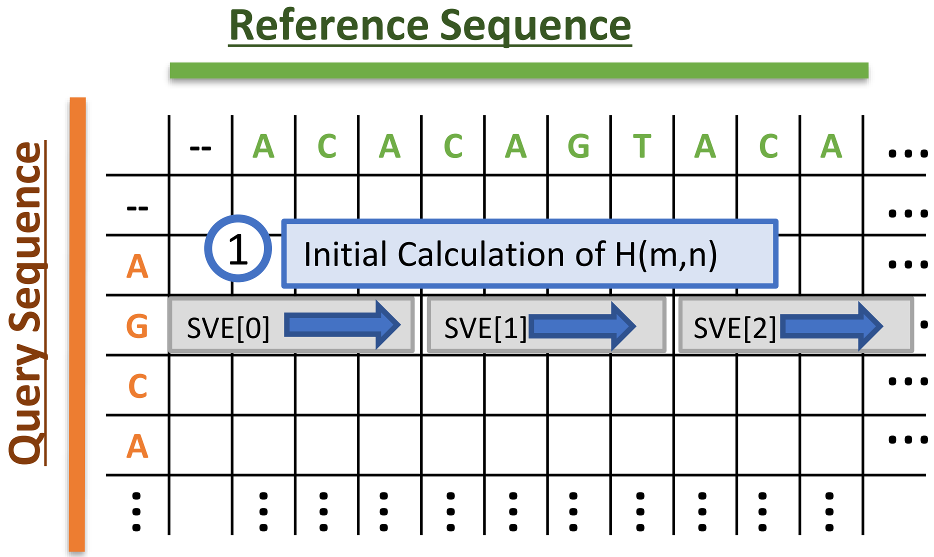
Batch Smith-Waterman



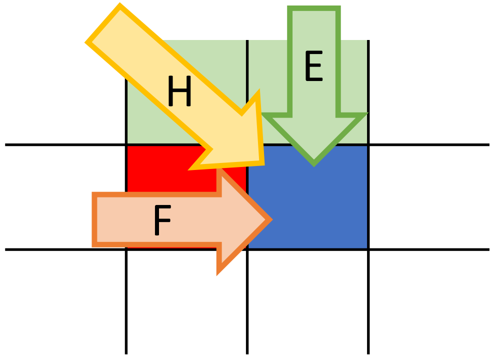
[4] *Rognes*

Sliced Smith-Waterman
(striped)



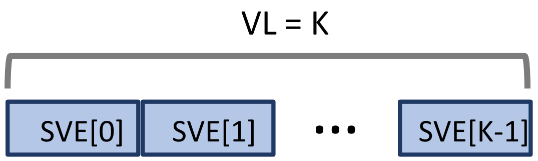


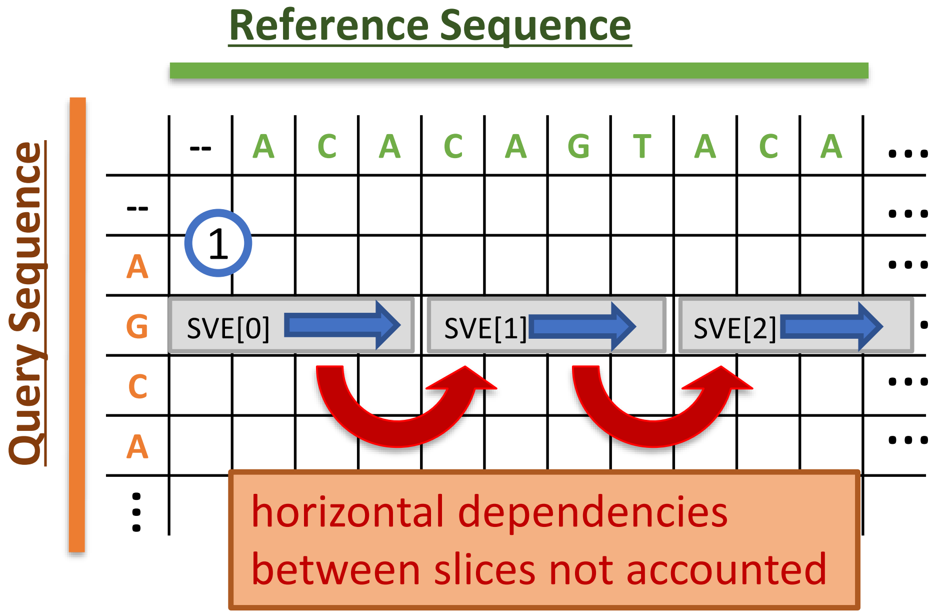
$$H(m,n) = max \begin{cases} E(m,n) \\ F(m,n) \\ H(m-1,n-1) + S(a_m, b_n) \end{cases}$$



[3] *Farrar*

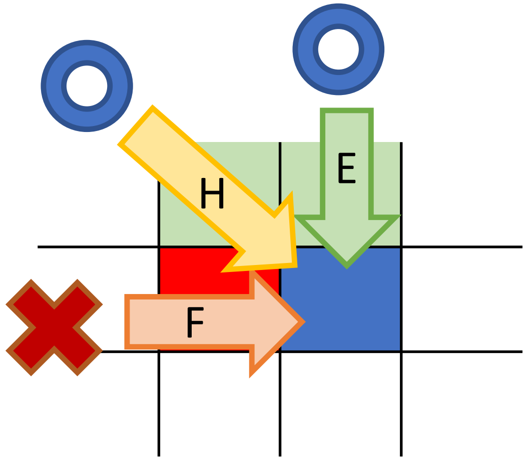
Sliced Smith-Waterman
(striped)





horizontal dependencies between slices not accounted

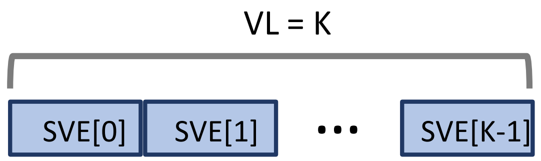
Value of F need to be re-calculated



[3] *Farrar*

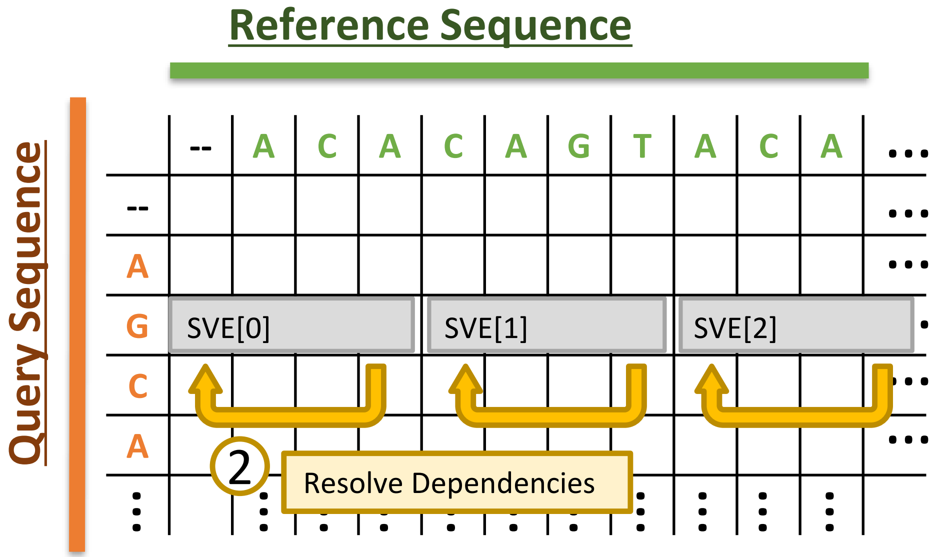
Sliced Smith-Waterman

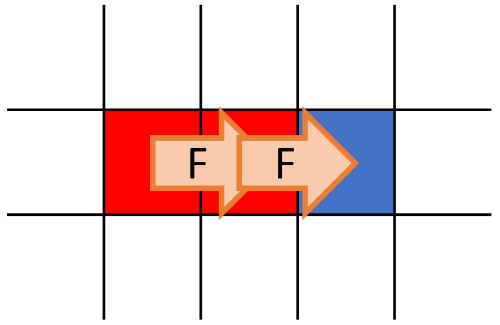
(striped)



$$F(m,n) = max\begin{cases} H(m-1,n) - g_o \\ F(m-1,n) - g_e \end{cases}$$

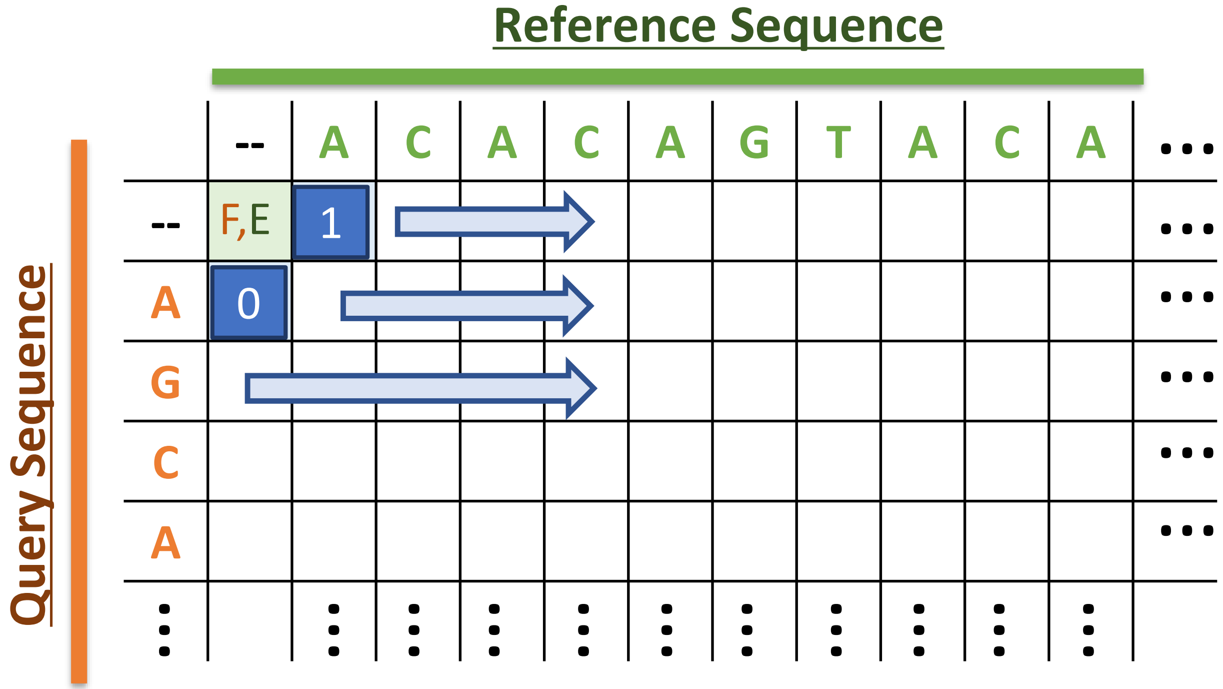
$$\boldsymbol{H(m,n)} = \max(\boldsymbol{F(m,n)}, \ \boldsymbol{H(m,n)})$$



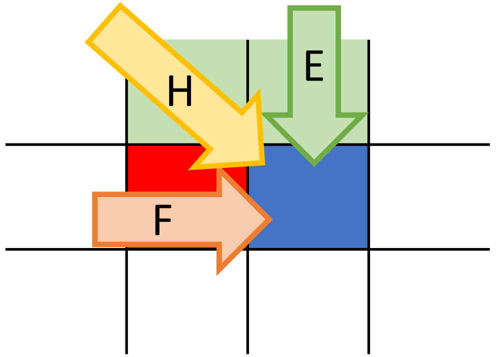


[3] *Farrar*

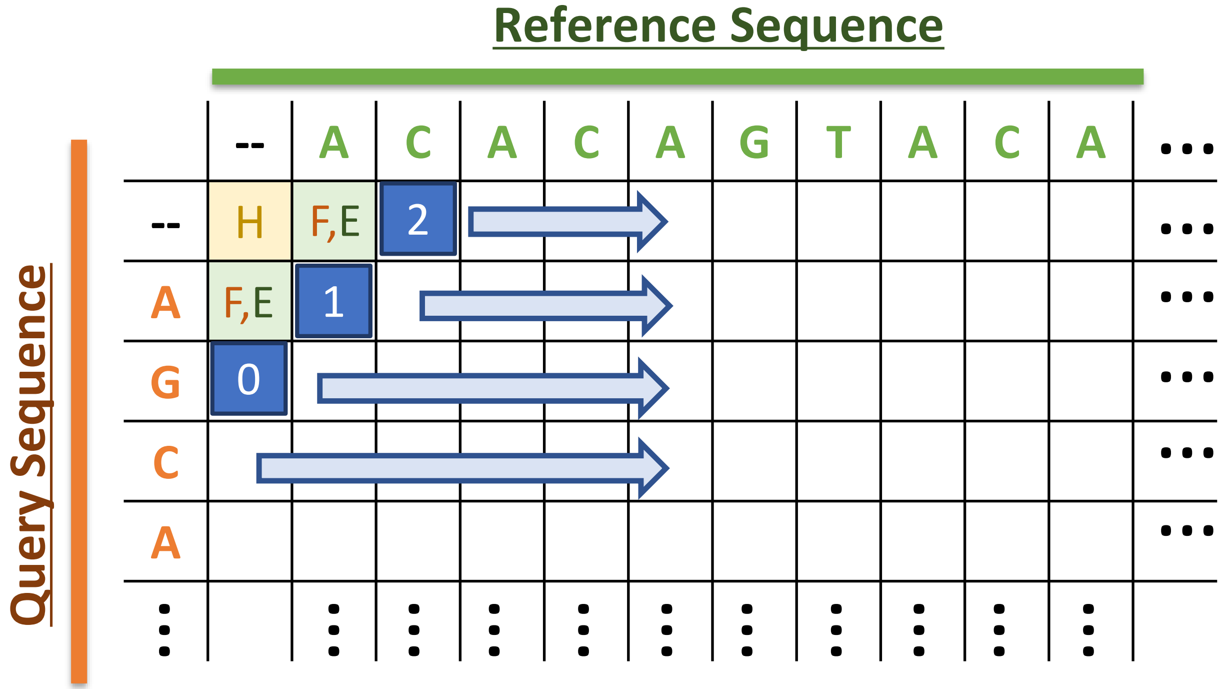
Wavefront Smith-Waterman



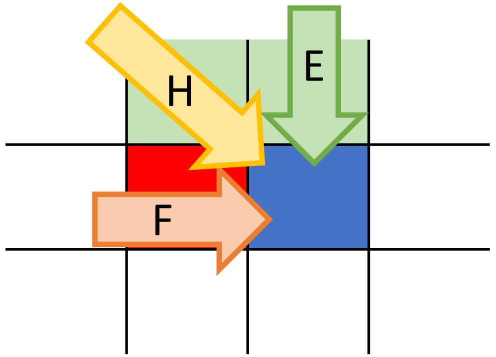
All dependency comes from previous execution



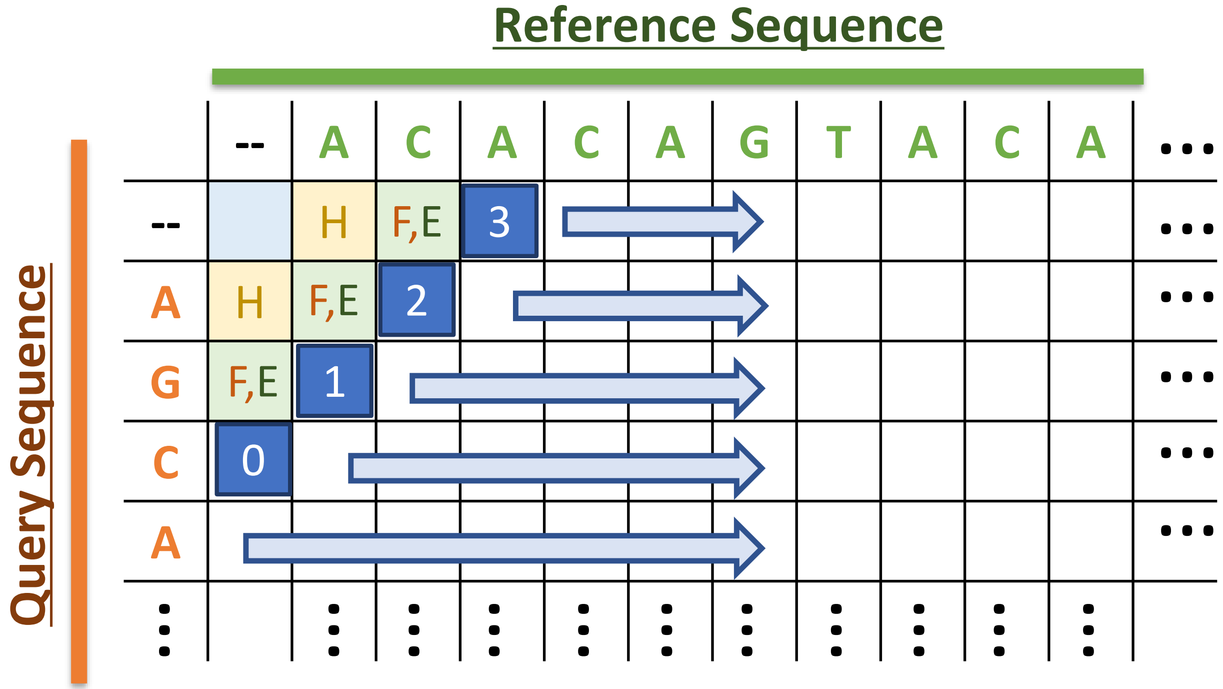
Wavefront Smith-Waterman



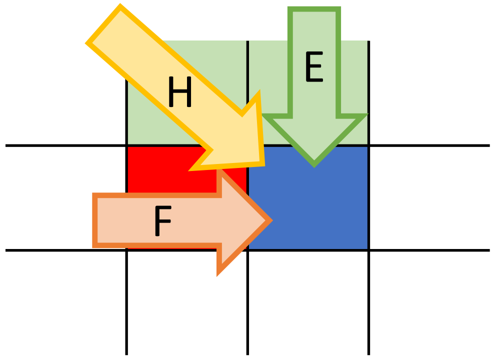
All dependency comes from previous execution



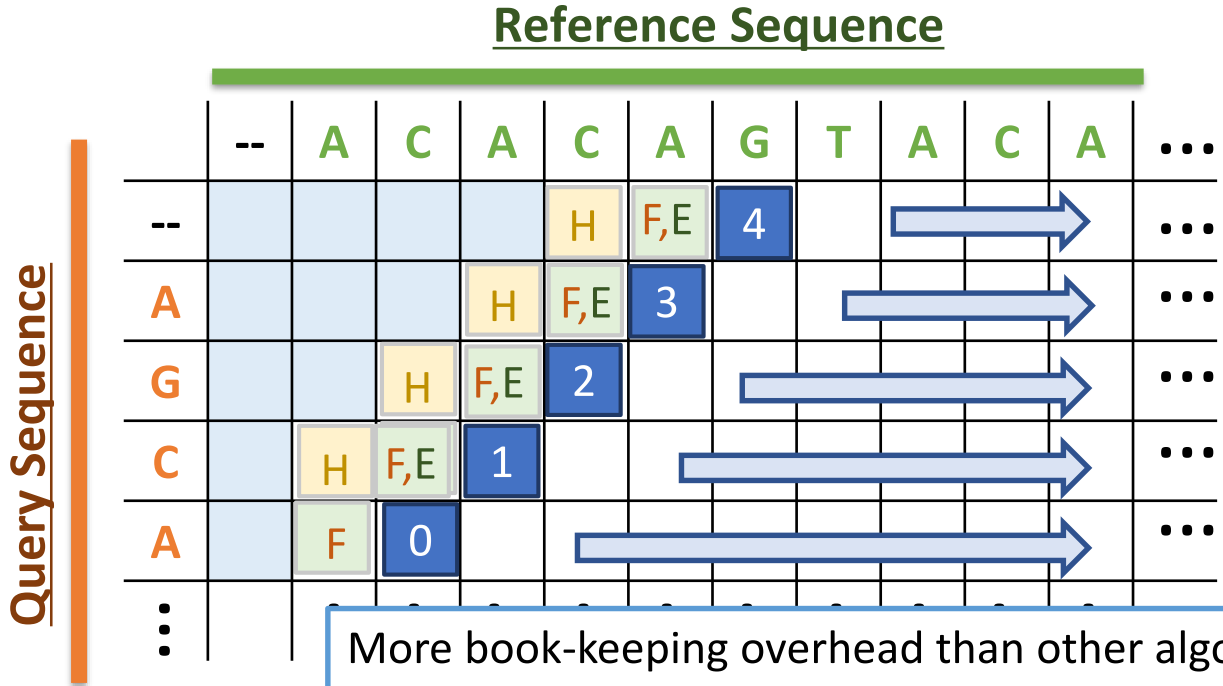
Wavefront Smith-Waterman



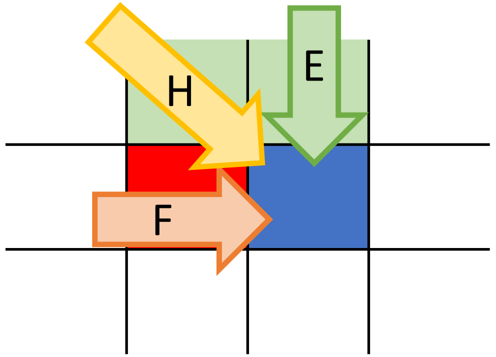
All dependency comes from previous execution



Wavefront Smith-Waterman



All dependency comes from previous execution



More book-keeping overhead than other algorithms:

- Keep track of H values of two prev. iterations
- F and E values from prev. iteration

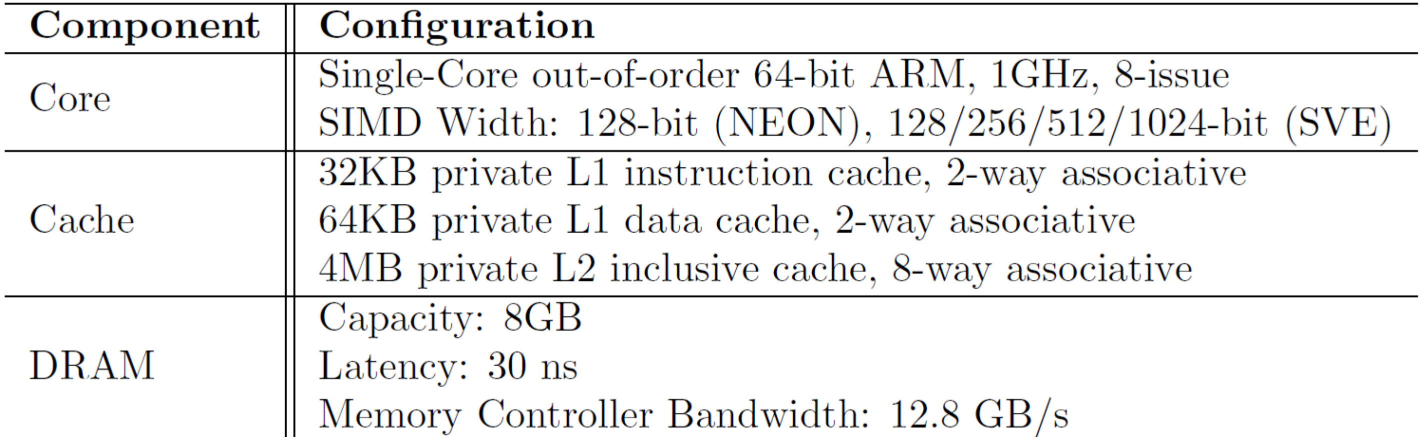
[2] *Wozniak* et al

Experimental Evaluation:

Smith-Waterman on gem5 w/ SVE

Experimental Setup

Gem5 Simulator w/ ARM SVE Simulation



Component	Configuration
Core	Single-Core out-of-order 64-bit ARM, 1GHz, 8-issue SIMD Width: 128-bit (NEON), 128/256/512/1024-bit (SVE)
Cache	32KB private L1 instruction cache, 2-way associative 64KB private L1 data cache, 2-way associative 4MB private L2 inclusive cache, 8-way associative
DRAM	Capacity: 8GB Latency: 30 ns Memory Controller Bandwidth: 12.8 GB/s

Experimental Setup

Application:

Smith-Waterman – Batch, Sliced, and Wavefront

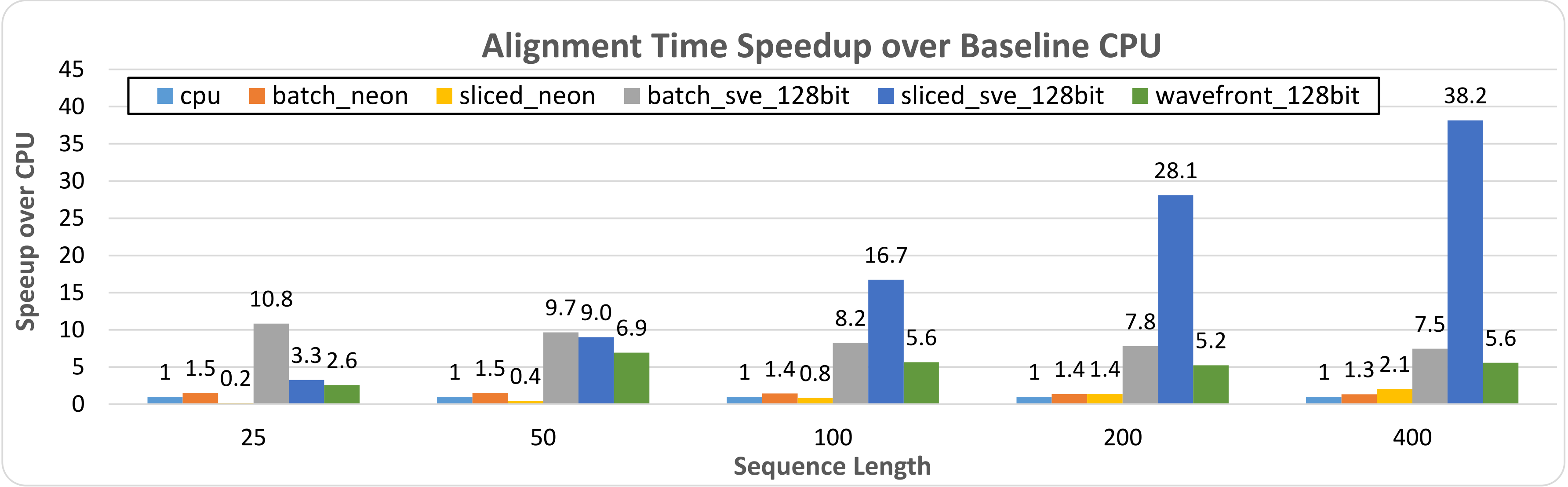
- Reference :

  25-400 bps samples from *E. Coli 536* Gene (4.9 Mbps)
- Query :

  1000 x 25-400 bps samples through WGSim

Advantage of SVE over Traditional System

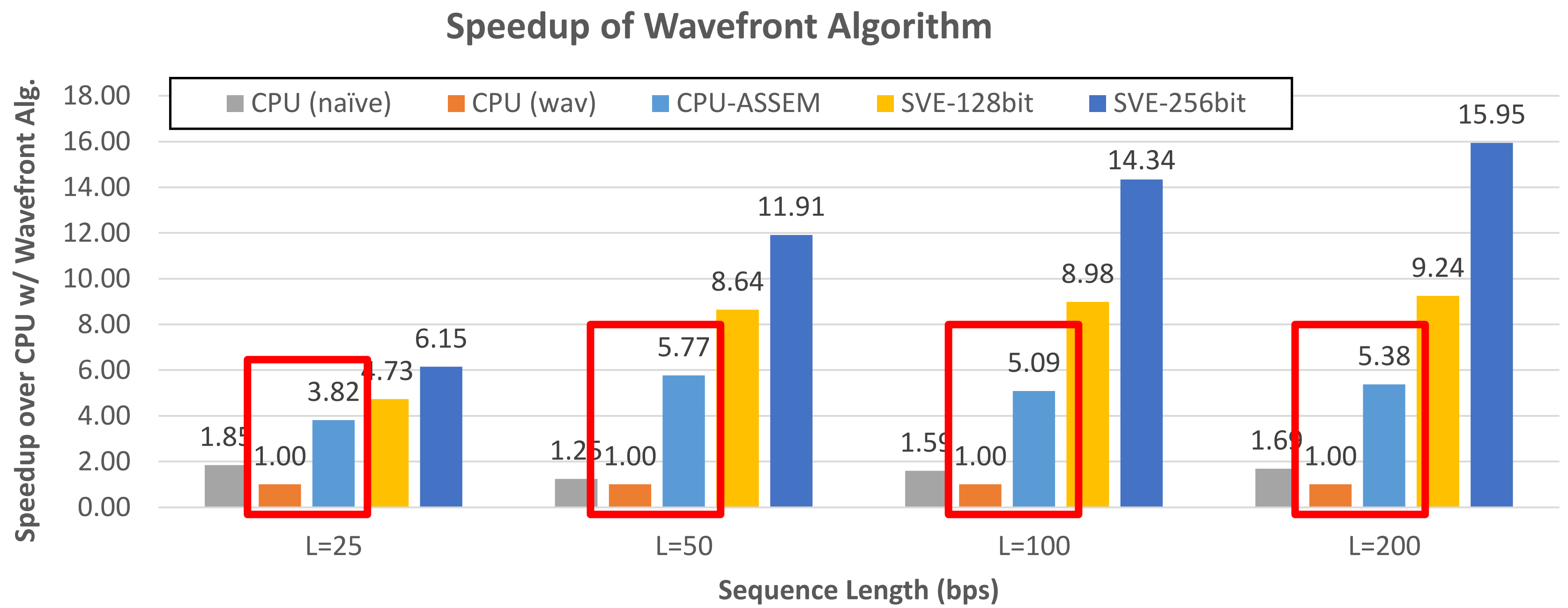
- CPU, NEON implementation written in C. SVE hand-written in assembly.
- SVE outperforms both CPU and NEON implementations by at least 3x
- Batch, Sliced and Wavefront used 32-bit, 16-bit and 64-bit vectors respectively.



Sequence Length	cpu	batch_neon	sliced_neon	batch_sve_128bit	sliced_sve_128bit	wavefront_128bit
25	1	1.5	0.2	10.8	3.3	2.6
50	1	1.5	0.4	9.7	9.0	6.9
100	1	1.4	0.8	8.2	16.7	5.6
200	1	1.4	1.4	7.8	28.1	5.2
400	1	1.3	2.1	7.5	38.2	5.6

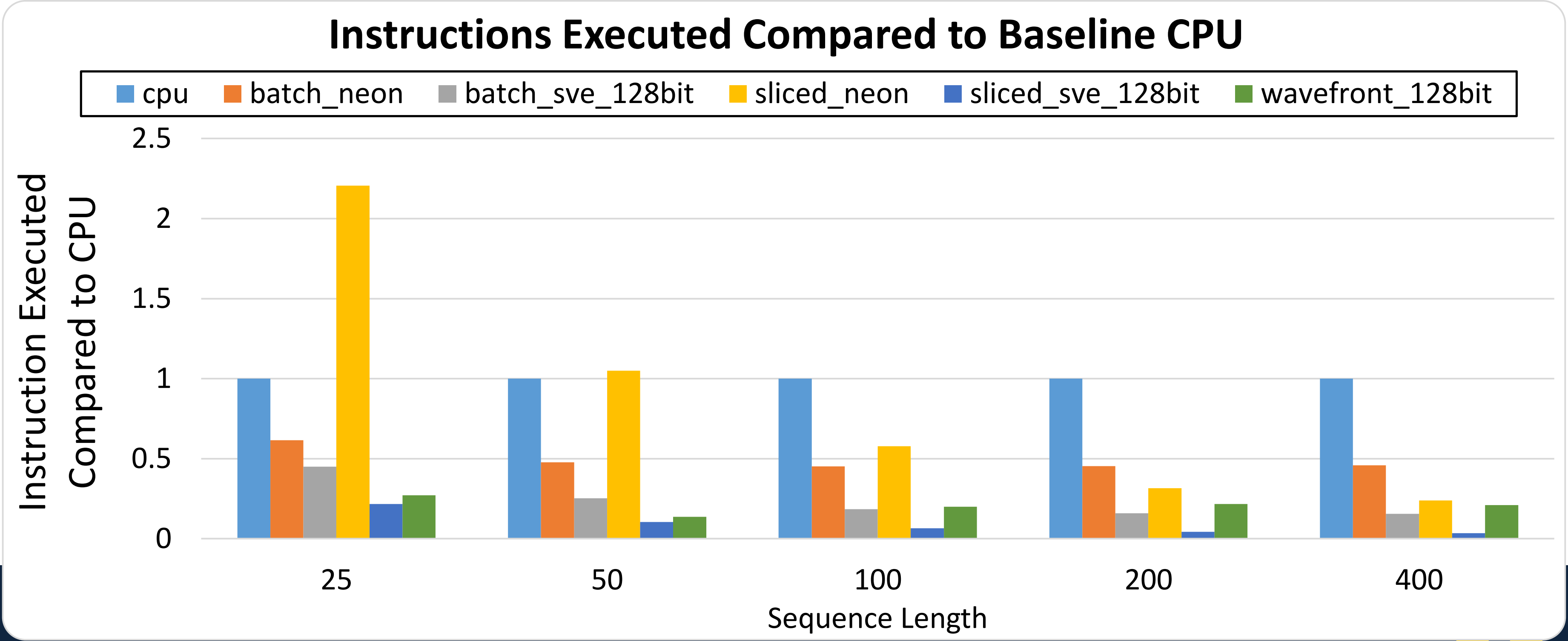
Impact of Handwritten Assembly

- Hand-written assembly code of Wavefront Algorithm has 4-6x speedup over C code.



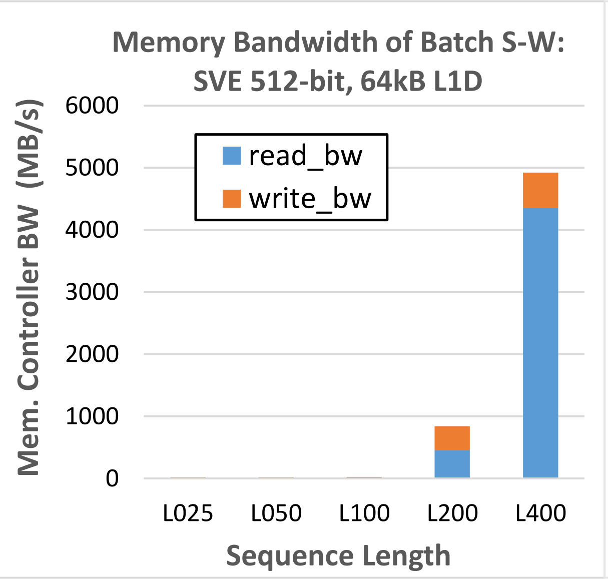
Advantage of SVE over Traditional System

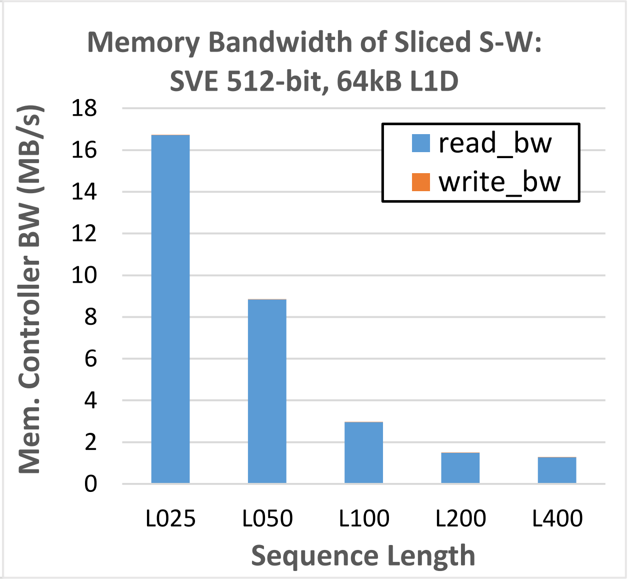
- SVE reduces the instruction execution significantly compared to CPU or NEON

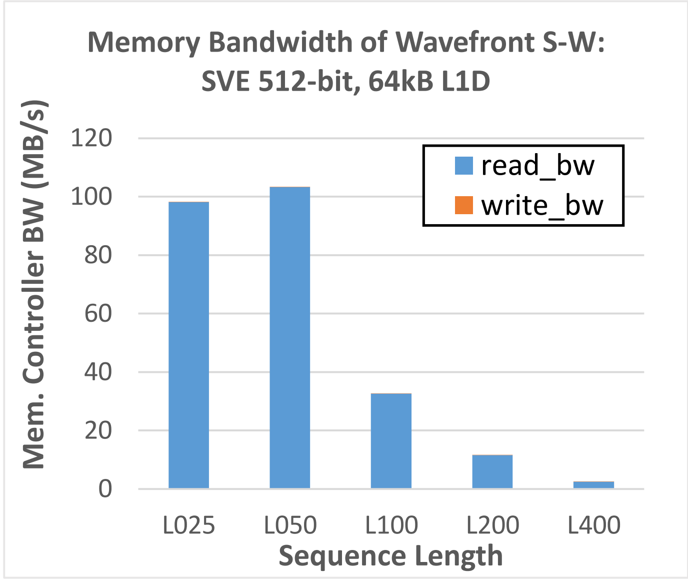


Memory Bandwidth Comparison

- Sliced and Wavefront significantly reduce the memory bandwidth compared to the Batch algorithm

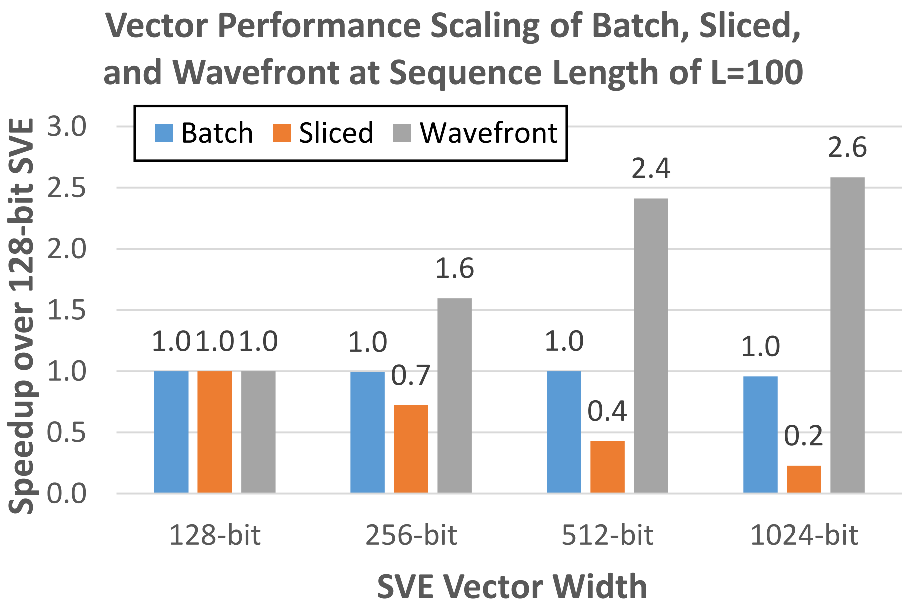


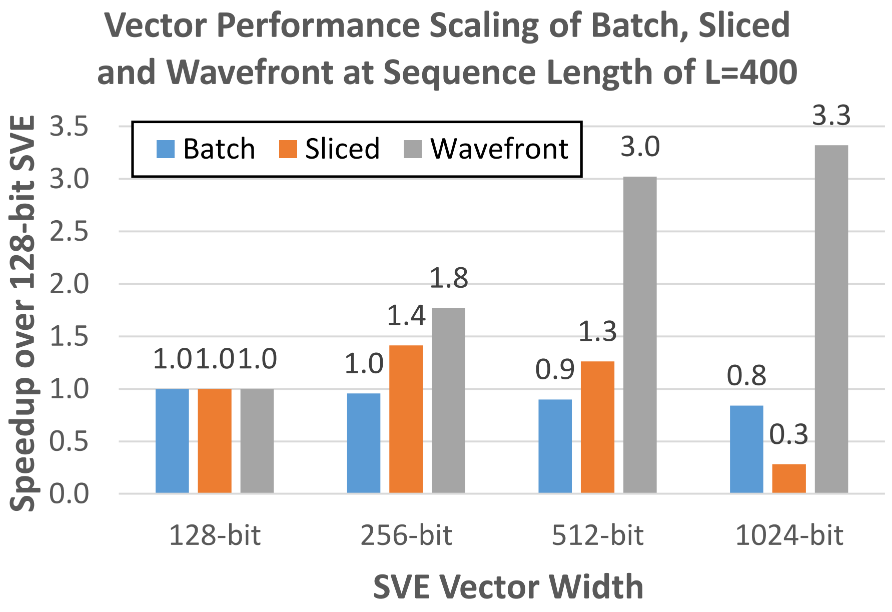




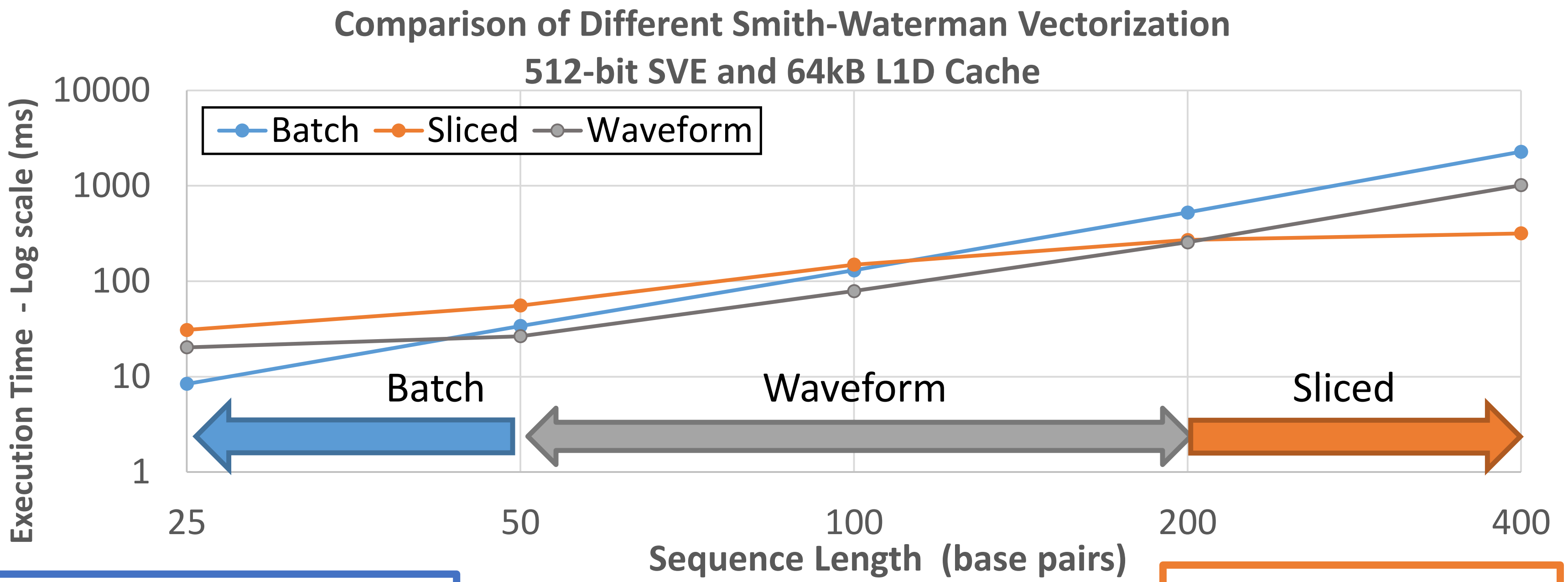
Vector Scaling of Different Algorithms

- Batch and Sliced show marginal improvement with increasing vector length
- Difficult to keep up with increased memory demand
- Need to resolve dependencies.





Fixed HW Options: Batch vs Sliced vs Waveform @512-bit



Batch
- Low overhead.
- Poor scaling.
- Fastest for short seq.

Waveform
- Efficient use of vector Lanes
- Fastest for medium seq.

Sliced
- High overhead.
- Execution bypassing
- Fastest for long seq.

HW with Variable Vector Length

- Given freedom to choose the hardware for each sequence length, we can establish a set of optimal algorithm-hardware pair.

Read Length	Algorithm	Vector Length	Speedup Over 512-bit Wavefront
< 50 bps	Batch	128-bit	2.77
50-100 bps	Wavefront	1024-bit	1.03
100-400 bps	Sliced	256-bit	1.23-3.06

Conclusion

Smith-Waterman on SVE:

+ Select Optimal Vector Length & Algorithm depending on Input
+ Lower Instruction Footprint

- Improvements to memory controller can lead to improved performance
- Wavefront algorithm use 64-bit vectors due to limitations on gather-scatter instruction addressing.

Key References

[1] Smith TF, Waterman MS, "*Identification of common molecular subsequences*" J Mol Biol 147

[2] Wozniak A. "*Using video-oriented instructions to speed up sequence comparison*" Comput Appl Biosci. 1997

[3] Farrar M, "*Striped Smith-Waterman speeds database searches six times over other SIMD implementations*" Bioinformatics, Vol 23, Issue 2, 15 January 2007

[4] Rognes T, "*Faster Smith-Waterman database searches with inter-sequence SIMD parallelization*" Bioinformatics 2011

[5] Zhao M, Lee W, Garrison E., Marth G. "*SSW Library: An SIMD Smith-Waterman C/C++ Library for Use in Genomic Applications*"

[6] Li H, Durbin R. "*Fast and accurate short read alignment with Burrows-Wheeler transform*" Bioinformatics 25

[7] Steinfadt S. "*SWAMPT+: Enhanced Smith-Waterman Search for Parallel Models*"

Questions?