Computer Architecture
Lecture 11c: Scrooge

Joël Lindegger
ETH Zürich
Fall 2023
2 November 2023
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
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**

<table>
<thead>
<tr>
<th>Text</th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact Match</td>
<td>1111</td>
<td>1111</td>
<td>1111</td>
<td>1111</td>
<td>1111</td>
</tr>
<tr>
<td>1 Edit</td>
<td>0110</td>
<td>0110</td>
<td>0110</td>
<td>1100</td>
<td>1110</td>
</tr>
<tr>
<td>2 Edits</td>
<td>0000</td>
<td>0000</td>
<td>1000</td>
<td>1100</td>
<td>1100</td>
</tr>
<tr>
<td>3 Edits</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>4 Edits</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
</tr>
</tbody>
</table>

[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**

<table>
<thead>
<tr>
<th>Text</th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact Match</td>
<td>1111</td>
<td>1111</td>
<td>1111</td>
<td>1111</td>
<td>1111</td>
</tr>
<tr>
<td>1 Edit</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1110</td>
</tr>
<tr>
<td>2 Edits</td>
<td>0000</td>
<td>0000</td>
<td>1000</td>
<td>1100</td>
<td>1100</td>
</tr>
<tr>
<td>3 Edits</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>4 Edits</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
</tr>
</tbody>
</table>

**Algorithm 1 GenASM-DC Algorithm**

**Inputs:** text, pattern, k  
**Outputs:** editDist

1. \(n \leftarrow \text{LENGTH(text)}\)
2. \(m \leftarrow \text{LENGTH(pattern)}\)
3. \(\text{PM} \leftarrow \text{BUILDPATTERNMasks(pattern)}\)
4. \(R[n][d] \leftarrow 11\ldots1 \ll d \quad \text{Initialize for all } 0 \leq d \leq k\)
5. \(\text{for } i \in (n - 1) : -1 : 0 \text{ do}\)
6. \(\text{char} \leftarrow \text{text}[i]\)
7. \(\text{curPM} \leftarrow \text{PM[char]}\)
8. \(\text{R}[i][0] \leftarrow (\text{R}[i+1][0] \ll 1) \mid \text{curPM}\)
9. \(\text{for } d \in 1 : k \text{ do}\)
10. \(I \leftarrow \text{R}[i][d-1] \ll 1\)
11. \(D \leftarrow \text{R}[i+1][d-1]\)
12. \(S \leftarrow \text{R}[i+1][d-1] \ll 1\)
13. \(M \leftarrow (\text{R}[i+1][d] \ll 1) \mid \text{curPM}\)
14. \(\text{R}[i][d] \leftarrow I \& D \& S \& M\)
15. \(\text{editDist} \leftarrow \arg \min_d \{\text{MSB}(R[0][d]) = 0\}\)

[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
<table>
<thead>
<tr>
<th>1</th>
<th>Background</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Analysis of GenASM</td>
</tr>
<tr>
<td>3</td>
<td>Scrooge Algorithm</td>
</tr>
<tr>
<td>4</td>
<td>Scrooge Implementations</td>
</tr>
<tr>
<td>5</td>
<td>Evaluation</td>
</tr>
<tr>
<td>6</td>
<td>Conclusion</td>
</tr>
</tbody>
</table>
Pairwise Sequence Alignment (PSA)

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

SAFARI
SALAMI
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**

<table>
<thead>
<tr>
<th>SAFARI</th>
<th>SALAMI</th>
</tr>
</thead>
</table>

**SAFARI**

<table>
<thead>
<tr>
<th>SAFARI</th>
<th>SAFARI</th>
</tr>
</thead>
</table>

**SAFARI**
Pairwise Sequence Alignment (PSA)

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

**substitutions**

```
SAFARI
SALAMI
```

```
SA\-FARI
SAHFARI
```
Pairwise Sequence Alignment (PSA)

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

**Substitutions**

```
SAFARI
SALAMI
```

**Insertions**

```
SA- FARI
SAHFARI
```

```
SAFARI
SAARI
```
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
- The total number of edits should be minimal

The CIGAR string is the output of PSA
### Arithmetic Dynamic Programming for PSA

#### Needleman-Wunsch

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

Next entry is calculated from three neighbors using arithmetic operations

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>G</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
The GenASM Algorithm

<table>
<thead>
<tr>
<th>Text</th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact Match</td>
<td><strong>1111</strong></td>
<td><strong>1111</strong></td>
<td><strong>1111</strong></td>
<td><strong>1111</strong></td>
<td><strong>1111</strong></td>
</tr>
<tr>
<td>1 Edit</td>
<td><strong>0110</strong></td>
<td><strong>1010</strong></td>
<td><strong>1100</strong></td>
<td><strong>1110</strong></td>
<td><strong>1110</strong></td>
</tr>
<tr>
<td>2 Edits</td>
<td><strong>0000</strong></td>
<td><strong>0000</strong></td>
<td><strong>1000</strong></td>
<td><strong>1100</strong></td>
<td><strong>1100</strong></td>
</tr>
<tr>
<td>3 Edits</td>
<td><strong>0000</strong></td>
<td><strong>0000</strong></td>
<td><strong>0000</strong></td>
<td><strong>1000</strong></td>
<td><strong>1000</strong></td>
</tr>
<tr>
<td>4 Edits</td>
<td><strong>0000</strong></td>
<td><strong>0000</strong></td>
<td><strong>0000</strong></td>
<td><strong>0000</strong></td>
<td><strong>0000</strong></td>
</tr>
</tbody>
</table>

**GenASM**

Next entry is calculated from three neighbors using bitwise operations

Particularly efficient in hardware
The GenASM Algorithm (Traceback)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact Match</td>
<td>1111</td>
<td>1111</td>
<td>1111</td>
<td>1111</td>
<td>1111</td>
</tr>
<tr>
<td>1 Edit</td>
<td>0</td>
<td>1110</td>
<td>0</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>2 Edits</td>
<td>0000</td>
<td>0000</td>
<td>1000</td>
<td>1100</td>
<td>1100</td>
</tr>
<tr>
<td>3 Edits</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>4 Edits</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
</tr>
</tbody>
</table>

Search leftmost column for the topmost 0

The row number is the edit distance

Traceback obtains the CIGAR string by backtracking the origin of the topmost 0 in the leftmost column.
<table>
<thead>
<tr>
<th>1</th>
<th>Background</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td><strong>Analysis of GenASM</strong></td>
</tr>
<tr>
<td>3</td>
<td>Scrooge Algorithm</td>
</tr>
<tr>
<td>4</td>
<td>Scrooge Implementations</td>
</tr>
<tr>
<td>5</td>
<td>Evaluation</td>
</tr>
<tr>
<td>6</td>
<td>Conclusion</td>
</tr>
</tbody>
</table>
Analysis of GenASM

Can we do better?

ASIC [Senol Cali+], Application Specific Integrated Circuit

Is GenASM suitable to commodity hardware?

CPU

GPU

[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?
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.

Lost Performance due to Limited Bandwidth

Operational Intensity

CPU
Intel Xeon Gold 5118

Global Memory Bw. 768 GB/s

GenASM Algorithm

Desired Operating Point

Throughput
Higher is better

GPU
NVIDIA RTX A6000

DRAM Bw. 115 GB/s

GenASM Algorithm

Actual Operating Point
If Data Resides Off-Chip
Memory Footprint Analysis of GenASM

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

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Intel Xeon Gold 5118</strong></td>
<td>32KiB L1D per Core</td>
<td>99KiB Shared Memory per SM</td>
</tr>
<tr>
<td><strong>NVIDIA RTX A6000</strong></td>
<td>96.5KiB GenASM Memory Footprint</td>
<td>96.5KiB GenASM Memory Footprint</td>
</tr>
</tbody>
</table>

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

- CPU: Intel Xeon Gold 5118
  - 32KiB L1D per Core
  - 96.5KiB GenASM Memory Footprint

- GPU: NVIDIA RTX A6000
  - 99KiB Shared Memory per SM
  - 96.5KiB GenASM Memory Footprint
Memory Footprint Analysis of GenASM

Inefficiency #2

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

To Utilize Hardware Multithreading

Second GenASM Instance

96.5KiB GenASM Memory Footprint

Second GenASM Instance

99KiB Shared Memory per SM

Third GenASM Instance

96.5KiB GenASM Memory Footprint

CPU
Intel Xeon Gold 5118

GPU
NVIDIA RTX A6000
### Unnecessary Work in GenASM

<table>
<thead>
<tr>
<th>Text</th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact Match</td>
<td>1111</td>
<td>1111</td>
<td>1111</td>
<td>1111</td>
<td>1111</td>
</tr>
<tr>
<td>1 Edit</td>
<td>0110</td>
<td>0110</td>
<td>1000</td>
<td>1100</td>
<td>1110</td>
</tr>
<tr>
<td>2 Edits</td>
<td>0000</td>
<td>0000</td>
<td>1000</td>
<td>1100</td>
<td>1100</td>
</tr>
<tr>
<td>3 Edits</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>4 Edits</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
</tr>
</tbody>
</table>

**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

1 Background
2 Analysis of GenASM
3 Scrooge Algorithm
4 Scrooge Implementations
5 Evaluation
6 Conclusion
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

SENEN
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**

<table>
<thead>
<tr>
<th>Text</th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Exact Match</strong></td>
<td>1111</td>
<td>1111</td>
<td>1111</td>
<td>1111</td>
<td>1111</td>
</tr>
<tr>
<td><strong>1 Edit</strong></td>
<td>0110</td>
<td>0100</td>
<td>1100</td>
<td>1110</td>
<td>1110</td>
</tr>
<tr>
<td><strong>2 Edits</strong></td>
<td>0000</td>
<td>0000</td>
<td>1000</td>
<td>1100</td>
<td>1100</td>
</tr>
<tr>
<td><strong>3 Edits</strong></td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td><strong>4 Edits</strong></td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
</tr>
</tbody>
</table>
SENE: Store Entries, Not Edges

<table>
<thead>
<tr>
<th>Text</th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact Match</td>
<td>1111</td>
<td>1111</td>
<td>1111</td>
<td>1111</td>
<td>1111</td>
</tr>
<tr>
<td>1 Edit</td>
<td>0010</td>
<td>0000</td>
<td>0000</td>
<td>0100</td>
<td>1110</td>
</tr>
<tr>
<td>2 Edits</td>
<td>0000</td>
<td>0000</td>
<td>1000</td>
<td>1100</td>
<td>1100</td>
</tr>
<tr>
<td>3 Edits</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>4 Edits</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
</tr>
</tbody>
</table>

Stored by Scrooge

Stored by GenASM

0000 Insertion

0000 Deletion

0000 Match

0000 Entry
SENE: Store Entries, Not Edges

<table>
<thead>
<tr>
<th>Text</th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact Match</td>
<td>1111</td>
<td>1111</td>
<td>1111</td>
<td>1111</td>
<td>1111</td>
</tr>
<tr>
<td>1 Edit</td>
<td>0000</td>
<td>1000</td>
<td>0000</td>
<td>1100</td>
<td>1110</td>
</tr>
<tr>
<td>2 Edits</td>
<td>0000</td>
<td>0000</td>
<td>1000</td>
<td>1100</td>
<td>1100</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Text</th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact Match</td>
<td>1111</td>
<td>111</td>
<td>1111</td>
<td>1111</td>
<td>1111</td>
</tr>
<tr>
<td>1 Edit</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1100</td>
<td>1110</td>
</tr>
<tr>
<td>2 Edits</td>
<td>0000</td>
<td>0000</td>
<td>1000</td>
<td>1100</td>
<td>1100</td>
</tr>
<tr>
<td>3 Edits</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>4 Edits</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
</tr>
</tbody>
</table>

- 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

<table>
<thead>
<tr>
<th>Text</th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact Match</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1 Edit</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2 Edits</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 Edits</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Stop building the table as soon as a 0 is found in the leftmost bit and start traceback.

Cannot be Reached by Traceback

**ET eliminates** the **unnecessary work** on average, at least **25%** of cells are unnecessary.
Outline

1 Background

2 Analysis of GenASM

3 Scrooge Algorithm

4 Scrooge Implementations

5 Evaluation

6 Conclusion
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: 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.
Outline

1. Background
2. Analysis of GenASM
3. Scrooge Algorithm
4. Scrooge Implementations
5. Evaluation
6. Conclusion
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

- Higher is better

Alignments per Second

Long Reads

- CPU
- GPU

$10^2$ $10^3$ $10^4$ $10^5$
For long reads, **Scrooge outperforms GenASM by 2.1x on CPU and 5.9x on GPU.**
Short Read Throughput

Alignments per Second

Higher is better

CPU

GPU

Short Reads

$10^6$

$10^7$
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
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

SAFARI 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

ETH Zurich
SAFARI Research Group
Backup Slides
ASIC Breakdown

<table>
<thead>
<tr>
<th>ASIC Implementation</th>
<th>DC Logic (mm²)</th>
<th>TB Logic (mm²)</th>
<th>DC SRAM (mm²)</th>
<th>TB SRAM (mm²)</th>
<th>Total Area (mm²)</th>
<th>DC Logic (W)</th>
<th>TB Logic (W)</th>
<th>DC SRAM (W)</th>
<th>TB SRAM (W)</th>
<th>Total Power (W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GenASM</td>
<td>0.049</td>
<td>0.016</td>
<td>0.013</td>
<td>0.256</td>
<td>0.334</td>
<td>0.033</td>
<td>0.004</td>
<td>0.009</td>
<td>0.055</td>
<td>0.101</td>
</tr>
<tr>
<td>Scrooge</td>
<td>0.049</td>
<td>0.016</td>
<td>0.013</td>
<td>0.014</td>
<td>0.093</td>
<td>0.033</td>
<td>0.004</td>
<td>0.009</td>
<td>0.003</td>
<td>0.049</td>
</tr>
</tbody>
</table>

Scrooge has insignificant computation overheads

Significant resource savings from memory footprint and bandwidth reductions

**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

**a) Intel Xeon Gold 5118**

- Peak AVX-512 Tput: 441 Gop/s
- Peak Scalar Tput: 110 Gop/s
- L1 Bw: 3.670 GB/s
- L2 Bw: 1.435 GB/s
- L3 Bw: 414 GB/s
- DRAM Bw: 115 GB/s
- Operational Intensity (64-bit op/B)

**b) NVIDIA A6000**

- Peak Comp. Tput: 4,838 Gop/s
- Shared Memory Bw: 19,353 GB/s
- Global Memory Bw: 768 GB/s
- Operational Intensity (64-bit op/B)

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

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

![Graph showing speedup over 4 threads for different benchmarks with respect to CPU threads. The graph is divided into two sections: Long Reads and Short Reads. The benchmarks include WFA adaptive, WFA exact, KSW2 extz, KSW2 extz2_sse, Edlib, GenASM, and Scrooge (Full).]
GPU Thread Scaling

(a) Shared Memory

(b) Global Memory
CPU Optimization Sensitivity

(a) CPU Scaling

(b) CPU W Sensitivity
Accuracy Comparison

The graph compares the accuracy of different alignment algorithms across various window sizes. The x-axis represents the window size (W), and the y-axis shows the correctly aligned bases as a fraction. The algorithms compared are:

- Ground Truth
- Darwin
- Edlib
- KSW2
- WFA adaptive
- Scrooge

The graph illustrates how each algorithm performs under different window sizes, with Scrooge achieving near-perfect alignment across all window sizes, as indicated by the blue line converging to 1.0 on the y-axis.
Accuracy Sensitivity to Window Size $W$

![Graph showing accuracy sensitivity to window size with different alignment score and window size values for Scrooge and Edlib.]
Accuracy Sensitivity to Window Overlap

![Graph showing the sensitivity of accuracy to window overlap for different window sizes. The graphs compare long and short reads with ground truth and Edlib.]
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

![Graph 1: Distribution of Sequence Length](image1)

![Graph 2: Distribution of Edit Fraction](image2)
Short Read Dataset

![Sequence Length vs. Density](image1)

- Mean sequence length: 151.18 bases

![Edit Fraction vs. Density](image2)

- Mean edit fraction: 10.48%