Scrooge

A Fast and Memory-Frugal Genomic Sequence Aligner for CPUs, GPUs, and ASICs

Joël Lindegger

Damla Senol Cali, Mohammed Alser, Juan Gómez-Luna,
Nika Mansouri Ghiasi, Onur Mutlu

April 14th 2023
BIO-Arch

ETH zürich

SAFARI
SAFARI Research Group

Efficient Pairwise Alignment is Needed

- **Pairwise sequence alignment** is a **recurring kernel** in common genomics workloads, including **read mapping** and ***de novo* assembly**

- **Pairwise sequence alignment** is often the **bottleneck** in these applications

GenASM [Senol Cali+]

- GenASM is a pairwise sequence alignment algorithm proposed in prior work [Senol Cali+]

- GenASM builds a dynamic programming (DP) table of bitvectors, followed by a traceback operation

Text	A	C	G	T	-
Exact Match	1111	1111	1111	1111	1111
1 Edit	0110	0010	1000	1100	1110
2 Edits	0000	0000	1000	1100	1100
3 Edits	0000	0000	0000	1000	1000
4 Edits	0000	0000	0000	0000	0000

[Senol Cali+], "GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis", MICRO 2020

GenASM [Senol Cali+]

- GenASM is a pairwise sequence alignment algorithm proposed in prior work [Senol Cali+]

- GenASM builds a dynamic programming (DP) table of bitvectors, followed by a traceback operation

Text	A	C	G	T	-
Exact Match	1111	1111	1111	1111	1111
1 Edit	0	0	0	0	1110
2 Edits	0000	0000	1000	1100	1100
3 Edits	0000	0000	0000	1000	1000
4 Edits	0000	0000	0000	0000	0000

Algorithm 1 GenASM-DC Algorithm

Inputs: text, pattern, k
Outputs: editDist

```
1:  n ← LENGTH(text)
2:  m ← LENGTH(pattern)
3:  PM ← BUILDPATTERNMASKS(pattern)
4:
5:  R[n][d] ← 11...1 ≪ d            ▷ Initialize for all 0 ≤ d ≤ k
6:
7:  for i in (n − 1) : −1 : 0 do
8:      char ← text[i]
9:      curPM ← PM[char]
10:
11:     R[i][0] ← (R[i + 1][0] ≪ 1) | curPM          ▷ exact match
12:     for d in 1 : k do
13:         I ← R[i][d − 1] ≪ 1                      ▷ insertion
14:         D ← R[i + 1][d − 1]                      ▷ deletion
15:         S ← R[i + 1][d − 1] ≪ 1                  ▷ substitution
16:         M ← (R[i + 1][d] ≪ 1) | curPM            ▷ match
17:         R[i][d] ← I & D & S & M
18:
19: editDist ← arg min_d {MSB(R[0][d]) = 0}
```

Only Bitwise Operations

[Senol Cali+], "GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis", MICRO 2020

Our Goals

Build a practical and efficient implementation of the GenASM algorithm for multiple computing platforms

Compete with state-of-the-art pairwise sequence aligners like Edlib, KSW2, and BiWFA

Scrooge

Three novel algorithmic improvements
which address inefficiencies in the GenASM algorithm

Efficient open-source implementations
for CPUs and GPUs

Key Results

Scrooge consistently outperforms GenASM
- 2.1x speedup over GenASM on CPU
- 5.9x speedup over GenASM on GPU
- 3.6x better area efficiency than GenASM as an ASIC

Scrooge consistently outperforms state-of-the-art CPU and GPU baselines, including KSW2, Edlib, and BiWFA

SAFARI

Outline

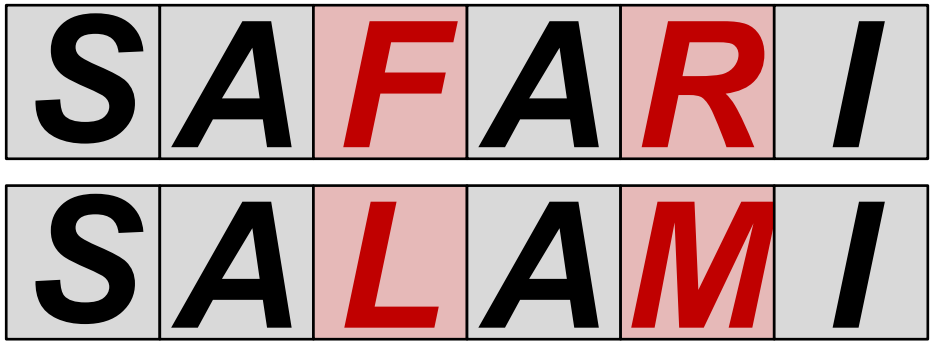
Pairwise Sequence Alignment (PSA)

- Compare a pair of strings
- while allowing substitutions, insertions, and deletions

S	A	F	A	R	I
S	A	L	A	M	I

Pairwise Sequence Alignment (PSA)

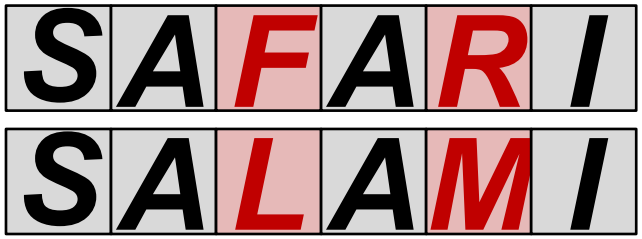
- Compare a pair of strings
- while allowing **substitutions**, insertions, and deletions

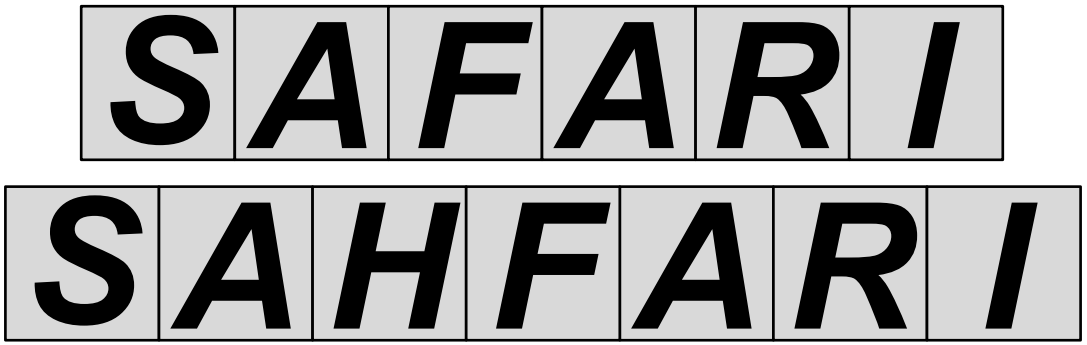


Pairwise Sequence Alignment (PSA)

- Compare a pair of strings
- while allowing substitutions, insertions, and deletions

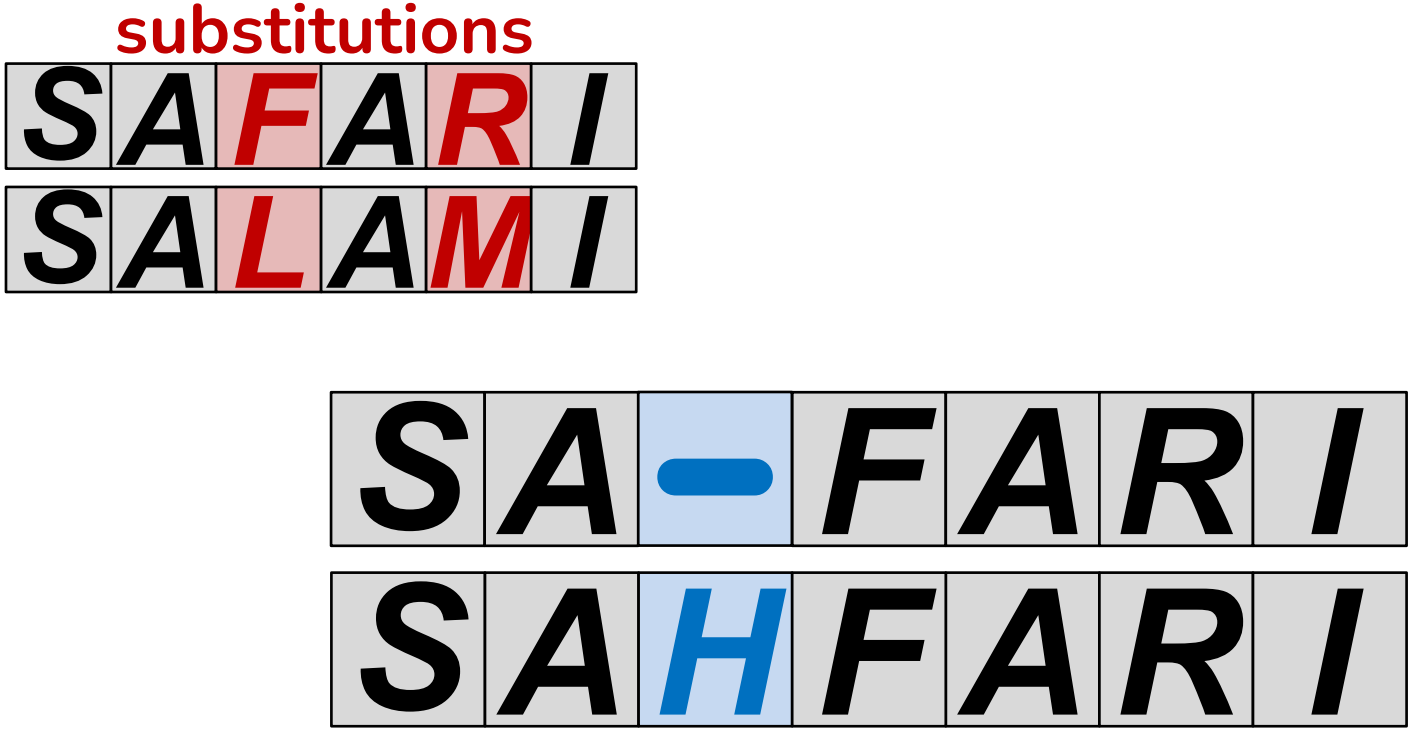
substitutions





Pairwise Sequence Alignment (PSA)

- Compare a pair of strings
- while allowing substitutions, insertions, and deletions



Pairwise Sequence Alignment (PSA)

- Compare a pair of strings
- while allowing substitutions, insertions, and deletions

substitutions

S	A	F	A	R	I
S	A	L	A	M	I

insertions

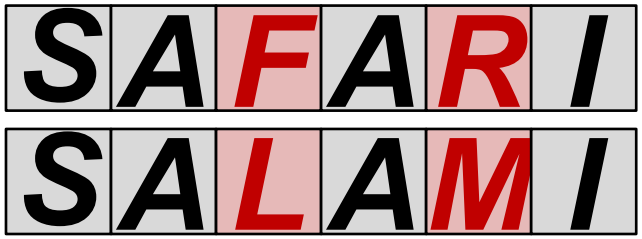
S	A	–	F	A	R	I
S	A	H	F	A	R	I

S	A	F	A	R	I
S	A	A	R	I	

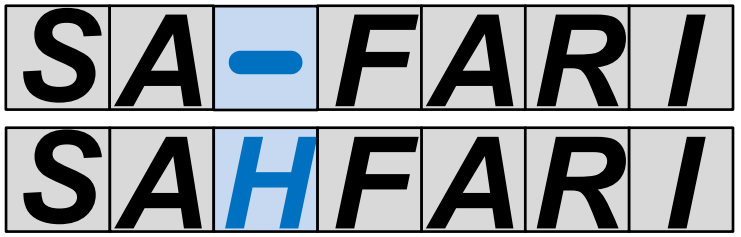
Pairwise Sequence Alignment (PSA)

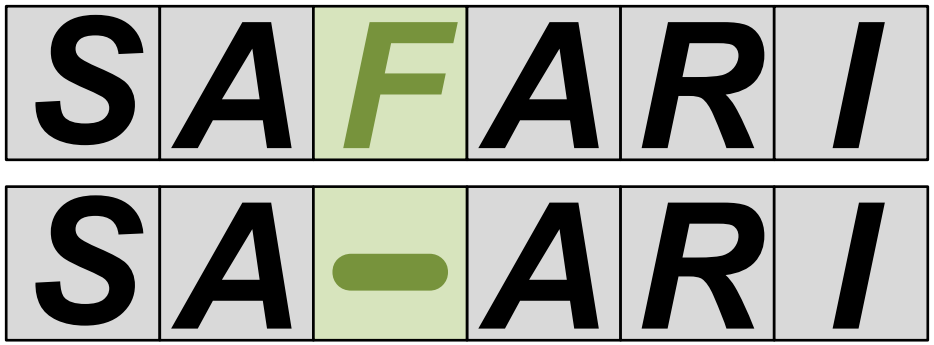
- Compare a pair of strings
- while allowing substitutions, insertions, and deletions

substitutions



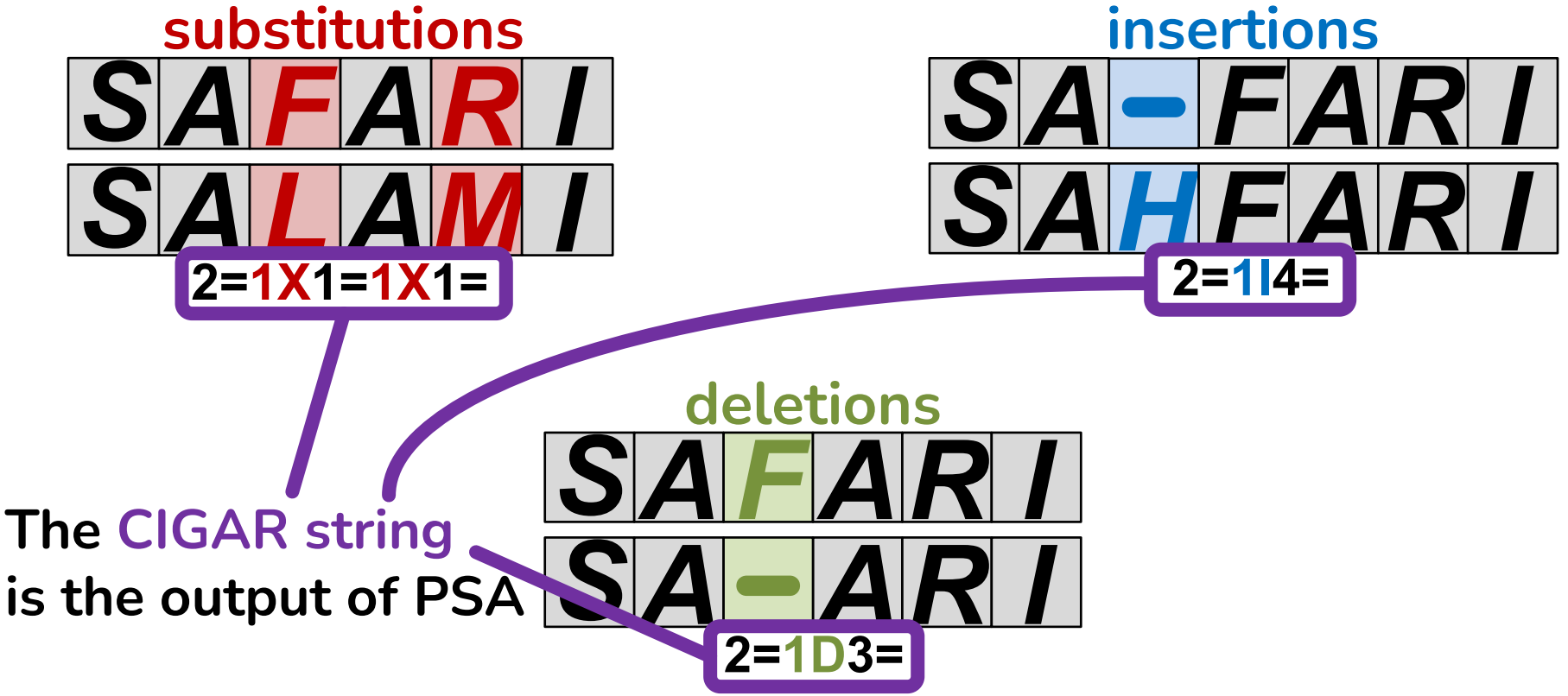
insertions



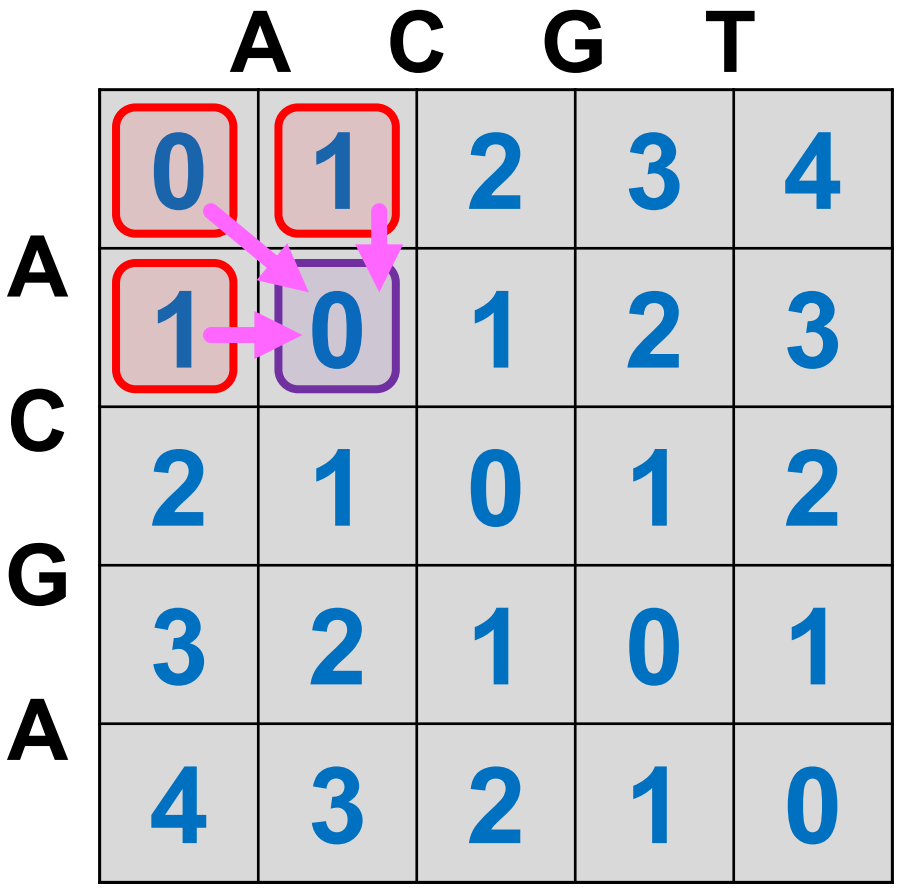


Pairwise Sequence Alignment (PSA)

- Compare a pair of strings
- while allowing substitutions, insertions, and deletions
- The total number of edits should be minimal



Arithmetic Dynamic Programming for PSA



Needleman-Wunsch Smith-Waterman-Gotoh, WFA, ...
Next entry is calculated from three neighbors
using arithmetic operations

The GenASM Algorithm

Text	A	C	G	T	-
Exact Match	1111	1111	1111	1111	1111
1 Edit	0110	1010	1100	1110	1110
2 Edits	0000	0000	1000	1100	1100
3 Edits	0000	0000	0000	1000	1000
4 Edits	0000	0000	0000	0000	0000

GenASM

Next entry is calculated from three neighbors using bitwise operations

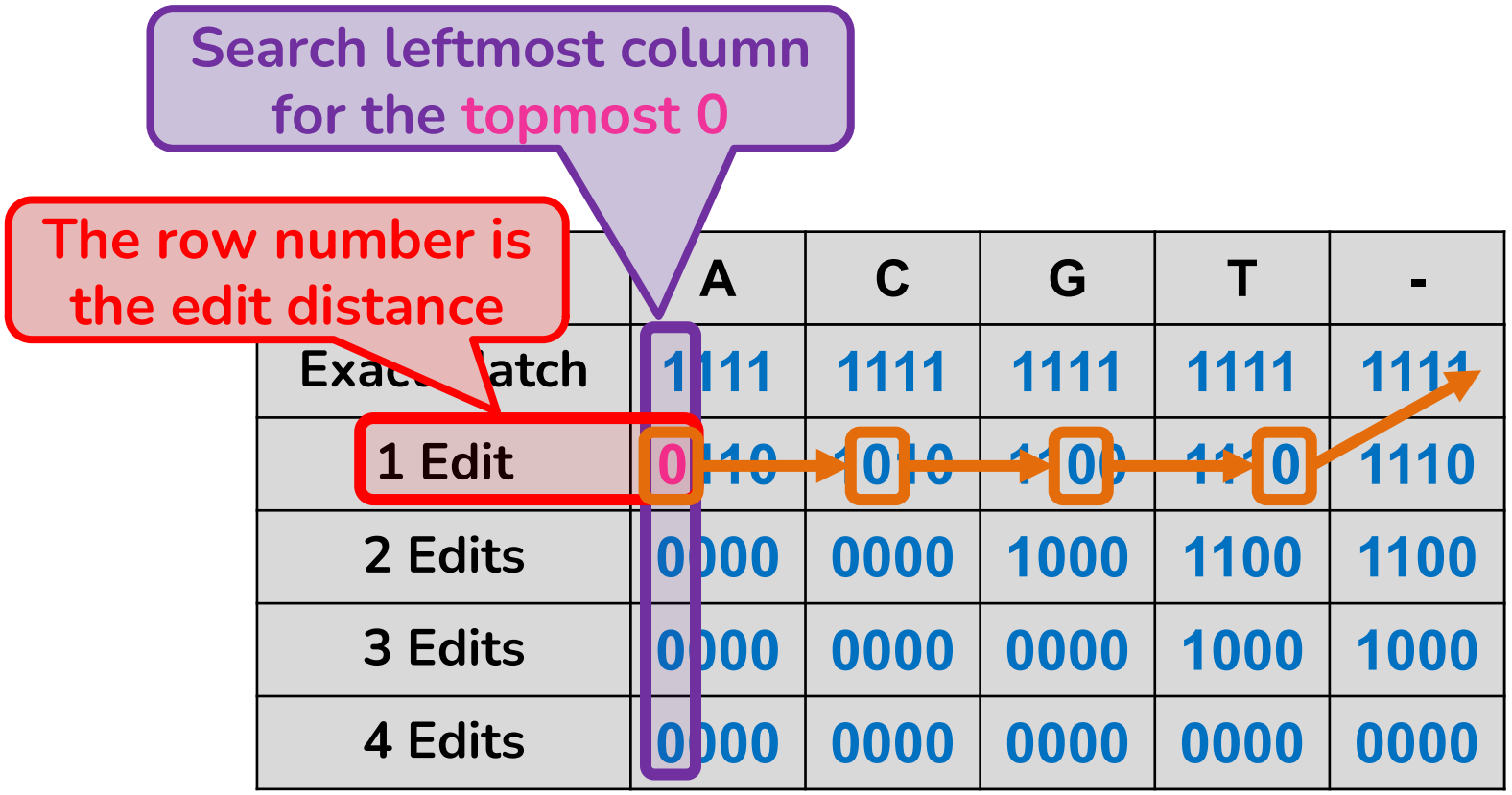
Particularly efficient in hardware

		A	C	G	T
		0	1	2	3	4
A		1	0	1	2	3
C		2	1	0	1	2
G		3	2	1	0	1
A		4	3	2	1	0

Needleman-Wunsch Smith-Waterman-Gotoh, WFA, ...

Next entry is calculated from three neighbors using arithmetic operations

The GenASM Algorithm (Traceback)



	A	C	G	T	-
Exact Match	1111	1111	1111	1111	1111
1 Edit	0110	0010	1000	1100	1110
2 Edits	0000	0000	1000	1100	1100
3 Edits	0000	0000	0000	1000	1000
4 Edits	0000	0000	0000	0000	0000

Traceback obtains the CIGAR string by backtracking the origin of the topmost 0 in the leftmost column.

Outline

Analysis of GenASM

ASIC [Senol Cali+]

Application **S**pecific **I**ntegrated **C**ircuit

CPU

GPU

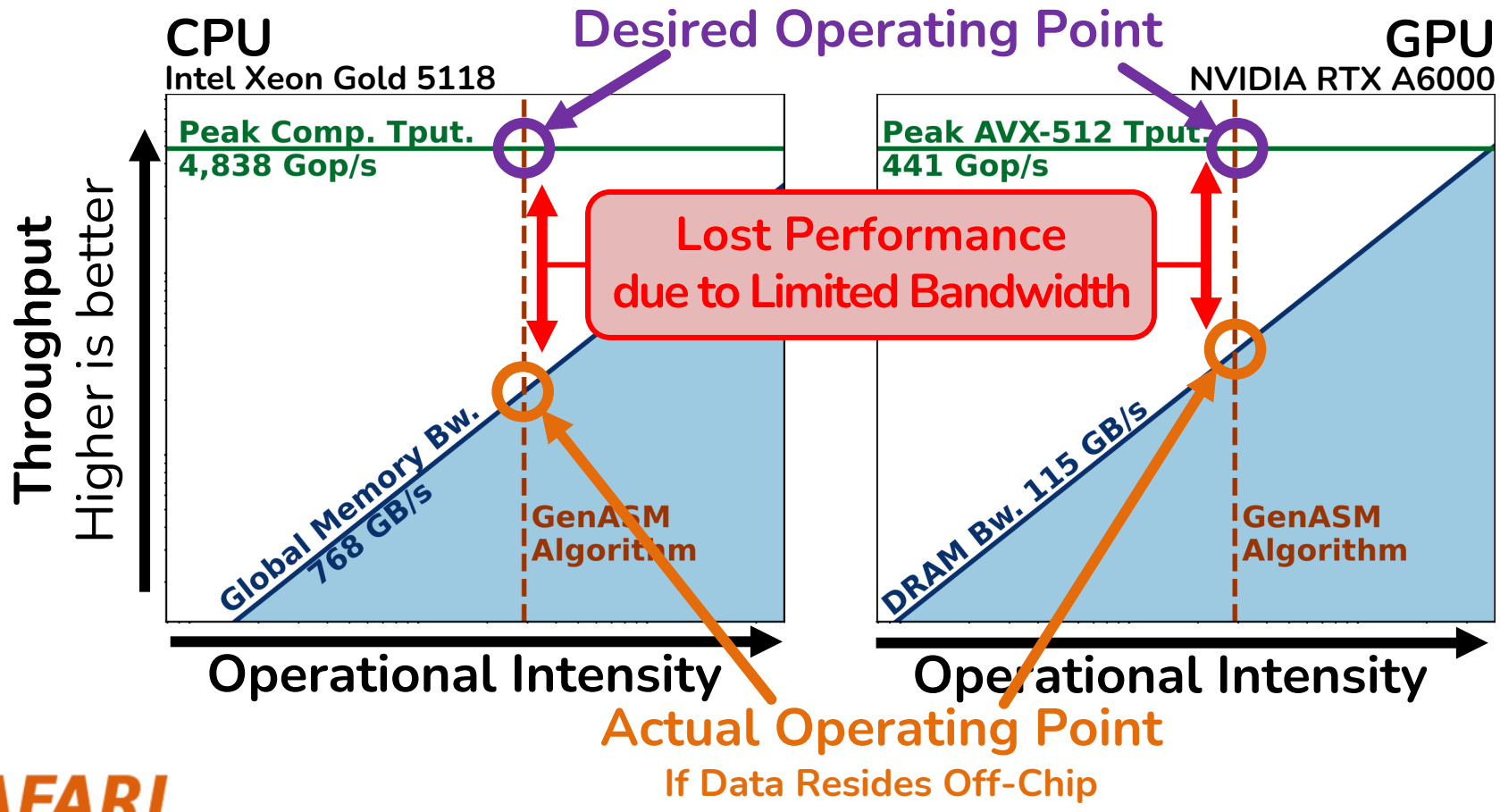
Can we do better?

Is GenASM suitable to commodity hardware?

[Senol Cali+], "GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis", MICRO 2020

Roofline Analysis of GenASM

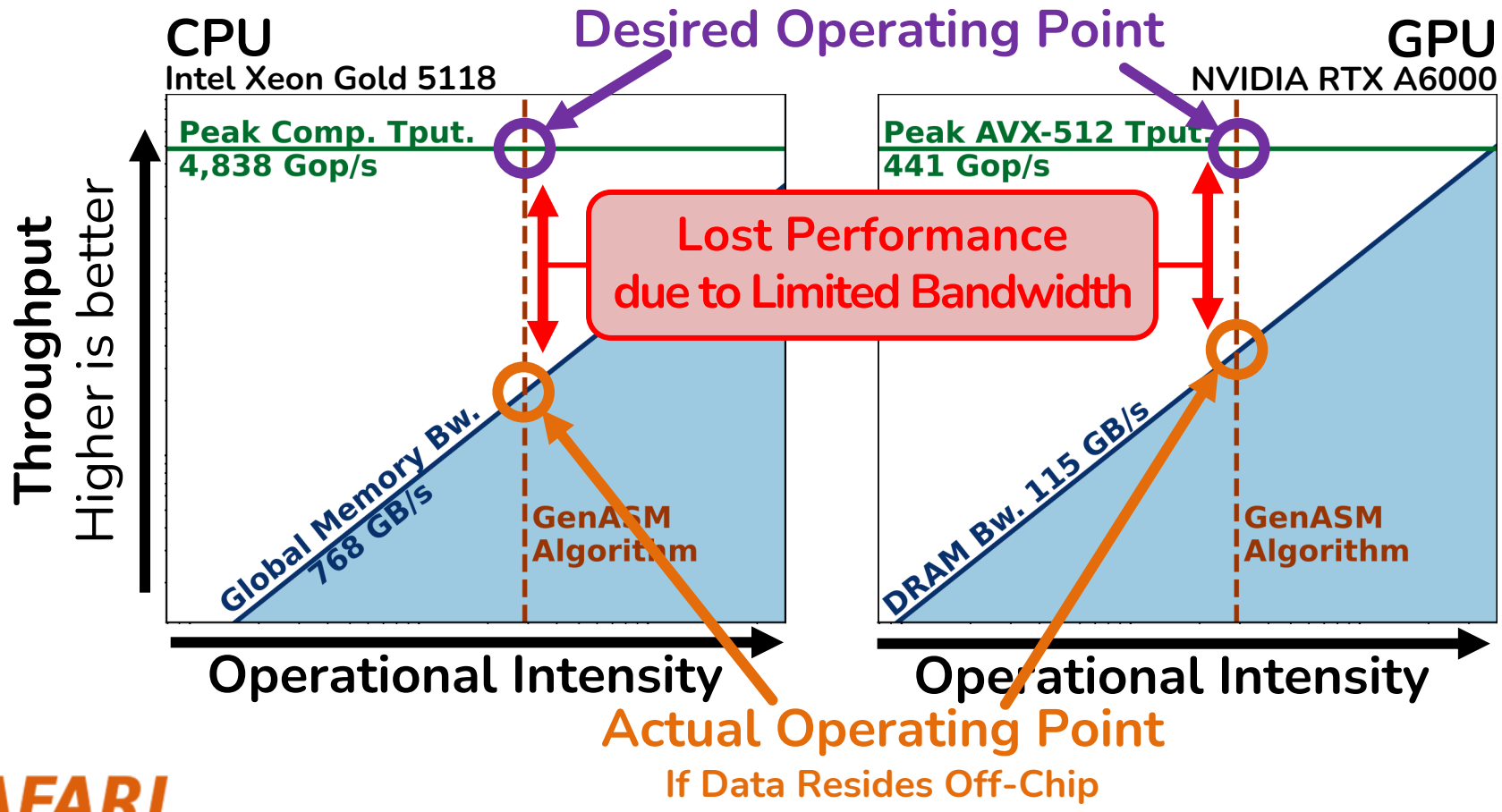
- Does commodity hardware have enough memory bandwidth for the GenASM algorithm?



Roofline Analysis of GenASM

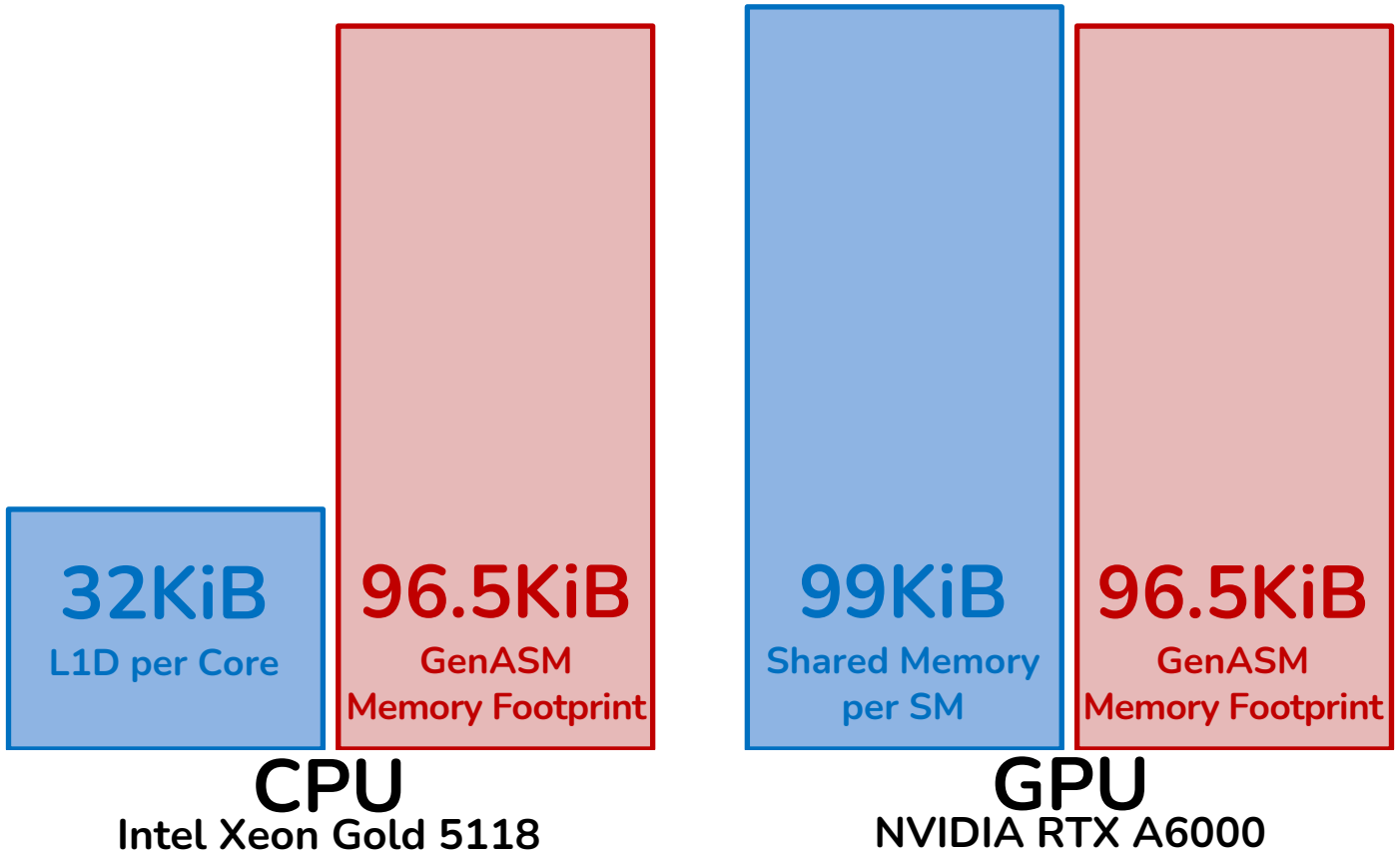
Inefficiency #1

GenASM cannot saturate commodity hardware with computation due to too much data movement



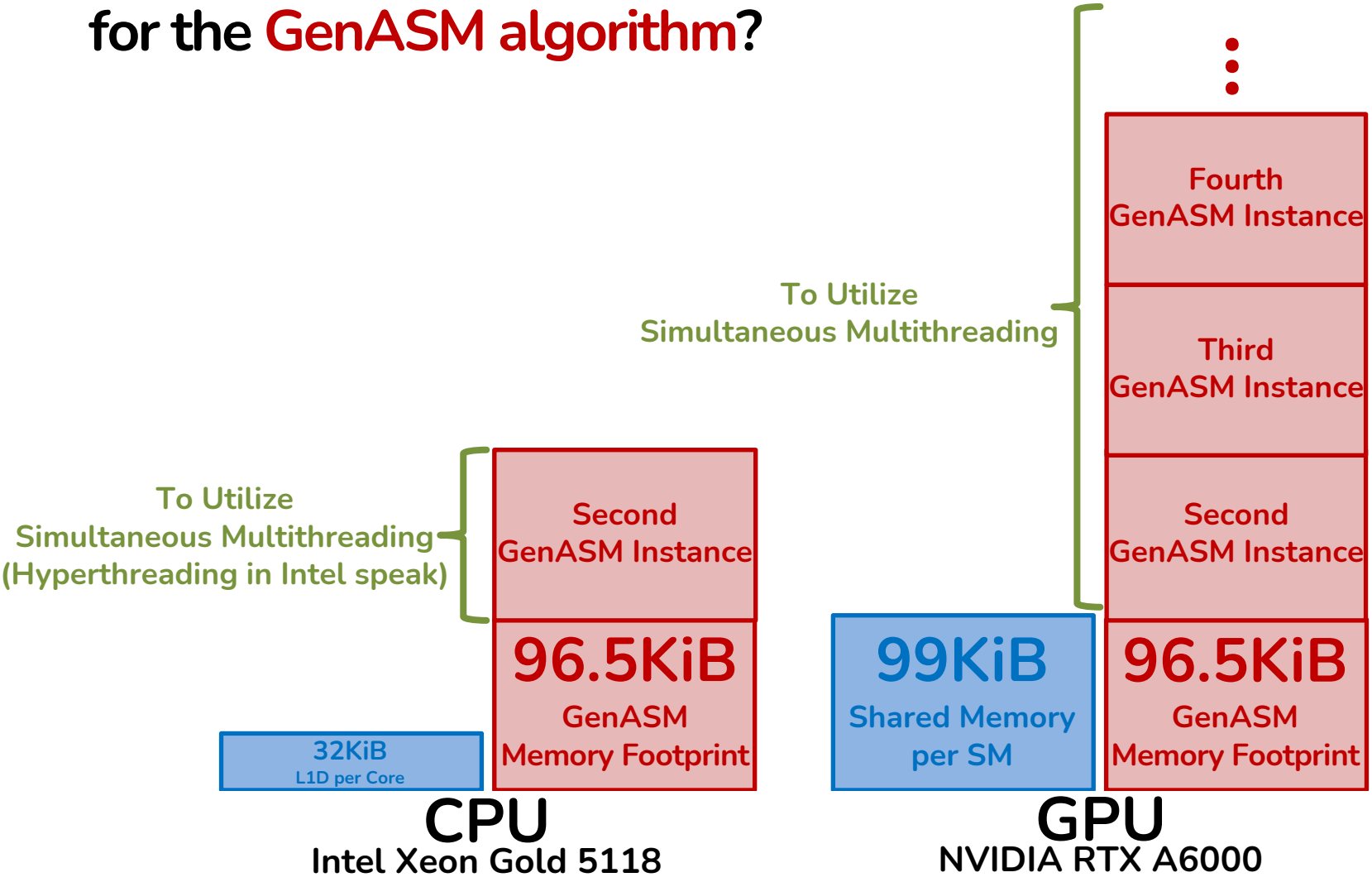
Memory Footprint Analysis of GenASM

- Does commodity hardware have enough on-chip memory for the GenASM algorithm?



Memory Footprint Analysis of GenASM

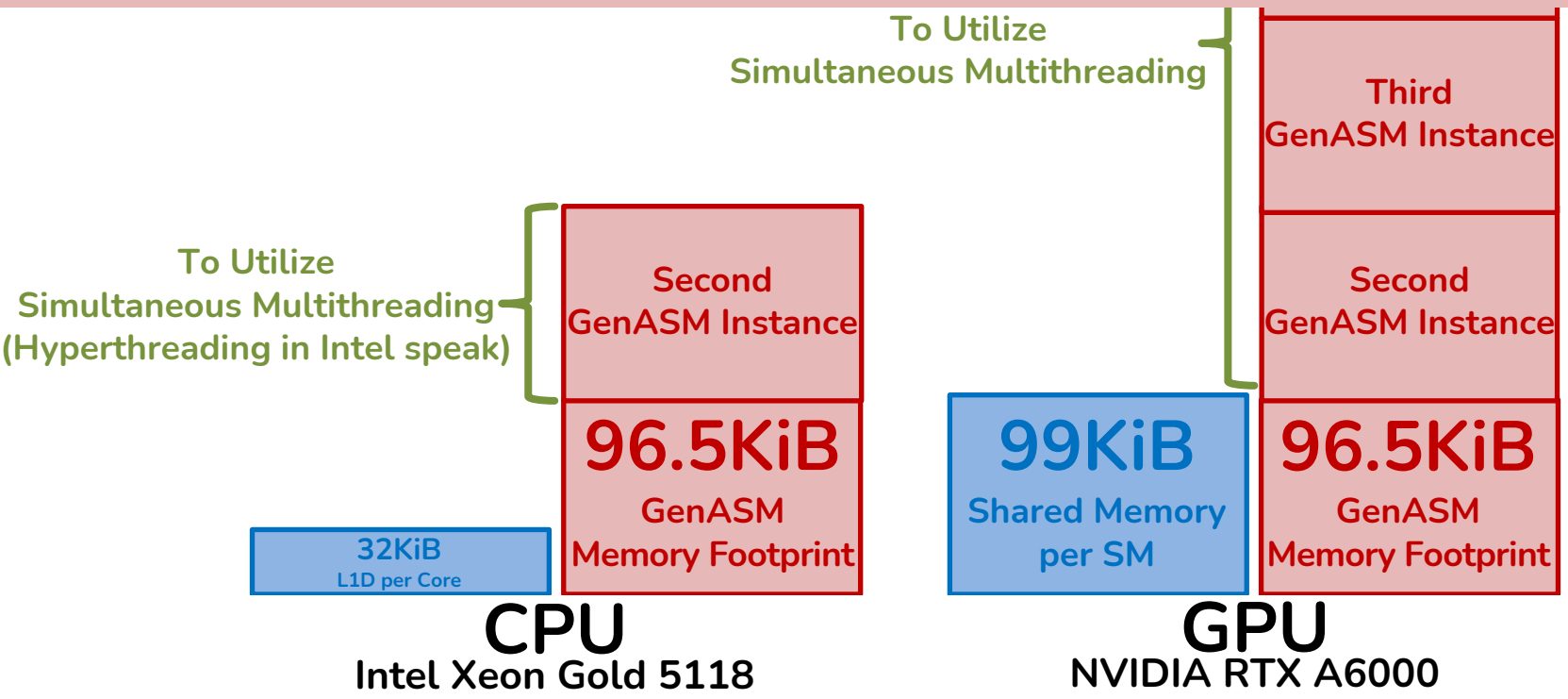
- Does commodity hardware have enough on-chip memory for the GenASM algorithm?



Memory Footprint Analysis of GenASM

Inefficiency #2

GenASM has a large memory footprint, especially when multiple instances are kept in memory for simultaneous multithreading



Unnecessary Work in GenASM

Text	A	C	G	T	-
Exact Match	1111	1111	1111	1111	1111
1 Edit	0110	0110	1100	1110	1110
2 Edits	0000	0000	1000	1100	1100
3 Edits	0000	0000	0000	1000	1000
4 Edits	0000	0000	0000	0000	0000

Cannot be Reached by Traceback

Inefficiency #3

GenASM does unnecessary work by computing DP cells which cannot be reached by Traceback

Inefficiencies in GenASM

1. Large memory bandwidth requirement

2. Large memory footprint

3. Unnecessary work

Outline

Scrooge Algorithm

Memory Improvements

reduce the memory footprint and data movement

SENE
Store Entries, not Edges

DENT
Discard Entries, not Used by Traceback

Efficiency Improvement

eliminates the unnecessary work

ET
Early Termination

Scrooge Algorithm

Memory Improvements

reduce the memory footprint and data movement

SENE
Store Entries, not Edges

DENT
Discard Entries, not Used by Traceback

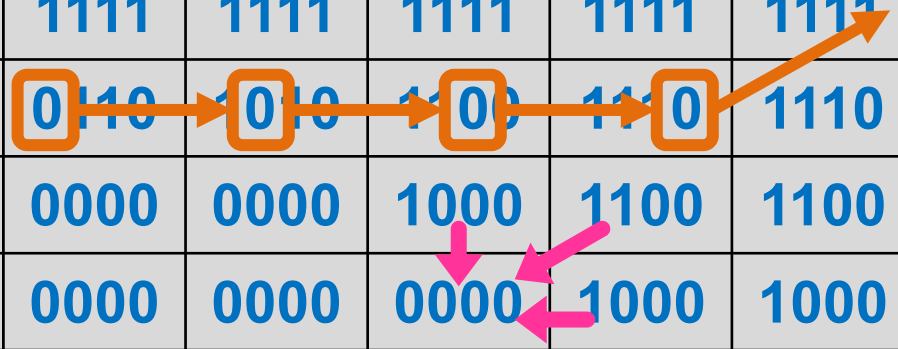
Efficiency Improvement

eliminates the unnecessary work

ET
Early Termination

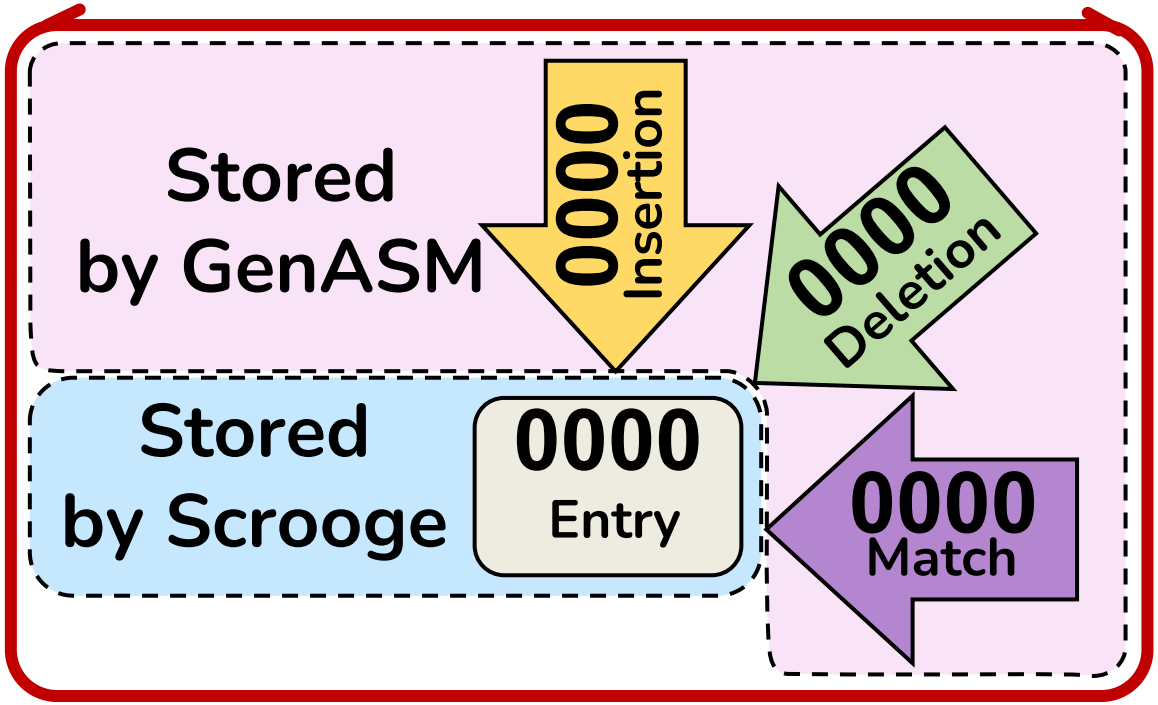
SENE: Store Entries, Not Edges

Text	A	C	G	T	-
Exact Match	1111	1111	1111	1111	1111
1 Edit	0110	0110	1100	1110	1110
2 Edits	0000	0000	1000	1100	1100
3 Edits	0000	0000	0000	1000	1000
4 Edits	0000	0000	0000	0000	0000



SENE: Store Entries, Not Edges

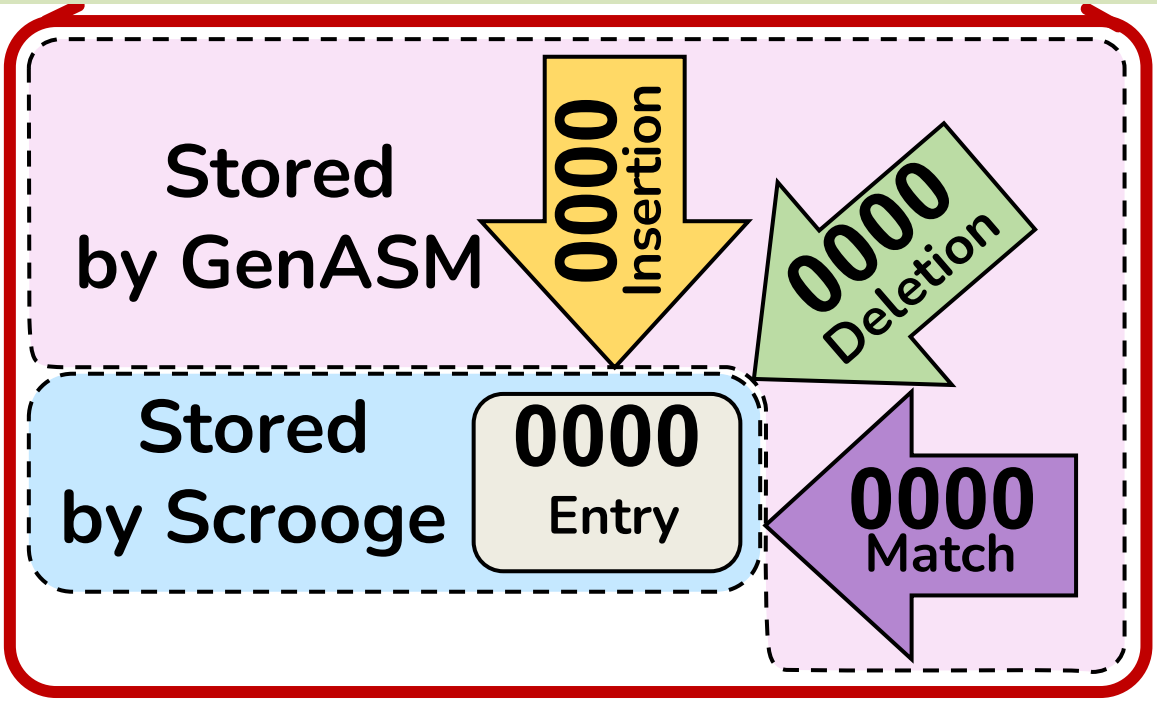
Text	A	C	G	T	-
Exact Match	1111	1111	1111	1111	1111
1 Edit	0	0	0	0	1110
2 Edits	0000	0000	1000	1100	1100
3 Edits	0000	0000	0000	1000	1000
4 Edits	0000	0000	0000	0000	0000



SENE: Store Entries, Not Edges

Text	A	C	G	T	-
Exact Match	1111	1111	1111	1111	1111
1 Edit	0	0	0	0	1110
2 Edits	0000	0000	1000	1100	1100

SENE results in a **3x reduction** in **memory footprint** and **data movement**



Scrooge Algorithm

Memory Improvements

reduce the memory footprint and data movement

SENE
Store Entries, not Edges

DENT
Discard Entries, not Used by Traceback

Efficiency Improvement

eliminates the unnecessary work

ET
Early Termination

DENT: Discard Entries Not Used by Traceback

Text	A	C	G	T	-
Exact Match	1111	1111	1111	1111	1111
1 Edit	0110	0110	1100	1110	1110
2 Edits	0000	0000	1000	1100	1100
3 Edits	0000	0000	0000	1000	1000
4 Edits	0000	0000	0000	0000	0000

Traceback is confined due to the "windowing heuristic"

Remaining bits need to be computed, but not stored

DENT results in a 4x reduction in memory footprint and data movement

Scrooge Algorithm

Memory Improvements

reduce the memory footprint and data movement

SENE
Store Entries, not Edges

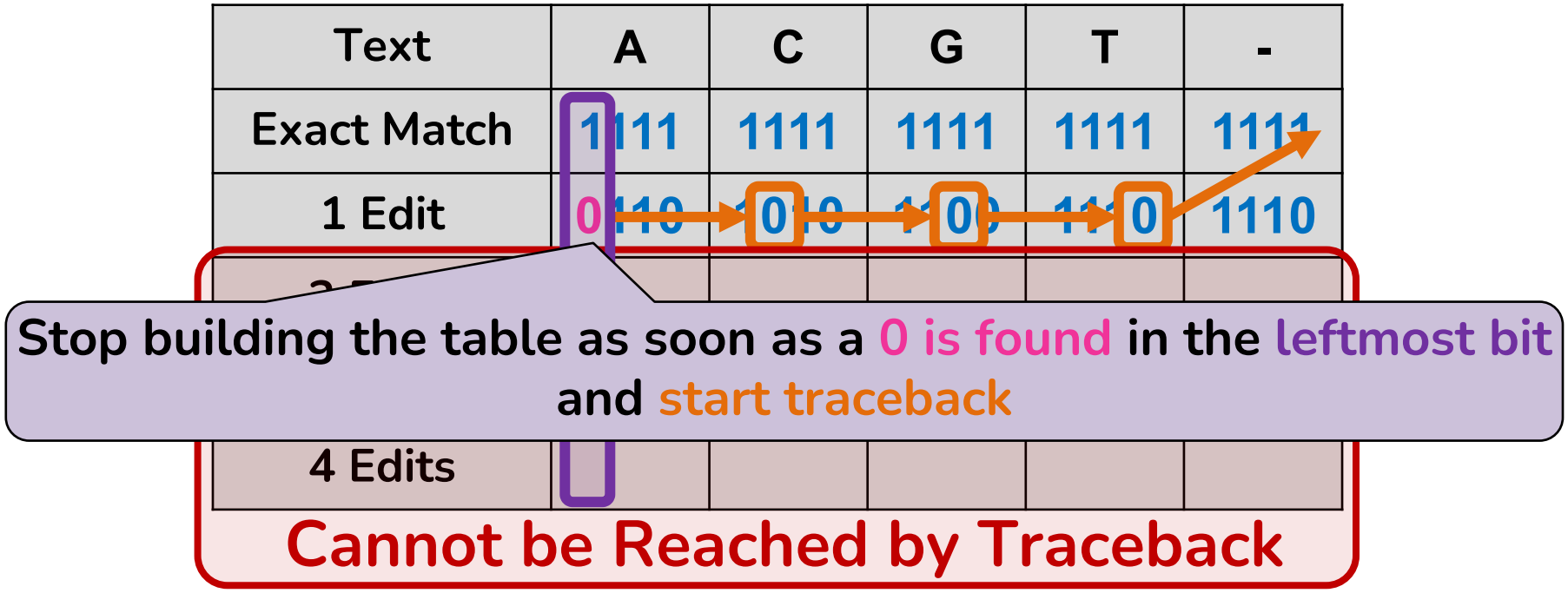
DENT
Discard Entries, not Used by Traceback

Efficiency Improvement

eliminates the unnecessary work

ET
Early Termination

ET: Early Termination



ET eliminates the unnecessary work

on average, at least 25% of cells are unnecessary

Outline

Scrooge CPU & GPU Implementations

- **We provide efficient open-source implementations of the Scrooge algorithm for CPUs and GPUs**
  - Easy-to-use library interface

- **CPU version**
  - C++
  - OpenMP for multithreading

- **GPU version**
  - C++
  - NVIDIA GPUs
    - CUDA 11.1
    - Compute capability 7.0+

Scrooge on GitHub

README.md

Scrooge: A fast and memory-frugal genomic sequence aligner for CPUs, GPUs and ASICs

Scrooge is a fast pairwise genomic sequence aligner. It efficiently aligns short and long genomic sequence pairs on multiple computing platforms. It is based on the GenASM algorithm (Senol Cali+, 2020), and adds multiple algorithmic improvements that significantly improve the throughput and resource efficiency for CPUs, GPUs and ASICs. For long reads, the CPU version of Scrooge achieves a 20.1x, 1.7x, and 2.1x speedup over KSW2, Edlib, and a CPU implementation of GenASM, respectively. The GPU version of Scrooge achieves a 4.0x 80.4x, 6.8x, 12.6x and 5.9x speedup over the CPU version of Scrooge, KSW2, Edlib, Darwin-GPU, and a GPU implementation of GenASM, respectively. We estimate an ASIC implementation of Scrooge to use 3.6x less chip area and 2.1x less power than a GenASM ASIC while maintaining the same throughput.

This repository contains Scrooge's CPU and GPU implementations, and several evaluation scripts. We describe Scrooge in our paper on arXiv and in Bioinformatics.

3 forks

Report repository

Releases

No releases published

Create a new release

Packages

No packages published

Publish your first package

Languages

C 33.5% C++ 32.4%

Python 18.1% Cuda 14.5%

Makefile 0.9% Shell 0.6%

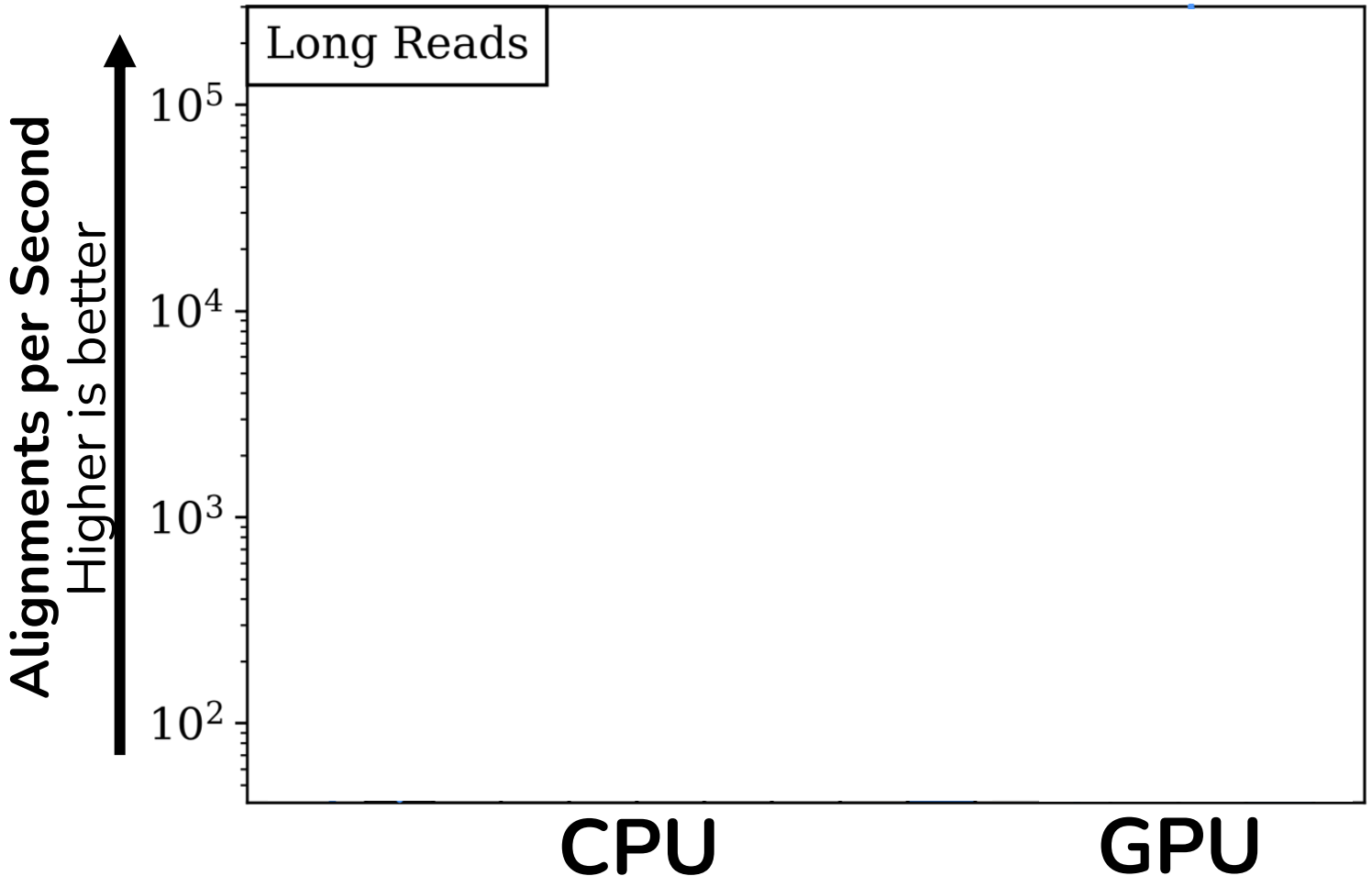
Scrooge on GitHub

Outline

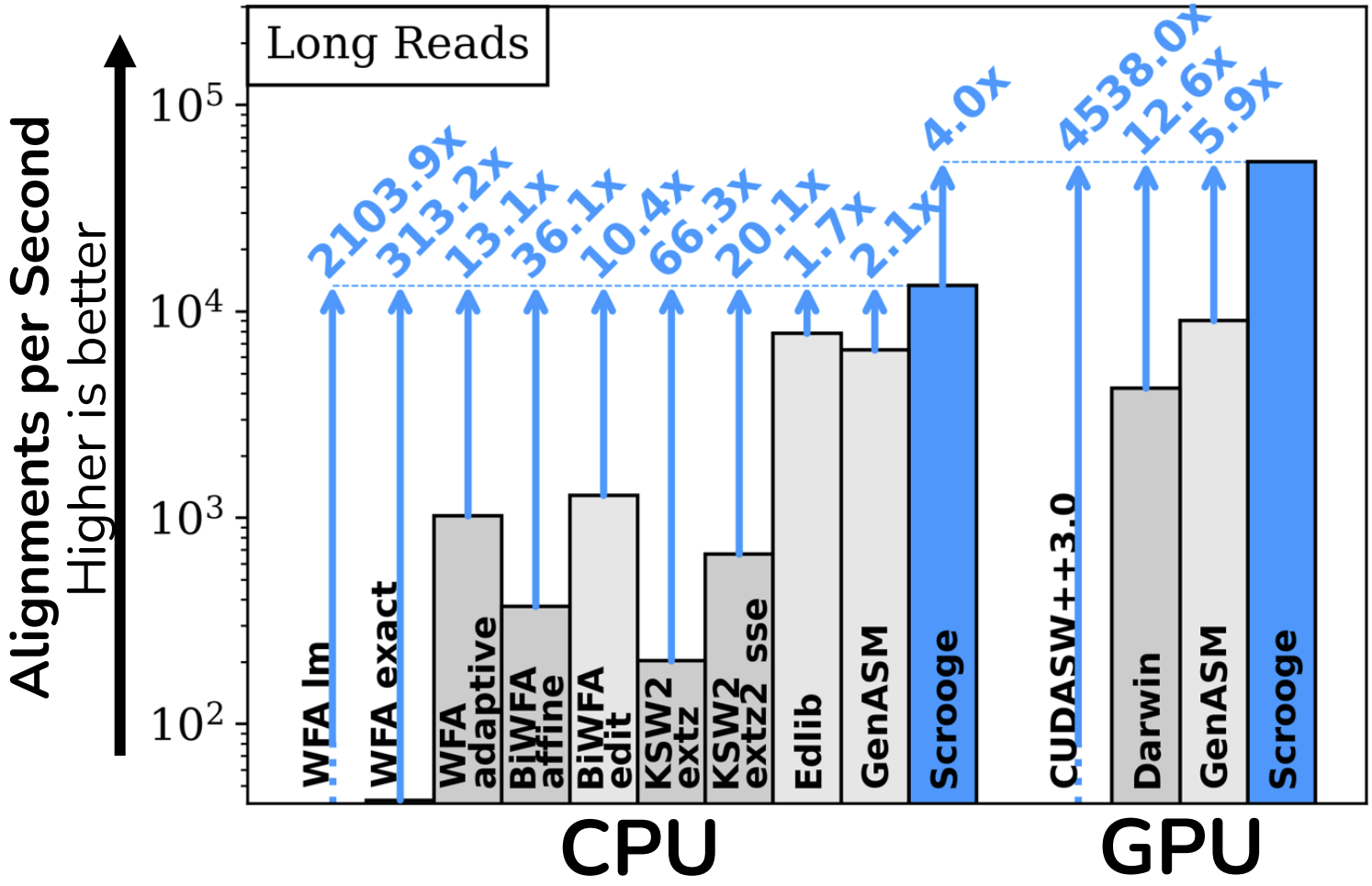
Methodology

- **Datasets**
  - Long reads
    - Simulated with PBSIM2 from the human reference genome GRCh38.p13
    - Chained with minimap2 to obtain 138,929 candidate pairs
  - Short reads
    - Illumina reads from SRR13278681
    - Chained with minimap2 to obtain 9,612,222 candidate pairs
- **CPU: dual-socket Intel Xeon Gold 5118**
  - 2× 12 physical cores, 2× 24 logical cores @ 3.2GHz
  - 196GiB DDR4 RAM
- **GPU: NVIDIA RTX A6000**
- **ASIC**
  - 28nm logic synthesis from [Senol Cali+]
  - SRAM numbers from CACTI 7

Long Read Throughput

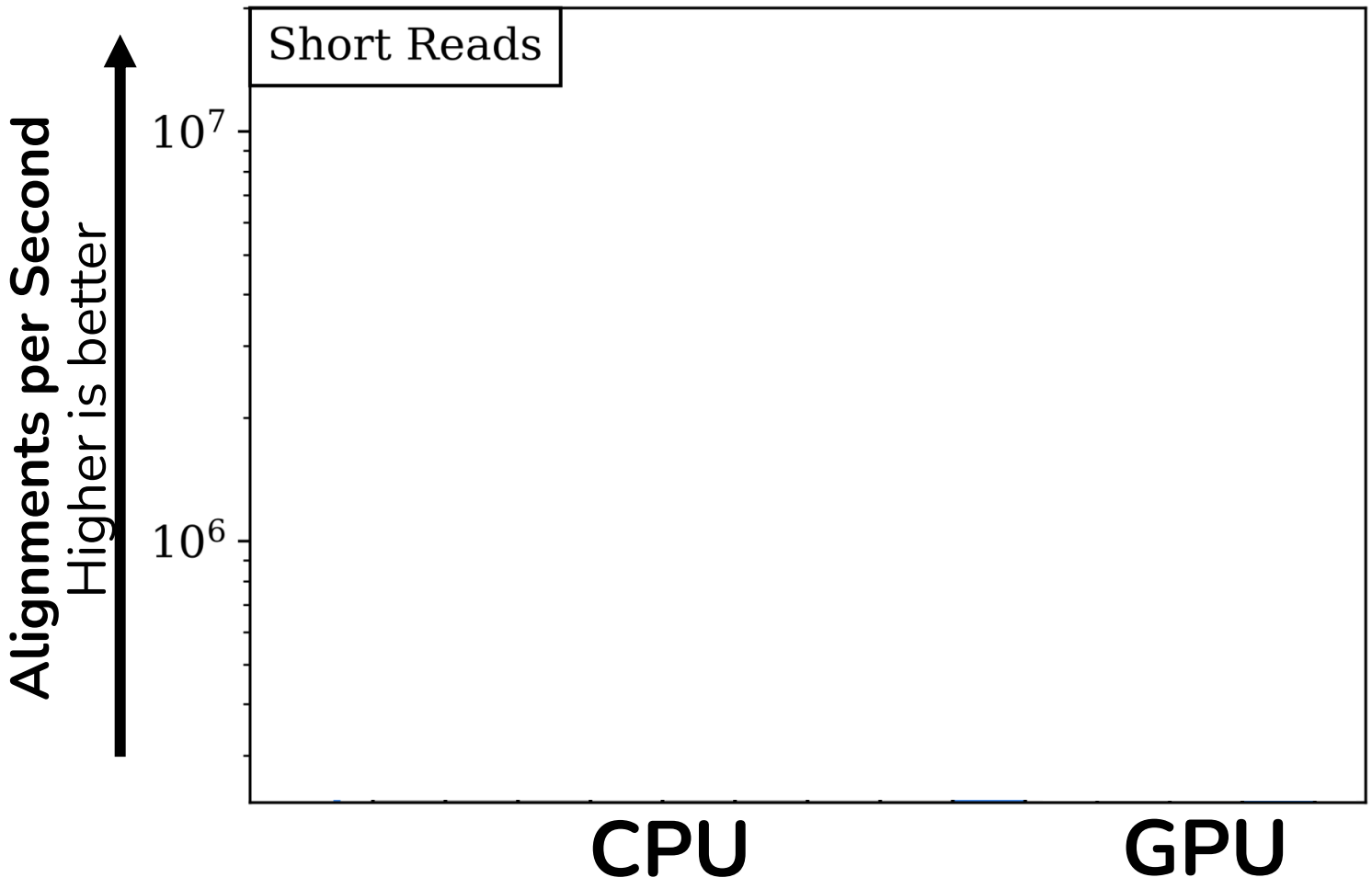


Long Read Throughput

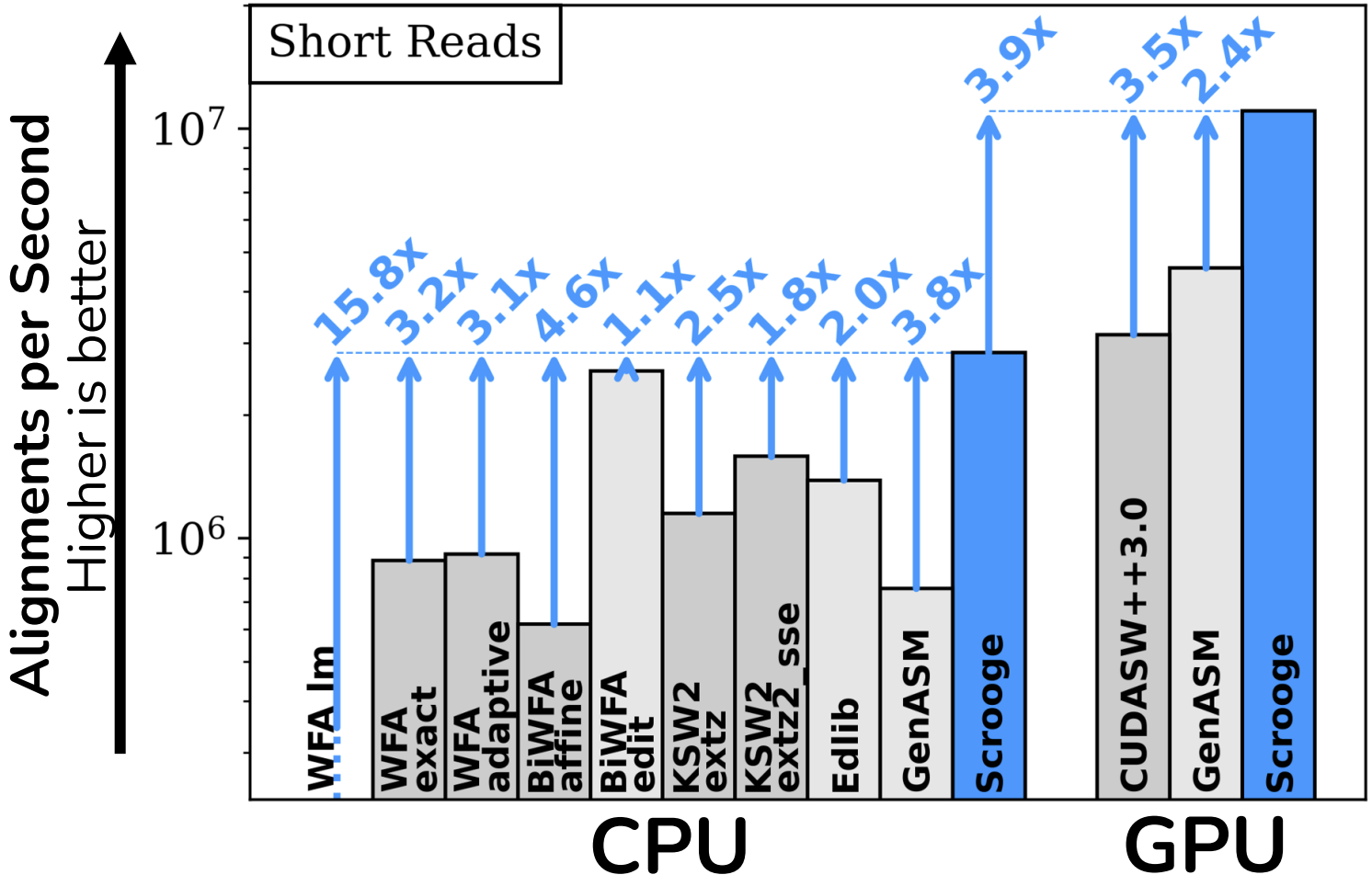


For long reads, Scrooge outperforms GenASM by 2.1x on CPU and 5.9x on GPU

Short Read Throughput



Short Read Throughput



For short reads, Scrooge outperforms GenASM by 3.8x on CPU and 2.4x on GPU

ASIC Results

Scrooge introduces
no significant computation overheads
over a GenASM ASIC

Scrooge's on-chip memory is much cheaper than GenASM's
due to the memory footprint and bandwidth reductions
(uses 18x less chip area and 18x less power)

Scrooge uses 3.6x less chip area
and 2.1x less power than a GenASM ASIC

More in the Paper: Evaluation

- Throughput sensitivity to each algorithmic improvement
- Thread scaling results
- Rigorous accuracy analysis
- Sensitivity analysis of throughput and accuracy
- ASIC breakdown

More in the Paper

Bioinformatics

ISCB INTERNATIONAL SOCIETY FOR COMPUTATIONAL BIOLOGY

Issues Advance articles Submit ▾ Alerts About ▾

Article Navigation

JOURNAL ARTICLE ACCEPTED MANUSCRIPT

Scrooge: A Fast and Memory-Frugal Genomic Sequence Aligner for CPUs, GPUs, and ASICs

Joël Lindegger ✉, Damla Senol Cali, Mohammed Alser, Juan Gómez-Luna, Nika Mansouri Ghiasi, Onur Mutlu ✉

Bioinformatics, btad151, https://doi.org/10.1093/bioinformatics/btad151

Published: 24 March 2023 **Article history** ▾

Scrooge in Bioinformatics

Scrooge on arXiv

Scrooge on GitHub

Outline

1 Background

2 Analysis of GenASM

3 Scrooge Algorithm

4 Scrooge Implementations

5 Evaluation

6 Conclusion

Conclusion

Motivation	**Pairwise sequence alignment (PSA)** is computationally costly and common step in bioinformatics pipelines. **GenASM** is a promising candidate for efficient PSA. For example, its ASIC implementation is up to **10,000x faster** than prior software aligners.
Goals	• Build a **practical** and **efficient implementation** of the GenASM algorithm for **multiple computing platforms** • **Compete** with **state-of-the-art pairwise sequence aligners** like Edlib, KSW2, and BiWFA
Scrooge	•**Three novel algorithmic improvements** address GenASM's inefficiencies •Efficient **open-source CPU** and **GPU** implementations
Key Results	Scrooge **consistently outperforms GenASM** • **2.1x speedup** over GenASM on CPU • **5.9x speedup** over GenASM on GPU • **3.6x** better **area efficiency** than GenASM on ASIC Scrooge **consistently outperforms state-of-the-art CPU and GPU baselines**, including KSW2, Edlib, and BiWFA

Scrooge

A Fast and Memory-Frugal Genomic Sequence Aligner for CPUs, GPUs, and ASICs

Joël Lindegger

Damla Senol Cali, Mohammed Alser, Juan Gómez-Luna,
Nika Mansouri Ghiasi, Onur Mutlu

April 14th 2023
BIO-Arch

ETH zürich

SAFARI
SAFARI Research Group

Backup Slides

ASIC Breakdown

Scrooge has insignificant computation overheads

Significant resource savings from memory footprint and bandwidth reductions

ASIC Implementation	Area (mm^2)					Power (W)				
	DC Logic	TB Logic	DC SRAM	TB SRAM	total	DC Logic	TB Logic	DC SRAM	TB SRAM	total
GenASM	0.049	0.016	0.013	0.256	0.334	0.033	0.004	0.009	0.055	0.101
Scrooge	0.049	0.016	0.013	0.014	0.093	0.033	0.004	0.009	0.003	0.049

Scrooge uses 3.6x less chip area and 2.1x less power than a GenASM ASIC

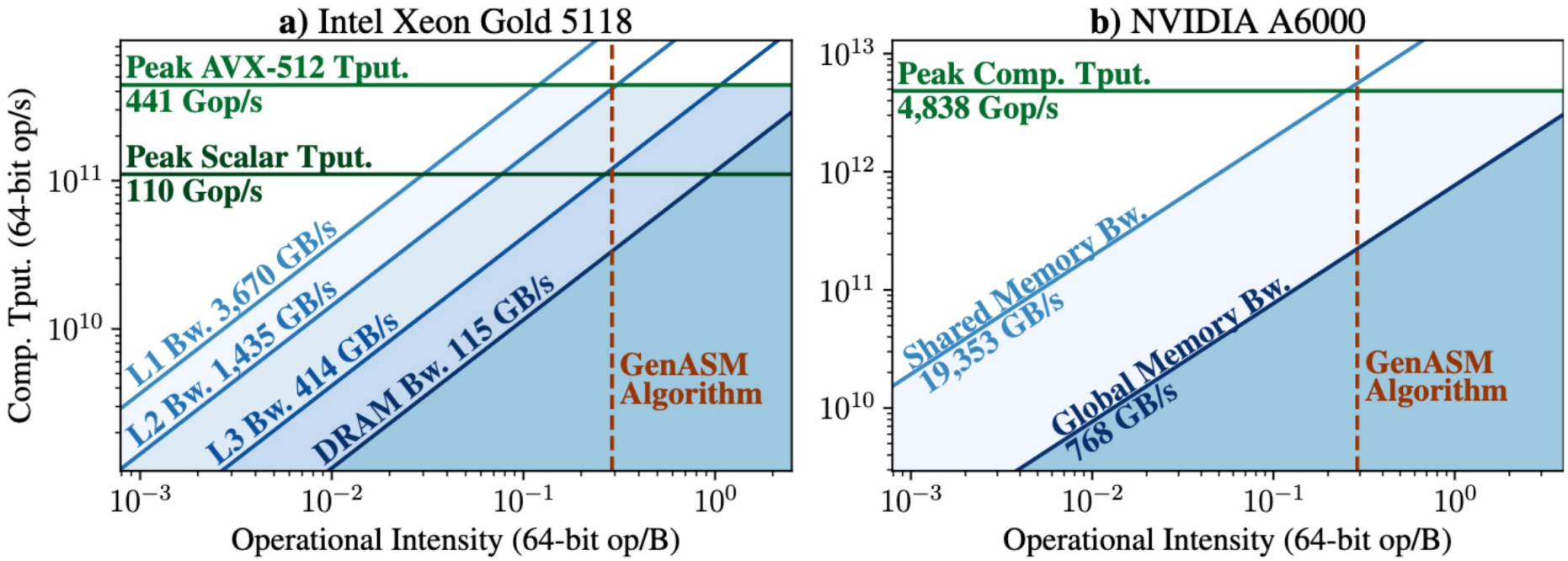
GenASM-DC Algorithm

Algorithm 1 GenASM-DC Algorithm

Inputs: text, pattern, k
Outputs: editDist

```
1:  n ← LENGTH(text)
2:  m ← LENGTH(pattern)
3:  PM ← BUILDPATTERNMASKS(pattern)
4:
5:  R[n][d] ← 11...1 ≪ d                    ▷ Initialize for all 0 ≤ d ≤ k
6:
7:  for i in (n − 1) : −1 : 0 do
8:      char ← text[i]
9:      curPM ← PM[char]
10:
11:     R[i][0] ← (R[i + 1][0] ≪ 1) | curPM          ▷ exact match
12:     for d in 1 : k do
13:         I ← R[i][d − 1] ≪ 1                      ▷ insertion
14:         D ← R[i + 1][d − 1]                      ▷ deletion
15:         S ← R[i + 1][d − 1] ≪ 1                  ▷ substitution
16:         M ← (R[i + 1][d] ≪ 1) | curPM            ▷ match
17:         R[i][d] ← I & D & S & M
18:
19: editDist ← arg min_d {MSB(R[0][d]) = 0}
```

Fulls Roofline Models

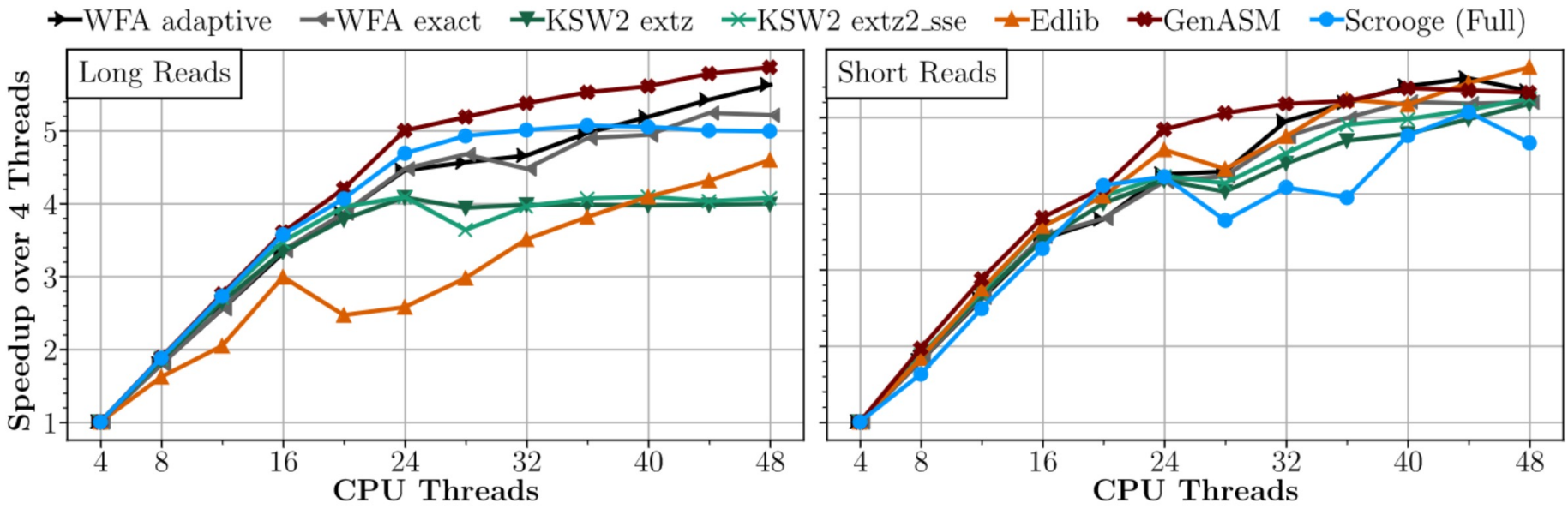


Bitvector Interpretation

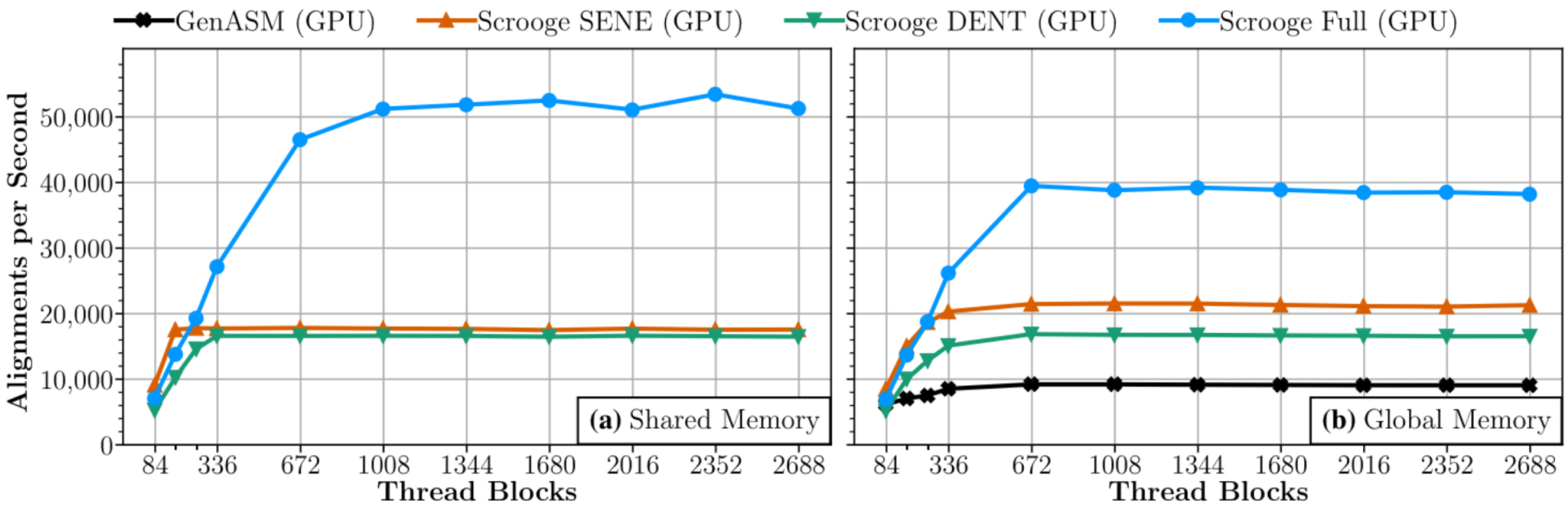
Theorem 1 *The entries (bitvectors) of* R *can be interpreted as follows:*

$$j\text{-th bit of } R[i][d] = 0 \iff distance\big(\texttt{text}[i:n), \texttt{pattern}[j:m)\big) \leq d$$

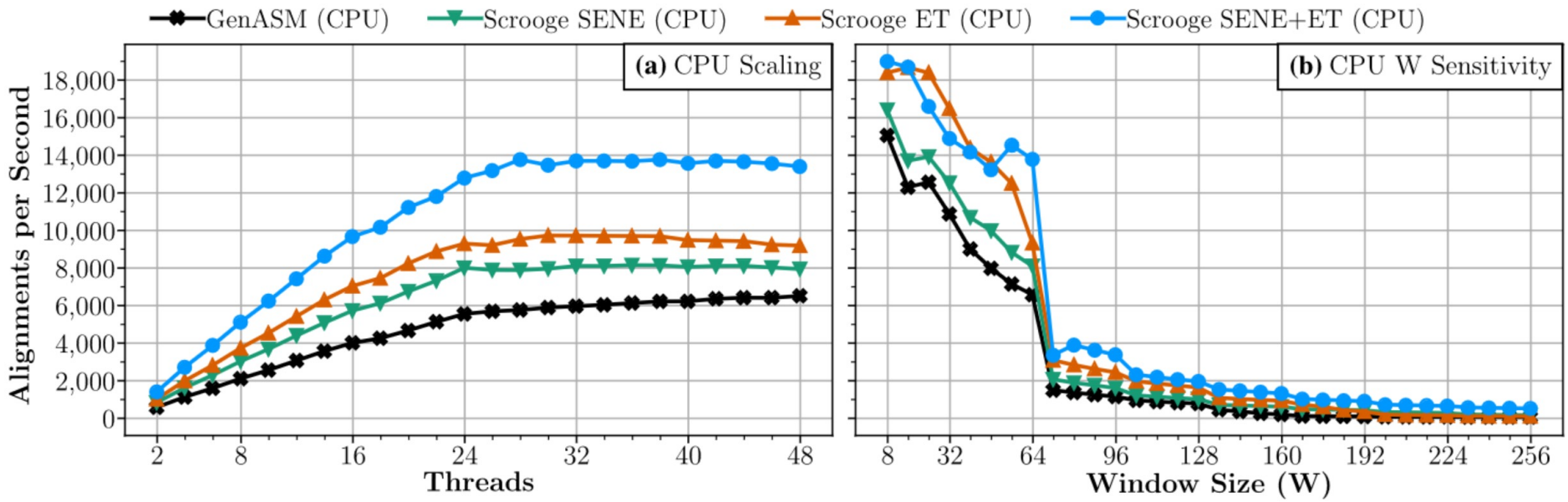
CPU Thread Scaling



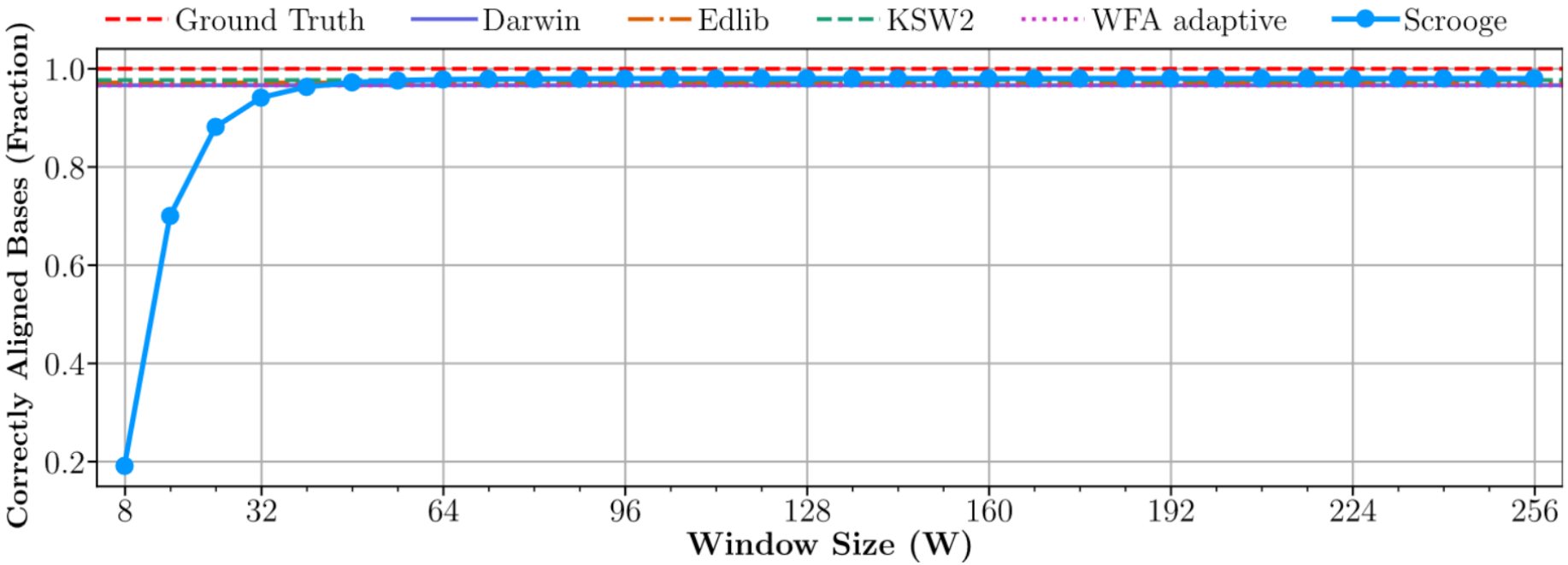
GPU Thread Scaling



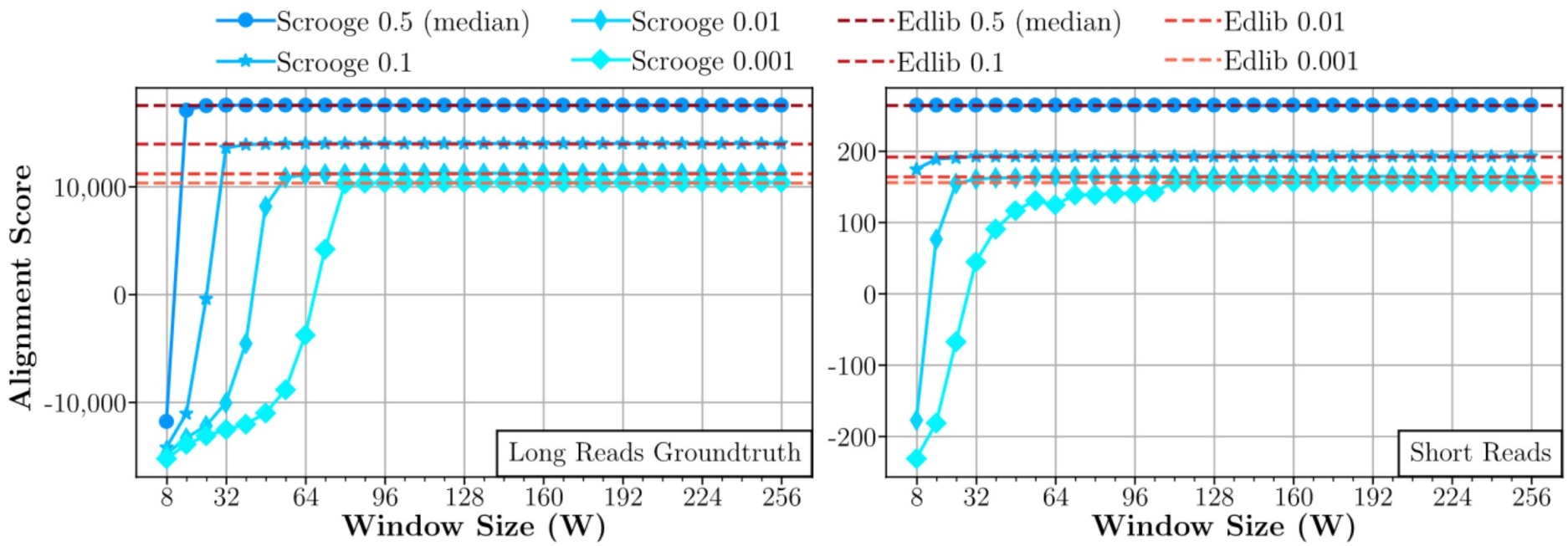
CPU Optimization Sensitivity



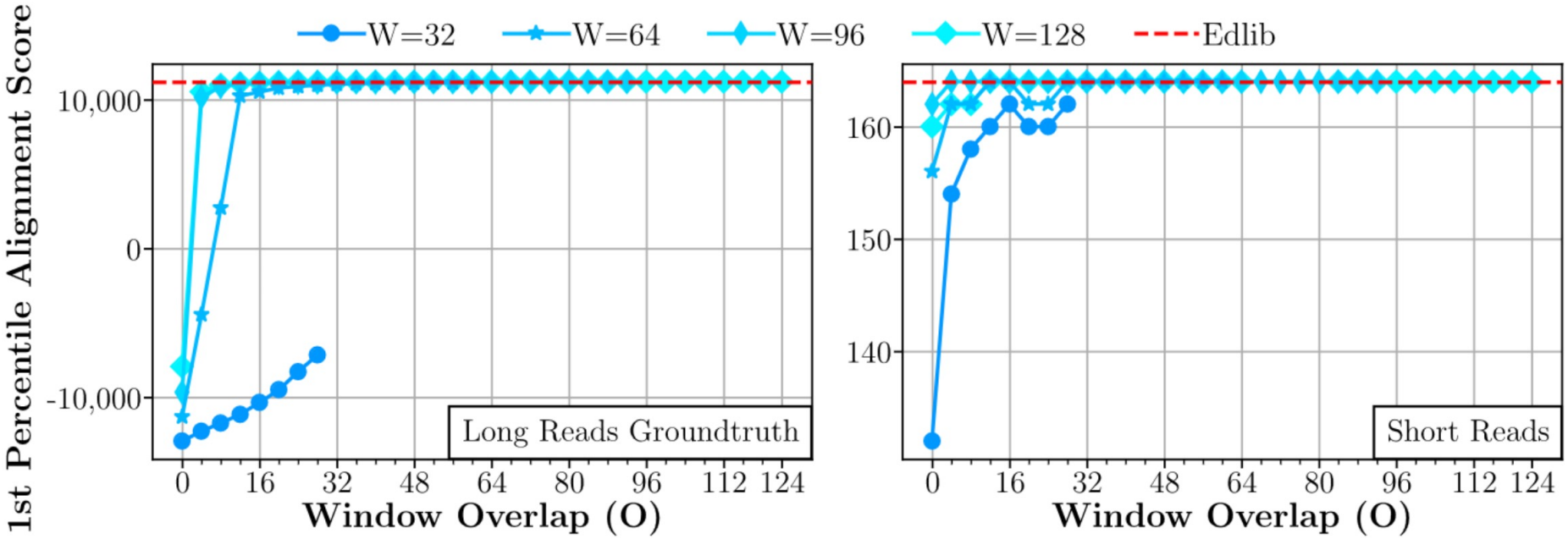
Accuracy Comparison



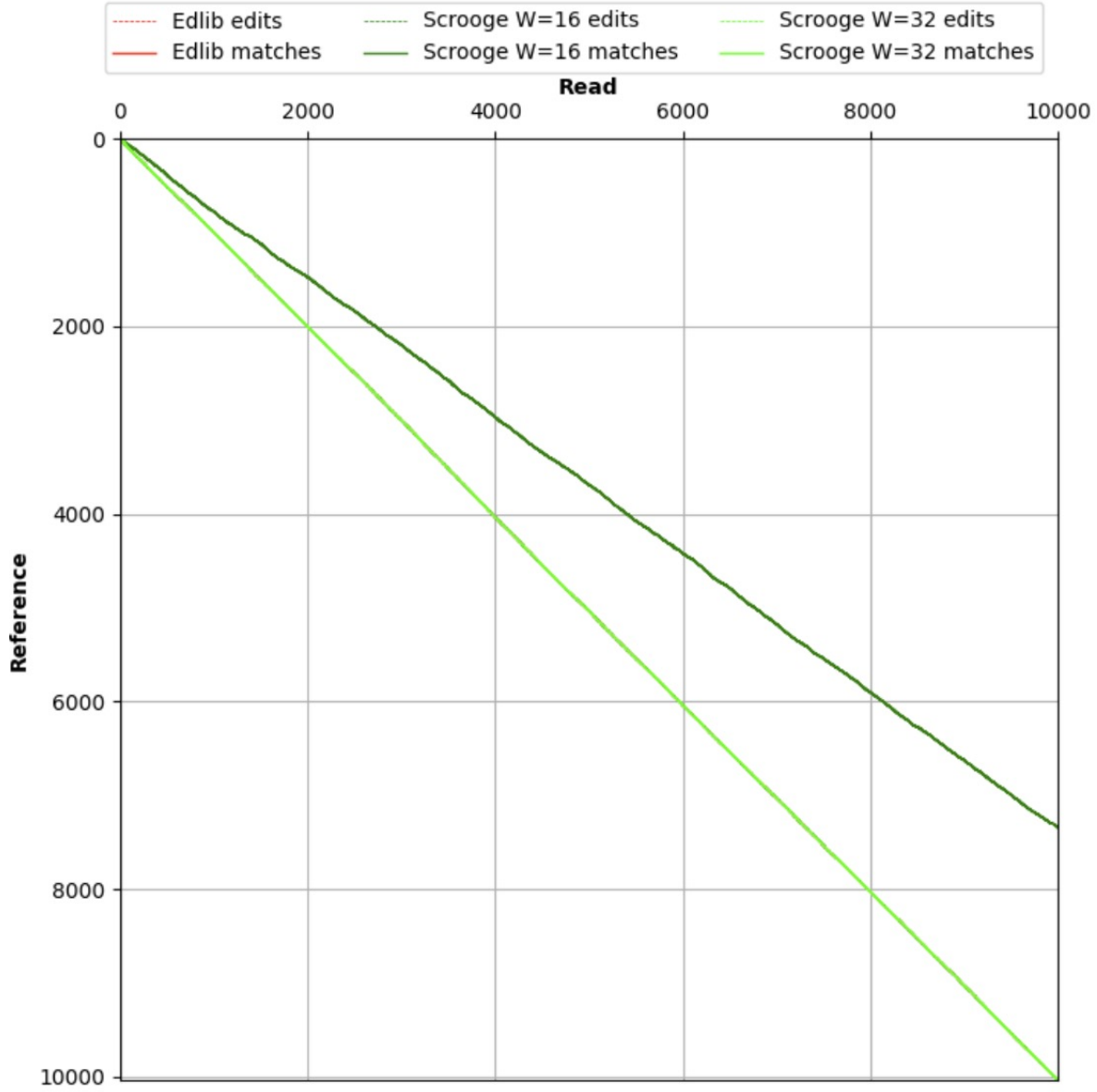
Accuracy Sensitivity to Window Size W



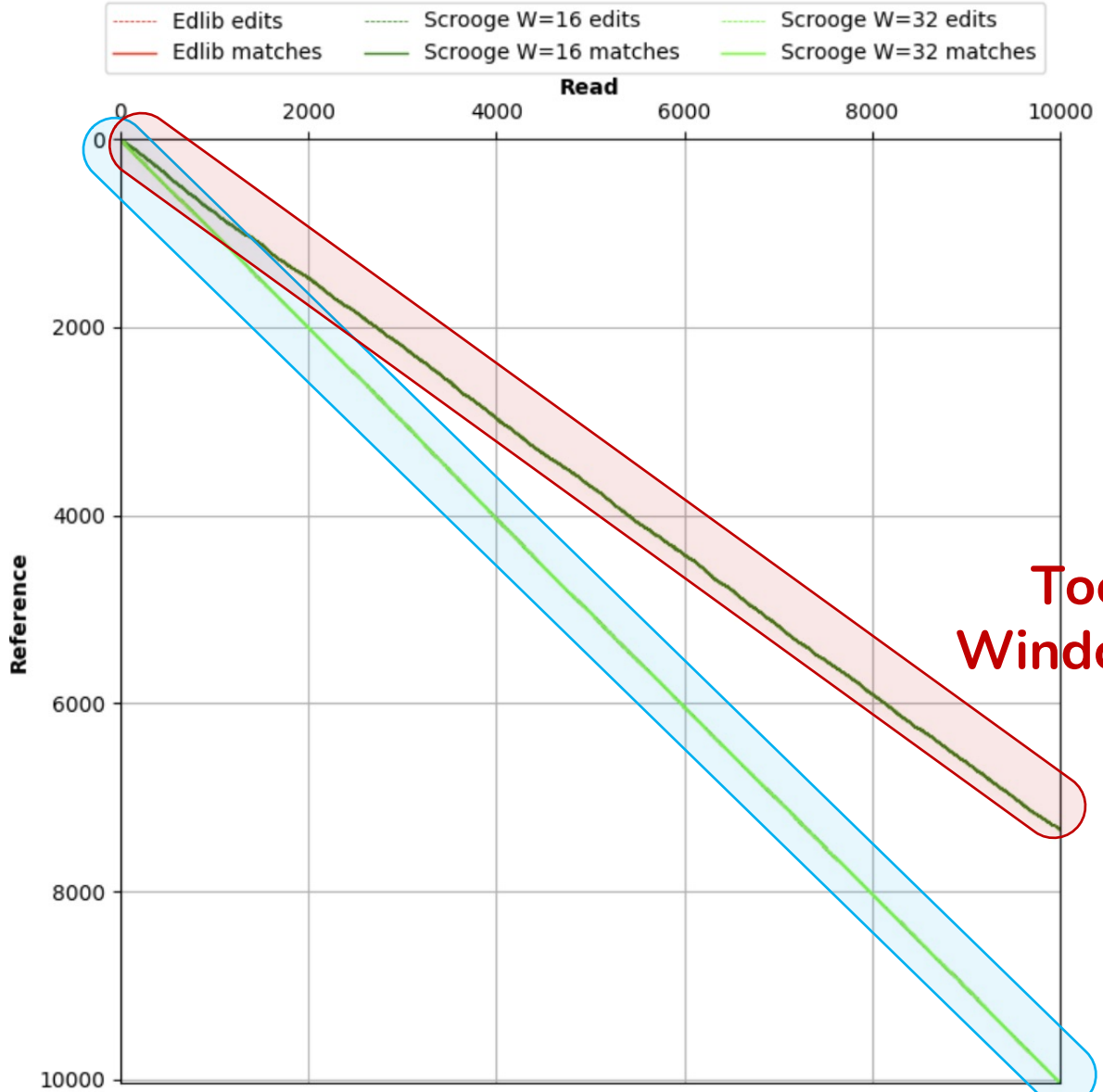
Accuracy Sensitivity to Window Overlap O



Failure Mode for Too Small Window Size W



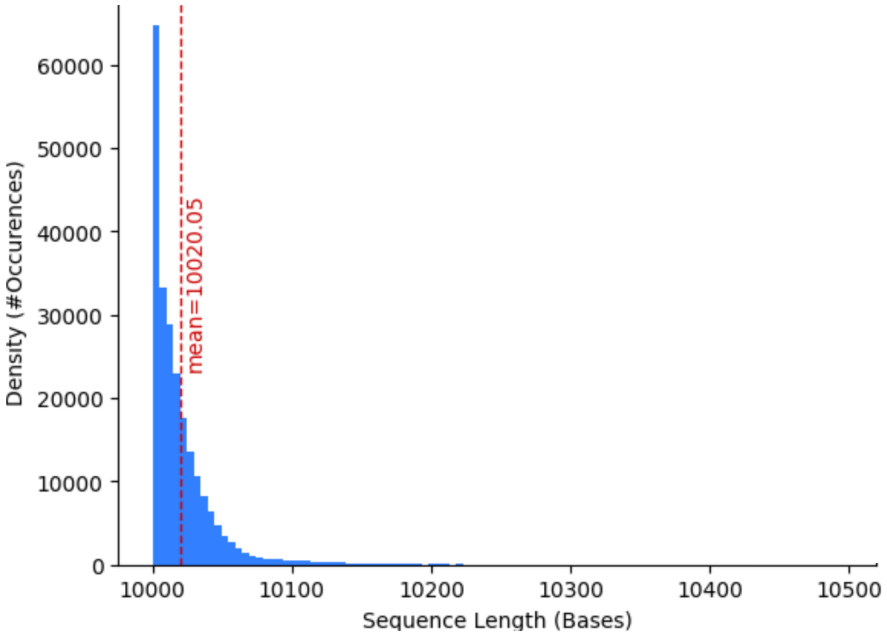
Failure Mode for Too Small Window Size W

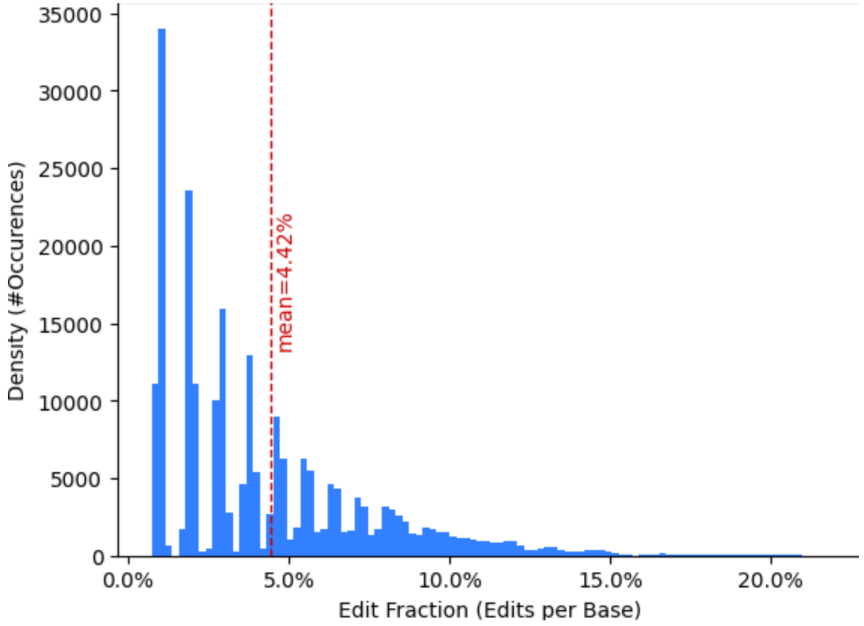


Too Small Window Size W

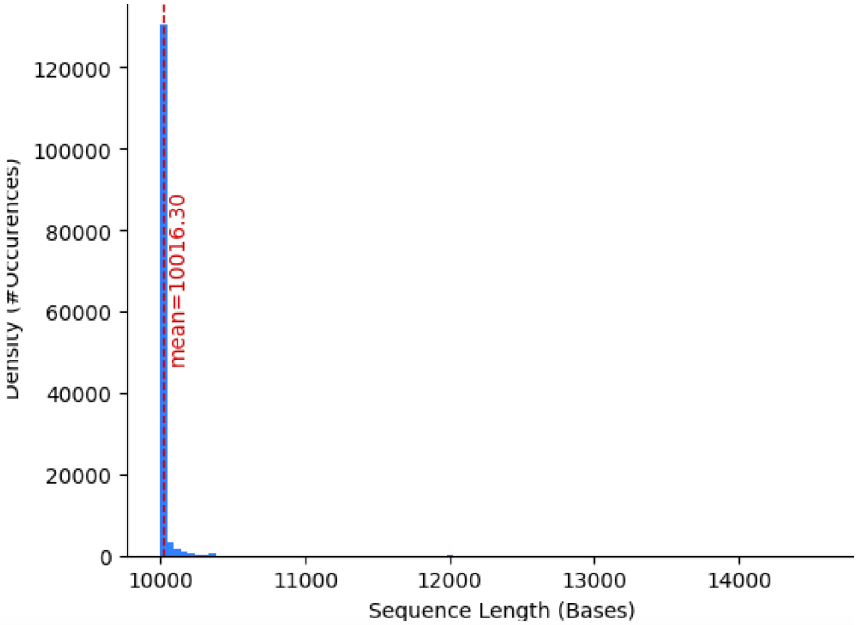
Correct Alignment

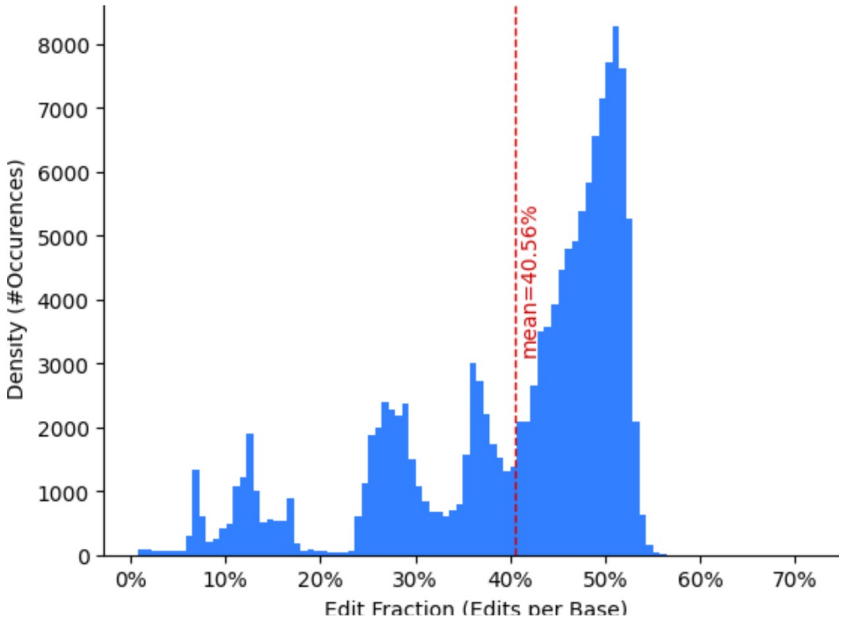
Long Read Dataset (Ground Truth)





Long Read Dataset





Short Read Dataset