Genome Sequencing

- **Genome sequencing**: Enables us to determine the order of the DNA sequence in an organism’s genome
  - Plays a pivotal role in:
    - Personalized medicine
    - Outbreak tracing
    - Understanding of evolution

- Modern genome sequencing machines extract smaller randomized fragments of the original DNA sequence, known as **reads**
  - *Short reads*: a few hundred base pairs, error rate of ~0.1%
  - *Long reads*: thousands to millions of base pairs, error rate of 10–15%
Genome Sequence Analysis

- **Read mapping:** *First key step* in genome sequence analysis (GSA)
  - Aligns *reads* to one or more possible locations within the *reference genome*, and
  - Finds the *matches* and *differences* between the read and the reference genome segment at that location

- Multiple steps of read mapping require *approximate string matching*
  - Approximate string matching (ASM) enables read mapping to account for *sequencing errors* and *genetic variations* in the reads

- Bottlenecked by the *computational power and memory bandwidth limitations* of existing systems
GenASM: ASM Framework for GSA

Our Goal:

Accelerate approximate string matching by designing a fast and flexible framework, which can accelerate multiple steps of genome sequence analysis

- **GenASM**: First ASM acceleration framework for GSA
  - Based upon the Bitap algorithm
    - Uses fast and simple bitwise operations to perform ASM
  - Modified and extended ASM algorithm
    - Highly-parallel Bitap with long read support
    - Novel bitvector-based algorithm to perform traceback
  - Co-design of our modified scalable and memory-efficient algorithms with low-power and area-efficient hardware accelerators
Use Cases & Key Results

(1) Read Alignment
- 116× speedup, 37× less power than Minimap2 (state-of-the-art SW)
- 111× speedup, 33× less power than BWA-MEM (state-of-the-art SW)
- 3.9× better throughput, 2.7× less power than Darwin (state-of-the-art HW)
- 1.9× better throughput, 82% less logic power than GenAx (state-of-the-art HW)

(2) Pre-Alignment Filtering
- 3.7× speedup, 1.7× less power than Shouji (state-of-the-art HW)

(3) Edit Distance Calculation
- 22–12501× speedup, 548–582× less power than Edlib (state-of-the-art SW)
- 9.3–400× speedup, 67× less power than ASAP (state-of-the-art HW)
Genome Analysis

Sequencing

Read Mapping

Variant Calling

Scientific Discovery

Source: Prof. Onur Mutlu’s lecture slides
Source: Prof. Onur Mutlu’s lecture slides

Billions of Short Reads

1 Sequencing

2 Read Mapping

Bottlenecked in Mapping!!

Illumina HiSeq4000

300 M bases/min

on average

2 M bases/min

(0.6%)
Read Mapping

Reference genome → Indexing → Hash-table based index

Reads → Seeding → Potential mapping locations

Reference segment → Pre-Alignment Filtering → Remaining potential mapping locations

Query read → Pre-Alignment Filtering

Read Alignment → Optimal alignment
Read alignment is often the bottleneck in read mapping.
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

![Alignment Example]

SAFARI

SALAMI
Pairwise Sequence Alignment (PSA)

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

Substitutions

\[
\text{SAFARI} \quad \text{SALAMI}
\]
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
SAHFAFI
```

```
SAFARI
SAARFI
```
Pairwise Sequence Alignment (PSA)

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

**Substitutions**

```
SAFARI
SALAMI
```

**Insertions**

```
SAFARI
SAHFAWI
```

```
SAFARI
SAHAFRAI
```
**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

Next entry is calculated from three neighbors using arithmetic operations
The Bitap 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>1111</td>
<td>1111</td>
<td>1111</td>
<td>1111</td>
<td>1111</td>
</tr>
<tr>
<td>1 Edit</td>
<td>0110</td>
<td>1010</td>
<td>1100</td>
<td>1110</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>

**Bitap**

Next entry is calculated from three neighbors using bitwise operations.

Particularly efficient in hardware.

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

Next entry is calculated from three neighbors using arithmetic operations.
Bitap Algorithm (cont’d.)

For each character of the text (char):

Copy previous R bitvectors as oldR

R[0] = (oldR[0] << 1) | PM [char]

For d = 1...k:

- deletion = oldR[d-1]
- substitution = oldR[d-1] << 1
- insertion = R[d-1] << 1
- match = (oldR[d] << 1) | PM [char]

R[d] = deletion & mismatch & insertion & match

Check MSB of R[d]:

- If 1, no match.
- If 0, match with d many errors.
Bitap Algorithm (cont’d.)

For each character of the text (char):

Copy previous R bitvectors as oldR

\[ R[0] = (\text{oldR}[0] \ll 1) \mid \text{PM}[\text{char}] \]

For \( d = 1 \ldots k \):

- deletion \( = \text{oldR}[d-1] \)
- substitution \( = \text{oldR}[d-1] \ll 1 \)
- insertion \( = \text{R}[d-1] \ll 1 \)
- match \( = (\text{oldR}[d] \ll 1) \mid \text{PM}[\text{char}] \)

\[ R[d] = \text{deletion} \& \text{mismatch} \& \text{insertion} \& \text{match} \]

Check MSB of \( R[d] \):

- If 1, no match.
- If 0, match with \( d \) many errors.

Data dependency between iterations (i.e., no parallelization)
Bitap Algorithm (cont’d.)

For each character of the text (char):

Copy previous R bitvectors as oldR

\[ R[0] = (\text{oldR}[0] \ll 1) \mid \text{PM} \text{[char]} \]

For \( d = 1 \ldots k \):

- \text{deletion} = \text{oldR}[d-1]
- \text{substitution} = \text{oldR}[d-1] \ll 1
- \text{insertion} = \text{R}[d-1] \ll 1
- \text{match} = (\text{oldR}[d] \ll 1) \mid \text{PM} \text{[char]} \]

\[ R[d] = \text{deletion} \& \text{mismatch} \& \text{insertion} \& \text{match} \]

Check MSB of \( R[d] \):

- If 1, no match.
- If 0, match with \( d \) many errors.

Does not store and process these intermediate bitvectors to find the optimal alignment (i.e., no traceback)
## Limitations of Bitap

1. **Data Dependency Between Iterations:**
   - Two-level data dependency forces the consecutive iterations to take place sequentially.

2. **No Support for Traceback:**
   - Bitap does not include any support for optimal alignment identification.

3. **No Support for Long Reads:**
   - Each bitvector has a length equal to the length of the pattern.
   - Bitwise operations are performed on these bitvectors.

4. **Limited Compute Parallelism:**
   - Text-level parallelism.
   - Limited by the number of compute units in existing systems.

5. **Limited Memory Bandwidth:**
   - High memory bandwidth required to read and write the computed bitvectors to memory.
GenASM: ASM Framework for GSA

- Approximate string matching (ASM) acceleration framework based on the Bitap algorithm
- *First ASM acceleration framework* for genome sequence analysis
- We overcome the *five limitations* that hinder Bitap’s use in genome sequence analysis:
  - Modified and extended ASM algorithm
    - Highly-parallel Bitap with long read support
    - Novel bitvector-based algorithm to perform *traceback*
  - Specialized, low-power and area-efficient hardware for both modified Bitap and novel traceback algorithms
The GenASM-DC 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><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>1010</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>

GenASM-DC stores all computed bitvectors for later “traceback”
The GenASM-TB Algorithm

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

<table>
<thead>
<tr>
<th></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>
</tr>
<tr>
<td>1 Edit</td>
<td>0110</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>2 Edits</td>
<td>0000</td>
<td>0000</td>
<td>1000</td>
<td>1100</td>
</tr>
<tr>
<td>3 Edits</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>1000</td>
</tr>
<tr>
<td>4 Edits</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
</tr>
</tbody>
</table>

The row number is the edit distance.

Search leftmost column for the topmost 0.

The GenASM-TB Algorithm

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

GenASM-DC:
generates bitvectors and performs edit Distance Calculation

GenASM-TB:
performs TraceBack and assembles the optimal alignment

Main Memory

Host CPU

DC-SRAM

GenASM-DC

GenASM-TB

GenASM-DC

TB-SRAM₁

TB-SRAM₂

... 

TB-SRAMₙ

GenASM-TB Accelerator

Accelerator
GenASM Hardware Design

Main Memory

Host CPU

DC-SRAM

GenASM-DC

GenASM-DC: generates bitvectors and performs edit Distance Calculation

GenASM-TB

GenASM-TB: performs TraceBack and assembles the optimal alignment

1. Reference text & query locations
2. Reference text & query pattern
3. Sub-text & sub-pattern
4. Generate bitvectors
5. Write bitvectors
6. Read bitvectors
7. Find the traceback output

TB-SRAM₁
TB-SRAM₂
TB-SRAMₙ
Our *specialized compute units* and *on-chip SRAMs* help us to:

- Match the rate of computation with memory capacity and bandwidth
- Achieve high performance and power efficiency
- Scale linearly in performance with the number of parallel compute units that we add to the system
GenASM-DC: Hardware Design

- **Linear cyclic systolic array** based accelerator
  - Designed to **maximize parallelism** and **minimize memory bandwidth and memory footprint**

![Diagram of Processing Block (PB) and Processing Core (PC)]
GenASM-TB: Hardware Design

- Very simple logic:
  1. **Reads the bitvectors** from one of the TB-SRAMs using the computed address
  2. **Performs the required bitwise comparisons** to find the traceback output for the current position
  3. **Computes the next TB-SRAM address** to read the new set of bitvectors

![Diagram of GenASM-TB: Hardware Design](image-url)
Evaluation Methodology

- We evaluate GenASM using:
  - Synthesized SystemVerilog models of the GenASM-DC and GenASM-TB accelerator datapaths
  - Detailed simulation-based performance modeling

- 16GB HMC-like 3D-stacked DRAM architecture
  - 32 vaults
  - 256GB/s of internal bandwidth, clock frequency of 1.25GHz
  - In order to achieve high parallelism and low power-consumption
  - Within each vault, the logic layer contains a GenASM-DC accelerator, its associated DC-SRAM, a GenASM-TB accelerator, and TB-SRAMs.
## Evaluation Methodology (cont’d.)

<table>
<thead>
<tr>
<th></th>
<th>SW Baselines</th>
<th>HW Baselines</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Read Alignment</strong></td>
<td>Minimap2&lt;sup&gt;1&lt;/sup&gt;</td>
<td>GACT (Darwin)&lt;sup&gt;3&lt;/sup&gt;</td>
</tr>
<tr>
<td></td>
<td>BWA-MEM&lt;sup&gt;2&lt;/sup&gt;</td>
<td>SillaX (GenAx)&lt;sup&gt;4&lt;/sup&gt;</td>
</tr>
<tr>
<td><strong>Pre-Alignment Filtering</strong></td>
<td>–</td>
<td>Shouji&lt;sup&gt;5&lt;/sup&gt;</td>
</tr>
<tr>
<td><strong>Edit Distance Calculation</strong></td>
<td>Edlib&lt;sup&gt;6&lt;/sup&gt;</td>
<td>ASAP&lt;sup&gt;7&lt;/sup&gt;</td>
</tr>
</tbody>
</table>

For Use Case 1: Read Alignment, we compare GenASM with:

- Minimap2 and BWA-MEM (state-of-the-art SW)
  - Running on Intel® Xeon® Gold 6126 CPU (12-core) operating @2.60GHz with 64GB DDR4 memory
  - Using two simulated datasets:
    - Long ONT and PacBio reads: 10Kbp reads, 10-15% error rate
    - Short Illumina reads: 100-250bp reads, 5% error rate

- GACT of Darwin and SillaX of GenAx (state-of-the-art HW)
  - Open-source RTL for GACT
  - Data reported by the original work for SillaX
  - GACT is best for long reads, SillaX is best for short reads
Evaluation Methodology (cont’d.)

- **For Use Case 2: Pre-Alignment Filtering**, we compare GenASM with:
  - *Shouji* (state-of-the-art **HW** – FPGA-based filter)
    - Using two datasets provided as test cases:
      - 100bp reference-read pairs with an edit distance threshold of 5
      - 250bp reference-read pairs with an edit distance threshold of 15

- **For Use Case 3: Edit Distance Calculation**, we compare GenASM with:
  - *Edlib* (state-of-the-art **SW**)
    - Using two 100Kbp and 1Mbp sequences with similarity ranging between 60%-99%
  - *ASAP* (state-of-the-art **HW** – FPGA-based accelerator)
    - Using data reported by the original work
Based on our synthesis of GenASM-DC and GenASM-TB accelerator datapaths using the Synopsys Design Compiler with a 28nm LP process:
- Both GenASM-DC and GenASM-TB operate @ 1GHz

<table>
<thead>
<tr>
<th></th>
<th>Area (mm²)</th>
<th>Power (W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GenASM-DC (64 PEs)</td>
<td>0.049</td>
<td>0.033</td>
</tr>
<tr>
<td>GenASM-TB</td>
<td>0.016</td>
<td>0.004</td>
</tr>
<tr>
<td>DC-SRAM (8 KB)</td>
<td>0.013</td>
<td>0.009</td>
</tr>
<tr>
<td>TB-SRAMs (64 x 1.5 KB)</td>
<td>0.256</td>
<td>0.055</td>
</tr>
</tbody>
</table>

Total (1 vault): 0.334 mm² 0.101 W
Total (32 vaults): 10.69 mm² 3.23 W

% of a Xeon CPU core: 1% 1%
Key Results – Area and Power

Based on our synthesis of GenASM-DC and GenASM-TB accelerator datapaths using the Synopsys Design Compiler with a 28nm LP process:
- Both GenASM-DC and GenASM-TB operate @ 1GHz

GenASM has low area and power overheads
Key Results – Use Case 1

(1) Read Alignment Step of Read Mapping
  o Find the optimal alignment of how reads map to candidate reference regions

(2) Pre-Alignment Filtering for Short Reads
  o Quickly identify and filter out the unlikely candidate reference regions for each read

(3) Edit Distance Calculation
  o Measure the similarity or distance between two sequences
GenASM achieves 648× and 116× speedup over 12-thread runs of BWA-MEM and Minimap2, while reducing power consumption by 34× and 37×
GenASM provides **3.9× better throughput**, **6.6× the throughput per unit area**, and **10.5× the throughput per unit power**, compared to GACT of Darwin.
Key Results – Use Case 1 (Short Reads)

**SW**
GenASM achieves 111× and 158× speedup over 12-thread runs of BWA-MEM and Minimap2, while reducing power consumption by 33× and 31×

**HW**
GenASM provides 1.9× better throughput and uses 63% less logic area and 82% less logic power, compared to SillaX of GenAx
Additional Details in the Paper

- Details of the GenASM-DC and GenASM-TB algorithms
- Big-O analysis of the algorithms
- Detailed explanation of evaluated use cases
- Evaluation methodology details (datasets, baselines, performance model)
- Additional results for the three evaluated use cases
- Sources of improvements in GenASM (algorithm-level, hardware-level, technology-level)
- Discussion of four other potential use cases of GenASM
GenASM [MICRO 2020]

Damla Senol Cali, Gurpreet S. Kalsi, Zulal Bingöl, Can Firtina, Lavanya Subramanian, Jeremie S. Kim, Rachata Ausavarungnirun, Mohammed Alser, Juan Gomez-Luna, Amirali Boroumand, Anant Nori, Allison Scibisz, Sreenivas Subramoney, Can Alkan, Saugata Ghose, and Onur Mutlu,

“GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis”

GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis

Damla Senol Cali†× Gurpreet S. Kalsi× Zülat Bingöl∨ Can Firtina○ Lavanya Subramanian‡ Jeremie S. Kim‡
Rachata Ausavarungnirun○ Mohammed Alser○ Juan Gomez-Luna○ Amirali Boroumand† Anant Nori×
Allison Scibisz† Sreenivas Subramoney× Can Alkan∨ Saugata Ghose×† Onur Mutlu†∨
†Carnegie Mellon University ×Processor Architecture Research Lab, Intel Labs ∨Bilkent University ○ETH Zürich
Facebook ○King Mongkut’s University of Technology North Bangkok *University of Illinois at Urbana–Champaign
P&S Genomics
Lecture 8a: GenASM

Joël Lindegger

ETH Zürich
Spring 2023
27 April 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

April 14th 2023
RECOMB-Seq

ETH zürich
SAFARI Research Group
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
Analysis of GenASM

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

Can we do better?

Is GenASM suitable to commodity hardware?
Roofline Analysis of GenASM

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

Throughput

Higher is better

CPU
- Intel Xeon Gold 5118
  - Peak Comp. Tput: 4,838 Gop/s
  - Global Memory Bw: 768 GB/s

GenASM Algorithm

GPU
- NVIDIA RTX A6000
  - Peak AVX-512 Tput: 441 Gop/s
  - DRAM Bw: 115 GB/s

Lost Performance due to Limited Bandwidth

Desired Operating Point

Actual Operating Point

If Data Resides Off-Chip
Roofline Analysis of GenASM

Inefficiency #1

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

Throughput

Higher is better

CPU

Intel Xeon Gold 5118

Peak Comp. Tput.
4,838 Gop/s

Global Memory Bw.
768 GB/s

GenASM Algorithm

Desired Operating Point

Lost Performance due to Limited Bandwidth

GPU

NVIDIA RTX A6000

Peak AVX-512 Tput.
441 Gop/s

DRAM Bw. 1.15 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?

**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

SAFARI
Memory Footprint Analysis of GenASM

- 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

To Utilize Simultaneous Multithreading (Hyperthreading in Intel speak)
Memory Footprint Analysis of GenASM

Inefficiency #2

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

**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

To Utilize Simultaneous Multithreading

SAFARI
**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>

**Inefficiency #3**

GenASM does **unnecessary work** by computing DP cells which cannot be reached by Traceback.
1. Large memory bandwidth requirement

2. Large memory footprint

3. Unnecessary work
Outline

1 Executive Summary

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

SEN
E
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

**SENEM**
- 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>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>010</td>
<td>1100</td>
<td>1110</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>
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>0110</td>
<td>0000</td>
<td>0000</td>
<td>0000</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 GenASM

0000 Insertion

0000 Deletion

Stored by Scrooge

0000 Entry

0000 Match
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>0+10</td>
<td>0+0</td>
<td>0+0</td>
<td>1+10</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

 SEN E
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>1111</td>
<td>1111</td>
<td>1111</td>
<td>1111</td>
</tr>
<tr>
<td>1 Edit</td>
<td>0100</td>
<td>0100</td>
<td>1100</td>
<td>1110</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

SENEN
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>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></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. Executive Summary
2. Analysis of GenASM
3. Scroooge Algorithm
4. Scroooge 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 Executive Summary
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

Alignments per Second

Higher is better

Long Reads

CPU

GPU

SAFARI
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

Short Reads

10^7

10^6

CPU

GPU
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
## 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
P&S Genomics

Lecture 8b: Scrooge

Joël Lindegger

ETH Zürich
Spring 2023
27 April 2023
Backup Slides
## ASIC Breakdown

<table>
<thead>
<tr>
<th>ASIC Implementation</th>
<th>Area ($mm^2$)</th>
<th>Power (W)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DC Logic</td>
<td>TB Logic</td>
</tr>
<tr>
<td>GenASM</td>
<td>0.049</td>
<td>0.016</td>
</tr>
<tr>
<td>Scrooge</td>
<td>0.049</td>
<td>0.016</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
- GenASM Algorithm

**b) NVIDIA A6000**

- Peak Comp. Tput.: 4,838 Gop/s
- Shared Memory Bw.: 19,353 GB/s
- Global Memory Bw.: 768 GB/s
- GenASM Algorithm
Theorem 1  The entries (bitvectors) of $R$ can be interpreted as follows:

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

(a) Shared Memory
(b) Global Memory
CPU Optimization Sensitivity

(a) CPU Scaling

(b) CPU W Sensitivity
Accuracy Comparison

![Accuracy Comparison Graph](image)

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

Correctly Aligned Bases (Fraction) vs. Window Size (W)

SAFARI
Accuracy Sensitivity to Window Size W

![Graph showing alignment score sensitivity to window size for Scrooge and Edlib with different settings.](image)

- **Scrooge**
  - 0.5 (median)
  - 0.01
  - 0.1
  - 0.001

- **Edlib**
  - 0.5 (median)
  - 0.01
  - 0.1
  - 0.001

*Long Reads Groundtruth* and *Short Reads* categories are indicated in the graphs.
Accuracy Sensitivity to Window Overlap O

1st Percentile Alignment Score vs. Window Overlap (O) for different window sizes (W=32, 64, 96, 128). The graphs show the alignment scores for Long Reads Groundtruth and Short Reads, with Edlib as a reference.
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)

- **Sequence Length (Bases) Distribution**
  - Mean: 10020.05

- **Edit Fraction (Edits per Base) Distribution**
  - Mean: 4.42%
Long Read Dataset
Short Read Dataset