Previous Lecture: GenASM and Scrooge

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>0110</td>
<td>0100</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>

- **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.**
SeGraM
A Universal Hardware Accelerator for Genomic Sequence-to-Graph and Sequence-to-Sequence Mapping
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%
Sequencing

1

Read Mapping

2

Bottlenecked in Mapping!!

Illumina HiSeq4000

300 M bases/min

on average

2 M bases/min (0.6%)

Source: Prof. Onur Mutlu’s lecture slides
Mapping the reads to a reference genome (i.e., *read mapping*) is a critical step in genome sequence analysis.

**Linear Reference**: ACGTACGT

- **Read**: ACGG
- **Alternative Sequence**: ACGGACGT
- **Alternative Sequence**: ACGTTACGT
- **Alternative Sequence**: ACG–ACGT

**Graph-based Reference**:

- **Read**: ACGG

---

**Sequence-to-Sequence (S2S) Mapping**

**Sequence-to-Graph (S2G) Mapping**

Sequence-to-graph mapping results in notable quality improvements. However, it is a more difficult computational problem, with no prior hardware design.
SeGraM: First Graph Mapping Accelerator

Our Goal:

Specialized, high-performance, scalable, and low-cost algorithm/hardware co-design that alleviates bottlenecks in multiple steps of sequence-to-graph mapping

SeGraM: First universal algorithm/hardware co-designed genomic mapping accelerator that can effectively and efficiently support:

- Sequence-to-graph mapping
- Sequence-to-sequence mapping
- Both short and long reads
Use Cases & Key Results

(1) Sequence-to-Graph (S2G) Mapping
- 5.9×/106× speedup, 4.1×/3.0× less power than GraphAligner for long and short reads, respectively (state-of-the-art SW)
- 3.9×/742× speedup, 4.4×/3.2× less power than vg for long and short reads, respectively (state-of-the-art SW)

(2) Sequence-to-Graph (S2G) Alignment
- 41×–539× speedup over PaSGAL with AVX-512 support (state-of-the-art SW)

(3) Sequence-to-Sequence (S2S) Alignment
- 1.2×/4.8× higher throughput than GenASM and GACT of Darwin for long reads (state-of-the-art HW)
- 1.3×/2.4× higher throughput than GenASM and SillaX of GenAX for short reads (state-of-the-art HW)
Outline

- Introduction
- Background
  - Read Mapping
  - Genome Graphs
  - Sequence-to-Graph Mapping
- SeGraM: Universal Genomic Mapping Accelerator
  - High-Level Overview
  - MinSeed
  - BitAlign
  - Use Cases
- Evaluation
- Conclusion
Solving the Puzzle (S2S Mapping)

Reference Genome

For a Human:
3 Billion Characters (3GB)

Determines e.g., Eye Color, Shape of Face, Allergies, ...
Solving the Puzzle (S2S Mapping)

Reference Genome

For a Human:
3 Billion Characters (3GB)

Determines e.g., Eye Color,
Shape of Face, Allergies, ...

Reads

150 – 2,000,000 Characters Each

Origin Locations are Unknown
Solving the Puzzle (S2S Mapping)

Reference Genome

For a Human:
3 Billion Characters (3GB)

Determines e.g., Eye Color, Shape of Face, Allergies, ...

Reads

150 – 2,000,000 Characters Each

Origin Locations are Unknown

S2S (Sequence-to-Sequence) Mapping
Recovers the Origin Locations
According to 1 Reference Genome

Joël Lindegger
SeedeX: A Genome Sequencing Accelerator for Optimal Alignments in Subminimal Space

Daichi Fujiki, Shunhao Wu, Nathan Ozog, Kush Goliya, David Blaauw, Satish Narayanasamy, Reetuparna Das
University of Michigan
{dfujiki, shunhao, ozog, kgoliya, blaauw, nsatish, reetudas}@umich.edu

GenAx: A Genome Sequencing Accelerator

Daichi Fujiki*, Arun Subramaniyan*, Tianjun Zhang*, Yu Zeng, Reetuparna Das, David Blaauw, Satish Narayanasamy
University of Michigan - Ann Arbor
{dfujiki, arunsub, tianjunz, yuzeng, reetudas, blaauw, nsatish}@umich.edu

Darwin: A Genomics Co-processor Provides up to 15,000× acceleration on long read assembly

Yatish Turakhia
Stanford University
yatish@stanford.edu

Gill Bejerano
Stanford University
bejerano@stanford.edu

William J. Dally
Stanford University
NVIDIA Research
dally@stanford.edu

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

Damla Senol Cali†, Gurpreet S. Kalsi†, Züalıl 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

A Framework for High-throughput Sequence Alignment using Real Processing-in-Memory Systems

Safiaa Diab1, Amir Nassereldine1, Mohammed Alser2, Juan Gómez Luna2, Onur Mutlu2, Izzat El Hajj1
1American University of Beirut, Lebanon  2ETH Zürich, Switzerland

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

Joël Lindegger1,*, Damla Senol Cali2, Mohammed Alser1, Juan Gómez-Luna1, Nika Mansouri Ghiasi1 and Onur Mutlu1,*
1Department of Information Technology and Electrical Engineering, ETH Zurich, Zurich 8006, Switzerland and 2Bionano Genomics, San Diego, CA 92121, USA.
Solving the Puzzle (S2S Mapping)

Reference Genome

Reads

Variants
Solving the Puzzle (S2S Mapping)

Reference Genome

Reads

Some Reads Can Be Mapped due to Sufficient Context

Some Reads Fail to Be Mapped Because They are Too Different from the Single Reference

Reference Bias!
Avoiding Reference Bias in Read Mapping

- **Solution 1:** Attempt to **map to all known** reference genomes one-by-one
  - For **N times slowdown** for N reference genomes
  - There could be **unknown reference genomes** (e.g., hybrids)

- **Solution 2:** Build a single **graph-based** reference that **unifies** all known genetic variations
  - **Avoids redundant computation** and data
  - Captures some **unknown reference genomes**
Genome Graphs

Genome graphs:

- Combine the **linear reference genome** with the **known genetic variations in the entire population** as a graph-based data structure.
- Enable us to move away from aligning with a single linear reference genome (**reference bias**) and **more accurately express the genetic diversity in a population**.

**Sequence #1:** ACGTACGT
Genome Graphs

Genome graphs:

- Combine the **linear reference genome** with the **known genetic variations in the entire population** as a graph-based data structure

- Enable us to move away from aligning with a single linear reference genome (**reference bias**) and more accurately express the genetic diversity in a population

**Sequence #1:** ACGTACGT

**Sequence #2:** ACGGACGT
Genome Graphs

Genome graphs:

- Combine the linear reference genome with the known genetic variations in the entire population as a graph-based data structure.
- Enable us to move away from aligning with a single linear reference genome (reference bias) and more accurately express the genetic diversity in a population.

Sequence #1: ACGTACGT
Sequence #2: ACGGACGT
Genome Graphs

Genome graphs:

- Combine the linear reference genome with the known genetic variations in the entire population as a graph-based data structure.
- Enable us to move away from aligning with a single linear reference genome (reference bias) and more accurately express the genetic diversity in a population.

**Sequence #1:** ACGTACGT

**Sequence #2:** ACGGACGT

**Sequence #3:** ACGTTACGT
Genome Graphs

Genome graphs:

- Combine the linear reference genome with the known genetic variations in the entire population as a graph-based data structure.
- Enable us to move away from aligning with a single linear reference genome (reference bias) and more accurately express the genetic diversity in a population.

Sequence #1: ACGTACGT
Sequence #2: ACGGACGT
Sequence #3: ACGTTACGT
Genome Graphs

Genome graphs:

- Combine the linear reference genome with the known genetic variations in the entire population as a graph-based data structure.
- Enable us to move away from aligning with a single linear reference genome (reference bias) and more accurately express the genetic diversity in a population.

Sequence #1: ACGTACGT
Sequence #2: ACGGACGT
Sequence #3: ACGTTACGT
Sequence #4: ACGACGT
Genome Graphs

Genome graphs:

- Combine the **linear reference genome** with the **known genetic variations in the entire population** as a graph-based data structure
- Enable us to move away from aligning with a single linear reference genome (**reference bias**) and more accurately express the genetic diversity in a population

**Sequence #1:** ACGTACGT  
**Sequence #2:** ACGGACGT  
**Sequence #3:** ACGTTACGT  
**Sequence #4:** ACGACGT
Sequence-to-Graph Mapping Pipeline

**Pre-Processing Steps (Offline)**

1. **Genome Graph Construction**
   (construct the graph using a linear reference genome and variations)

2. **Indexing**
   (index the nodes of the graph)

**Seed-and-Extend Steps (Online)**

1. **Seeding**
   (query the index & find the seed matches)

2. **Filtering/Chaining/Clustering**
   (filter out dissimilar query read and subgraph pairs)

3. **S2G Alignment**
   (perform distance/score calculation & traceback)

**Linear reference genome**

**Known genetic variations**

**Reads from sequenced genome**

**Hash-table-based index (of graph nodes)**

**Optimal alignment between read & subgraph**
Previous Lecture: GenASM and Scrooge

The GenASM Algorithm (Traceback)

- 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.
S2S vs. S2G Alignment

Sequence-to-Sequence (S2S) Alignment
In contrast to S2S alignment, S2G alignment must incorporate non-neighboring characters as well whenever there is an edge (i.e., hop) from the non-neighboring character to the current character.
Analysis of State-of-the-Art Tools

Based on our analysis with GraphAligner and vg:

**Observation 1:** Alignment step is the bottleneck

**Observation 2:** Alignment suffers from high cache miss rates

**Observation 3:** Seeding suffers from the DRAM latency bottleneck

**Observation 4:** Baseline tools scale sublinearly

**Observation 5:** Existing S2S mapping accelerators are unsuitable for the S2G mapping problem

**Observation 6:** Existing graph accelerators are unable to handle S2G alignment
Outline

- Introduction
- Background
  - Read Mapping
  - Genome Graphs
  - Sequence-to-Graph Mapping
- SeGraM: Universal Genomic Mapping Accelerator
  - High-Level Overview
  - MinSeed
  - BitAlign
  - Use Cases
- Evaluation
- Conclusion
SeGraM: Universal Genomic Mapping Accelerator

- *First universal genomic mapping accelerator* that can support both sequence-to-graph mapping and sequence-to-sequence mapping, for both short and long reads

- *First algorithm/hardware co-design* for accelerating sequence-to-graph mapping

- We base SeGraM upon a minimizer-based seeding algorithm
- We propose a *novel bitvector-based alignment algorithm* to perform approximate string matching between a read and a graph-based reference genome

- We co-design both algorithms with high-performance, scalable, and efficient hardware accelerators
SeGraM Hardware Design

MinSeed: first hardware accelerator for Minimizer-based Seeding

BitAlign: first hardware accelerator for (Bitvector-based) sequence-to-graph Alignment
SeGraM Hardware Design

MinSeed: first hardware accelerator for Minimizer-based Seeding

BitAlign: first hardware accelerator for (Bitvector-based) sequence-to-graph Alignment
MinSeed HW

- MinSeed = 3 computation modules + 3 scratchpads + memory interface
  - Computation modules: Implemented with simple logic
  - Scratchpads: 50kB in total; employ double buffering technique to hide the latency of MinSeed
  - High-Bandwidth Memory (HBM): Enables low-latency and highly-parallel memory access
BitAlign HW

- Linear cyclic systolic array-based accelerator
- Based on the GenASM hardware design*

[*] D. Senol Cali et al. "GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis" (MICRO’20)
The GenASM Algorithm (Traceback)

- 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.
BitAlign HW

- Linear cyclic systolic array-based accelerator
- Based on the GenASM hardware design*
- Incorporates hop queue registers to feed the bitvectors of non-neighboring characters/nodes (i.e., hops)

[*] D. Senol Cali et al. "GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis" (MICRO’20)
Overall System Design of SeGraM

High Bandwidth Memory (HBM2E) Stack

CH0  CH1  CH2  ...  CH6  CH7

SeGraM Module (1 x per HBM2E stack)

Host

MS  BA  SeGraM Acc.
MS  BA  SeGraM Acc.
MS  BA  SeGraM Acc.
MS  BA  SeGraM Acc.
MS  BA  SeGraM Acc.
MS  BA  SeGraM Acc.

x 4
Use Cases of SeGraM

(1) Sequence-to-Graph Mapping

(2) Sequence-to-Graph Alignment

(3) Sequence-to-Sequence Alignment

(4) Seeding
Outline

- Introduction
- Background
  - Read Mapping
  - Genome Graphs
  - Sequence-to-Graph Mapping
- SeGraM: Universal Genomic Mapping Accelerator
  - High-Level Overview
  - MinSeed
  - BitAlign
  - Use Cases
- Evaluation
- Conclusion
Evaluation Methodology

- **Performance, Area and Power Analysis:**
  - Synthesized SystemVerilog models of the MinSeed and BitAlign accelerator datapaths
  - Simulation- and spreadsheet-based performance modeling

- **Baseline Comparison Points:**
  - GraphAligner, vg, and HGA for sequence-to-graph mapping
  - PaSGAL for sequence-to-graph alignment
  - Darwin, GenAx, and GenASM for sequence-to-sequence alignment

- **Datasets:**
  - Graph-based reference: GRCh38 + 7 VCF files for HG001-007
  - Simulated datasets for both short and long reads
Based on our synthesis of MinSeed and BitAlign accelerator datapaths using the Synopsys Design Compiler with a 28nm process (@ 1GHz):

<table>
<thead>
<tr>
<th>Component</th>
<th>Area (mm²)</th>
<th>Power (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinSeed – Logic</td>
<td>0.017</td>
<td>10.8</td>
</tr>
<tr>
<td>Read Scratchpad (6 kB)</td>
<td>0.012</td>
<td>7.9</td>
</tr>
<tr>
<td>Minimizer Scratchpad (40 kB)</td>
<td>0.055</td>
<td>22.7</td>
</tr>
<tr>
<td>Seed Scratchpad (4 kB)</td>
<td>0.008</td>
<td>6.4</td>
</tr>
<tr>
<td>BitAlign – Edit Distance Calculation Logic with Hop Queue Registers (64 PEs)</td>
<td>0.393</td>
<td>378.0</td>
</tr>
<tr>
<td>BitAlign – Traceback Logic</td>
<td>0.020</td>
<td>2.7</td>
</tr>
<tr>
<td>Input Scratchpad (24 kB)</td>
<td>0.033</td>
<td>13.3</td>
</tr>
<tr>
<td>Bitvector Scratchpads (128 kB)</td>
<td>0.329</td>
<td>316.2</td>
</tr>
<tr>
<td><strong>Total – 1 SeGraM Accelerator</strong></td>
<td><strong>0.867</strong></td>
<td><strong>758.0 (0.8 W)</strong></td>
</tr>
<tr>
<td><strong>Total – 4 SeGraM Modules (32 SeGraM Accelerators)</strong></td>
<td><strong>27.744</strong></td>
<td><strong>24.3 W</strong></td>
</tr>
<tr>
<td><strong>HBM2E (4 stacks)</strong></td>
<td><strong>--</strong></td>
<td><strong>3.8 W</strong></td>
</tr>
</tbody>
</table>
SeGraM provides 5.9× and 3.9× throughput improvement over GraphAligner and vg, while reducing the power consumption by 4.1× and 4.4×
SeGraM provides 106× and 742× throughput improvement over GraphAligner and vg, while reducing the power consumption by 3.0× and 3.2×
Key Results – BitAlign (S2G Alignment)

BitAlign provides \(41\times-539\times\) speedup over PaSGAL
Key Results – BitAlign (S2S Alignment)

- BitAlign can also be used for sequence-to-sequence alignment
  - The cost of more functionality: extra hop queue registers
  - We do not sacrifice any performance

- For long reads (over GACT of Darwin and GenASM):
  - 4.8× and 1.2× throughput improvement,
  - 2.7× and 7.5× higher power consumption, and
  - 1.5× and 2.6× higher area overhead

- For short reads (over SillaX of GenAx and GenASM):
  - 2.4× and 1.3× throughput improvement
Outline

- Introduction
- Background
  - Read Mapping
  - Genome Graphs
  - Sequence-to-Graph Mapping
- SeGraM: Universal Genomic Mapping Accelerator
  - High-Level Overview
  - MinSeed
  - BitAlign
  - Use Cases
- Evaluation
- Conclusion
Additional Details in the Paper

- Details of the pre-processing steps of SeGraM
- Details of the MinSeed and BitAlign algorithms
- Details of the MinSeed and BitAlign hardware designs
- Bottleneck analysis of the existing tools
- Evaluation methodology details (datasets, baselines, performance model)
- Additional results for the three evaluated use cases
- Sources of improvements in SeGraM
- Comparison of GenASM and SeGraM
Conclusion

- **SeGraM**: First universal algorithm/hardware co-designed genomic mapping accelerator that supports:
  - Sequence-to-graph (S2G) & sequence-to-sequence (S2S) mapping
  - Short & long reads
  - **MinSeed**: First minimizer-based seeding accelerator
  - **BitAlign**: First (bitvector-based) S2G alignment accelerator

- **SeGraM supports multiple use cases**:
  - End-to-end S2G mapping
  - S2G alignment
  - S2S alignment
  - Seeding

- **SeGraM outperforms state-of-the-art software & hardware solutions**
SeGraM [ISCA 2022]

Damla Senol Cali, Konstantinos Kanellopoulos, Joel Lindegger, Zulal Bingol, Gurpreet S. Kalsi, Ziyi Zuo, Can Firtina, Meryem Banu Cavlak, Jeremie S. Kim, Nika Mansouri Ghiasi, Gagandeep Singh, Juan Gomez-Luna, Nour Almadhoun Alserr, Mohammed Alser, Sreenivas Subramoney, Can Alkan, Saugata Ghose, and Onur Mutlu

“SeGraM: A Universal Hardware Accelerator for Genomic Sequence-to-Graph and Sequence-to-Sequence Mapping”


SeGraM: A Universal Hardware Accelerator for Genomic Sequence-to-Graph and Sequence-to-Sequence Mapping

Damla Senol Cali¹ Konstantinos Kanellopoulos² Joël Lindegger² Zülal Bingöl³ Gurpreet S. Kalsi⁴ Ziyi Zuo⁵ Can Firtina² Meryem Banu Cavlak² Jeremie Kim² Nika Mansouri Ghiasi² Gagandeep Singh² Juan Gómez-Luna² Nour Almadhoun Alserr² Mohammed Alser² Sreenivas Subramoney⁴ Can Alkan³ Saugata Ghose⁶ Onur Mutlu²

¹Bionano Genomics ²ETH Zürich ³Bilkent University ⁴Intel Labs ⁵Carnegie Mellon University ⁶University of Illinois Urbana-Champaign
SeGraM: A Universal Genomic Mapping Accelerator for both Sequence-to-Graph Mapping and Sequence-to-Sequence Mapping

SeGraM is a universal genomic mapping accelerator that supports both sequence-to-graph mapping and sequence-to-sequence mapping, for both short and long reads. SeGraM consists of two main components: (1) MinSeed, the first minimizer-based seeding accelerator, which finds the candidate mapping locations (i.e., subgraphs) in a given...
The GenASM Algorithm (Traceback)

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.
Backup Slides
(SeGraM)
Genome Sequence Analysis

- Mapping the reads to a reference genome (i.e., read mapping) is a critical step in genome sequence analysis (GSA).

**Sequence-to-Sequence (S2S) Mapping**
- Maps reads collected from an individual to a known linear reference genome sequence.
- Emphasizes the genetic variations that are present in the single reference genome.
- Ignores other variations that are not represented in the single linear reference sequence.
- Introduces reference bias.
- Well studied with many available tools and accelerators.

**Sequence-to-Graph (S2G) Mapping**
- Replaces the linear reference sequence with a graph-based representation of the reference genome (genome graph).
- Captures the genetic variations and diversity across many individuals in a population.
- Results in notable quality improvements in GSA.
- More difficult computational problem.
- No prior hardware design for graph-based GSA.
SeGraM – Graph Structure

Node Table

Character Table

Edge Table
SeGraM – Index Structure

First Level: Buckets

Second Level: Minimizers

Third Level: Seed Locations
SeGraM – Selection of #Buckets

The graph illustrates the selection of buckets in terms of hash table size (GB) and the maximum number of minimizers in a bucket. The x-axis represents the number of buckets, while the y-axis shows the memory footprint (GB). The graph shows a trade-off between memory footprint and the number of minimizers per bucket, with lower bucket counts reducing memory usage but increasing the number of minimizers per bucket.
## Minimizers

<table>
<thead>
<tr>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>G</td>
</tr>
<tr>
<td>T</td>
</tr>
<tr>
<td>A</td>
</tr>
<tr>
<td>G</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>A</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$k$-mer$_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>G</td>
</tr>
<tr>
<td>T</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$k$-mer$_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
</tr>
<tr>
<td>T</td>
</tr>
<tr>
<td>A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$k$-mer$_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
</tr>
<tr>
<td>A</td>
</tr>
<tr>
<td>G</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$k$-mer$_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>G</td>
</tr>
<tr>
<td>C</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$k$-mer$_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>A</td>
</tr>
</tbody>
</table>


**lexicographically smallest $k$-mer**
MinSeed – Region Calculation

query read

graph-based reference

left-extension

right-extension

a*(1+E)

(m–b–1)*(1+E)
Algorithm 1: BitAlign Algorithm

Inputs: linearized and topologically sorted subgraph (reference), query-read (pattern), k (edit distance threshold)

Outputs: editDist (minimum edit distance), CIGARstr (traceback output)

1: n ← length of linearized reference subgraph
2: m ← length of query read
3: PM ← genPatternBitmasks(query-read)  ▶ pre-process the query read
4: 5: allR[n][d] ← 111...111  ▶ init R[d] bitvectors for all characters with 1s
6: 7: for i in (n-1):-1:0 do  ▶ iterate over each subgraph node
8:     curChar ← subgraph-nodes[i].char
9:     curPM ← PM[curChar]  ▶ retrieve the pattern bitmask
10: 11: R0 ← 111...111  ▶ status bitvector for exact match
12: for j in subgraph-nodes[i].successors do
13:     R0 ← ((R[j][0]<<1) | curPM) & R0  ▶ exact match calculation
14: allR[i][0] ← R0
15: 16: for d in 1:k do
17:     I ← (allR[i][d-1]<<1)  ▶ insertion
18:     Rd ← I  ▶ status bitvector for d errors
19: for j in subgraph-nodes[i].successors do
20:     D ← allR[j][d-1]  ▶ deletion
21:     S ← allR[j][d-1]<<1  ▶ substitution
22:     M ← (allR[j][d]<<1) | curPM  ▶ match
23:     Rd ← D & S & M & Rd
24: allR[i][d] ← Rd
25: <editDist, CIGARstr> ← traceback(allR, subgraph, query-read)
# BitAlign – HopBits

## Linearized Graph

<table>
<thead>
<tr>
<th>NodeID</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
BitAlign – Hop Length Selection

![Graph showing the fraction of total hops as a function of hop limit (number of nodes). The y-axis represents the fraction of total hops (%) ranging from 95 to 100, and the x-axis represents the hop limit (number of nodes) ranging from 0 to 20. The graph indicates an increasing trend as the hop limit increases.]
Use Cases of SeGraM

(1) **End-to-End Sequence-to-Graph Mapping**
   - The whole SeGraM design (MinSeed + BitAlign) should be employed
   - We can use SeGraM to perform mapping with both short and long reads

(2) **Sequence-to-Graph Alignment**
   - BitAlign can be used as a standalone sequence-to-graph aligner without the need of an initial seeding tool/accelerator (e.g., MinSeed)
   - BitAlign is orthogonal to and can be coupled with any seeding (or filtering) tool/accelerator

(3) **Sequence-to-Sequence Alignment**
   - BitAlign can also be used for sequence-to-sequence alignment, as it is a special and simpler variant of sequence-to-graph alignment

(4) **Seeding**
   - MinSeed can be used as a standalone seeding accelerator for both graph-based mapping and traditional linear mapping
   - MinSeed is orthogonal to and can be coupled with any alignment tool/accelerator
Sources of Improvement

- **Co-design approach for both seeding and alignment:**
  - Efficient and hardware-friendly algorithms for seeding and for alignment
  - Eliminating the data transfer bottleneck between the seeding and alignment steps of the genome sequence analysis pipeline, by placing their individual accelerators (MinSeed and BitAlign) adjacent to each other
  - Pipelining of the two accelerators within a SeGraM accelerator, which allows us to completely hide the latency of MinSeed

- **Overcoming the high cache miss rates** observed from the baseline tools by carefully designing and sizing the on-chip scratchpads and the hop queue registers and matching the rate of computation for the logic units with memory bandwidth and memory capacity
Sources of Improvement (cont’d.)

- **Addressing the DRAM latency bottleneck** by taking advantage of the natural channel subdivision exposed by HBM and eliminating any inter-accelerator interference-related latency in the memory system.

- **Scaling linearly across three dimensions:**
  - Within a single BitAlign accelerator, by incorporating processing elements (i.e., *iteration-level parallelism*),
  - Executing multiple seeds in parallel by using pipelined execution with the help of our double buffering approach (i.e., *seed-level parallelism*), and
  - Processing multiple reads concurrently without introducing inter-accelerator memory interference with the help of multiple HBM stacks that each contain the same content (i.e., *read-level parallelism*)
Backup Slides
(GenASM)
Approximate String Matching

- Sequenced genome may not exactly map to the reference genome due to genetic variations and sequencing errors

  Reference: AAAATA GTTTT A GTGCTACCTG
  Read: AAAATA GTTTT A C TGCTACCTG
  deletion substitution insertion

- Approximate string matching (ASM):
  - Detect the differences and similarities between two sequences
  - In genomics, ASM is required to:
    - Find the *minimum edit distance* (i.e., total number of edits)
    - Find the *optimal alignment* with a *traceback* step
      - Sequence of matches, substitutions, insertions and deletions, along with their positions
  - Usually implemented as a *dynamic programming* (DP) based algorithm
Bitap Algorithm

- Bitap\(^{1,2}\) performs ASM with fast and simple bitwise operations
  - Amenable to efficient hardware acceleration
  - Computes the minimum edit distance between a text (e.g., reference genome) and a pattern (e.g., read) with a maximum of \(k\) errors

- **Step 1: Pre-processing (per pattern)**
  - Generate a pattern bitmask (PM) for each character in the alphabet (A, C, G, T)
  - Each PM indicates if character exists at each position of the pattern

- **Step 2: Searching (Edit Distance Calculation)**
  - Compare all characters of the text with the pattern by using:
    - Pattern bitmasks
    - Status bitvectors that hold the partial matches
    - Bitwise operations

---

Step 2: Edit Distance Calculation

For each character of the text (char):

- Copy previous R bitvectors as oldR
- \( R[0] = (\text{oldR}[0] \ll 1) | \text{PM}[\text{char}] \)

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 = \( (\text{oldR}[d] \ll 1) | \text{PM}[\text{char}] \)

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

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

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

Large number of iterations
Bitap Algorithm (cont’d.)

- **Step 2: Edit Distance Calculation**

  For each character of the text (char):
  
  Copy previous R bitvectors as oldR
  
  \[ R[0] = (\text{oldR}[0] \ll 1) | \text{PM}[\text{char}] \]
  
  For \( d = 1 \ldots k \):
  
  - deletion \( = \text{oldR}[d-1] \)
  - substitution \( = \text{oldR}[d-1] \ll 1 \)
  - insertion \( = R[d-1] \ll 1 \)
  - match \( = (\text{oldR}[d] \ll 1) | \text{PM}[\text{char}] \)

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

  Check MSB of \( R[d] \):
  
  - If 1, no match.
  - If 0, match with \( d \) many errors.

  Data dependency between iterations (i.e., no parallelization)
Step 2: Edit Distance Calculation

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.

Does not store and process these intermediate bitvectors to find the optimal alignment (i.e., no traceback)
Example for the Bitap Algorithm

**PREPROCESSING**

Pattern Bitmaps:
- Pattern: CTGA
- \( PM(A) = 1110 \)
- \( PM(C) = 0111 \)
- \( PM(G) = 1101 \)
- \( PM(T) = 1011 \)

State Vectors:
- \( R_0 = 1111 \)
- \( R_1 = 1111 \)

---

**Text Region:** CGTGA

**Query Pattern:** CTGA

**Edit Distance Threshold (k):** 1

---

**Text[0]:** CGTGA

**oldR0:** 1111
**oldR1:** 0000

\( R_0 = (\text{oldR0} \ll 1) \lor PM(C) \)

- \( D: \text{oldR0} = 1111 \)
- \( S: \text{oldR0} \ll 1 = 1010 \)
- \( I: R_0 \ll 1 = 0110 \)
- \( M: (\text{oldR1} \ll 1) \lor PM(C) = 1111 \)

**Alignment Found @ Location=0**

---

**Text[1]:** CGTGA

**oldR0:** 1111
**oldR1:** 0000

\( R_0 = (\text{oldR0} \ll 1) \lor PM(T) \)

- \( D: \text{oldR0} = 1111 \)
- \( S: \text{oldR0} \ll 1 = 1010 \)
- \( I: R_0 \ll 1 = 0110 \)
- \( M: (\text{oldR1} \ll 1) \lor PM(T) = 1011 \)

**Alignment Found @ Location=1**

---

**Text[2]:** CGTGA

**oldR0:** 1101
**oldR1:** 1000

\( R_0 = (\text{oldR0} \ll 1) \lor PM(T) \)

- \( D: \text{oldR0} = 1101 \)
- \( S: \text{oldR0} \ll 1 = 1010 \)
- \( I: R_0 \ll 1 = 0110 \)
- \( M: (\text{oldR1} \ll 1) \lor PM(T) = 1011 \)

**Alignment Found @ Location=2**

---

**Text[3]:** CGTGA

**oldR0:** 1110
**oldR1:** 1100

\( R_0 = (\text{oldR0} \ll 1) \lor PM(G) \)

- \( D: \text{oldR0} = 1110 \)
- \( S: \text{oldR0} \ll 1 = 1110 \)
- \( I: R_0 \ll 1 = 1100 \)
- \( M: (\text{oldR1} \ll 1) \lor PM(G) = 1011 \)

**Alignment Found @ Location=3**

---

**Text[4]:** CGTGA

**oldR0:** 1111
**oldR1:** 1111

\( R_0 = (\text{oldR0} \ll 1) \lor PM(A) \)

- \( D: \text{oldR0} = 1111 \)
- \( S: \text{oldR0} \ll 1 = 1110 \)
- \( I: R_0 \ll 1 = 1110 \)
- \( M: (\text{oldR1} \ll 1) \lor PM(A) = 1100 \)

**Alignment Found @ Location=4**

---

**Text[5]:** CGTGA

**oldR0:** 1111
**oldR1:** 0000

\( R_0 = (\text{oldR0} \ll 1) \lor PM(C) \)

- \( D: \text{oldR0} = 1111 \)
- \( S: \text{oldR0} \ll 1 = 1110 \)
- \( I: R_0 \ll 1 = 1110 \)
- \( M: (\text{oldR1} \ll 1) \lor PM(C) = 0111 \)

**Alignment Found @ Location=5**
# 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

**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
  - Approximate string matching (ASM) acceleration framework based on the Bitap algorithm

- We overcome the *five limitations* that hinder Bitap’s use in GSA:
  - 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
GenASM Algorithm

- **GenASM-DC Algorithm:**
  - Modified Bitap for Distance Calculation
  - Extended for efficient long read support
  - Besides bit-parallelism that Bitap has, extended for parallelism:
    - Loop unrolling
    - Text-level parallelism

- **GenASM-TB Algorithm:**
  - Novel Bitap-compatible TraceBack algorithm
  - Walks through the intermediate bitvectors (match, deletion, substitution, insertion) generated by GenASM-DC
  - Follows a divide-and-conquer approach to decrease the memory footprint
Loop Unrolling in GenASM-DC

<table>
<thead>
<tr>
<th>Cycle#</th>
<th>Thread₁ ( R₀/₁/₂/... )</th>
<th>Thread₂ ( R₀/₄ )</th>
<th>Thread₃ ( R₁/₅ )</th>
<th>Thread₄ ( R₂/₆ )</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>T₀-R₀</td>
<td>T₀-R₀</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>T₁-R₀</td>
<td>T₀-R₁</td>
<td>-</td>
</tr>
<tr>
<td>#8</td>
<td>T₀-R₇</td>
<td>T₂-R₀</td>
<td>T₁-R₁</td>
<td>T₀-R₂</td>
</tr>
<tr>
<td>#9</td>
<td>T₁-R₀</td>
<td>T₃-R₀</td>
<td>T₂-R₁</td>
<td>T₁-R₂</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>T₀-R₄</td>
<td>T₃-R₁</td>
<td>T₂-R₂</td>
</tr>
<tr>
<td>#16</td>
<td>T₁-R₇</td>
<td>T₀-R₄</td>
<td>T₀-R₅</td>
<td>T₁-R₃</td>
</tr>
<tr>
<td>#17</td>
<td>T₂-R₀</td>
<td>T₁-R₄</td>
<td>T₁-R₅</td>
<td>T₀-R₆</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>T₀-R₄</td>
<td>T₂-R₅</td>
<td>T₁-R₆</td>
</tr>
<tr>
<td>#24</td>
<td>T₂-R₇</td>
<td>T₃-R₄</td>
<td>T₂-R₅</td>
<td>T₀-R₇</td>
</tr>
<tr>
<td>#25</td>
<td>T₃-R₀</td>
<td>-</td>
<td>T₃-R₅</td>
<td>T₂-R₆</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>-</td>
<td>T₀-R₅</td>
<td>T₁-R₇</td>
</tr>
<tr>
<td>#32</td>
<td>T₃-R₇</td>
<td>-</td>
<td>-</td>
<td>T₃-R₆</td>
</tr>
<tr>
<td>#33</td>
<td>T₃-R₇</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

- data **written to memory**
- data **read from memory**

- target cell \( R_d \)
- cells target cell depends on \( \text{oldR}_d, R_{d-1}, \text{oldR}_{d-1} \)
Traceback Example with GenASM-TB

(a) Deletion Example (Text Location=0)

Text[0]: C
(R0- : ..... )
(R1-M : 0111)
Match(C) <3,0,1>

Text[1]: G
(R0- : ..... )
(R1-D : 1011)
Del(-) <2,1,1>

Text[2]: T
(R0-M : 1011)
(R1- : ..... )
Match(T) <2,2,0>

Text[3]: G
(R0-M : 1101)
(R1- : ..... )
Match(G) <1,3,0>

Text[4]: A
(R0-M : 1110)
(R1- : ..... )
Match(A) <0,4,0>

(b) Substitution Example (Text Location=1)

Text[1]: G
(R0- : ..... )
(R1-S : 0110)
Subs(C) <3,1,1>

Text[2]: T
(R0-M : 1011)
(R1- : ..... )
Match(T) <2,2,0>

Text[3]: G
(R0-M : 1101)
(R1- : ..... )
Match(G) <1,3,0>

Text[4]: A
(R0-M : 1110)
(R1- : ..... )
Match(A) <0,4,0>

(c) Insertion Example (Text Location=2)

[-]
(R0- : ..... )
(R1-I : 0110)
Ins(C) <3,2,1>

Text[2]: T
(R0-M : 1011)
(R1- : ..... )
Match(T) <2,2,0>

Text[3]: G
(R0-M : 1101)
(R1- : ..... )
Match(G) <1,3,0>

Text[4]: A
(R0-M : 1110)
(R1- : ..... )
Match(A) <0,4,0>
GenASM Hardware Design

GenASM-DC: generates bitvectors and performs edit Distance Calculation

GenASM-TB: performs TraceBack and assembles the optimal alignment
GenASM Hardware Design

Main Memory

Host CPU

DC-SRAM

GenASM-DC

GenASM-TB

GenASM-DC: generates bitvectors and performs edit Distance Calculation

GenASM-TB: performs TraceBack and assembles the optimal alignment

1. reference & 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
GenASM Hardware Design

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
Use Cases of GenASM

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

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

(3) Edit Distance Calculation
   - Measure the similarity or distance between two sequences

We also discuss other possible use cases of GenASM in our paper:
   - Read-to-read overlap finding, hash-table based indexing, whole genome alignment, generic text search
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(^1) BWA-MEM(^2)</td>
<td>GACT (Darwin)(^3) SillaX (GenAx)(^4)</td>
</tr>
<tr>
<td><strong>Pre-Alignment Filtering</strong></td>
<td>–</td>
<td>Shouji(^5)</td>
</tr>
<tr>
<td><strong>Edit Distance Calculation</strong></td>
<td>Edlib(^6)</td>
<td>ASAP(^7)</td>
</tr>
</tbody>
</table>

Evaluation Methodology (cont’d.)

- 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
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 process:
  - Both GenASM-DC and GenASM-TB operate @ 1GHz

![Pie charts showing area and power consumption for different components.

Total (1 vault): 0.334 mm², 0.101 W
Total (32 vaults): 10.69 mm², 3.23 W
% of a Xeon CPU core: 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 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
   - Find the **optimal alignment** of how reads map to candidate reference regions

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

(3) Edit Distance Calculation
   - 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
Key Results – Use Case 2

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

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

3) Edit Distance Calculation
   - Measure the similarity or distance between two sequences
Key Results – Use Case 2

- Compared to Shouji:
  - 3.7× speedup
  - 1.7× less power consumption
  - False accept rate of 0.02% for GenASM vs. 4% for Shouji
  - False reject rate of 0% for both GenASM and Shouji

GenASM is more efficient in terms of both speed and power consumption, while significantly improving the accuracy of pre-alignment filtering.
Key Results – Use Case 3

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

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

(3) Edit Distance Calculation
   - Measure the similarity or distance between two sequences
**Key Results – Use Case 3**

<table>
<thead>
<tr>
<th>Similarity between two sequences</th>
<th>Edlib (100 Kbp)</th>
<th>GenASM (100 Kbp)</th>
<th>Edlib (1 Mbp)</th>
<th>GenASM (1 Mbp)</th>
</tr>
</thead>
<tbody>
<tr>
<td>99%</td>
<td>146×</td>
<td>627×</td>
<td>12501×</td>
<td>1458×</td>
</tr>
<tr>
<td>97%</td>
<td>146×</td>
<td>627×</td>
<td>12501×</td>
<td>1458×</td>
</tr>
<tr>
<td>94%</td>
<td>146×</td>
<td>627×</td>
<td>12501×</td>
<td>1458×</td>
</tr>
<tr>
<td>90%</td>
<td>146×</td>
<td>627×</td>
<td>12501×</td>
<td>1458×</td>
</tr>
<tr>
<td>80%</td>
<td>146×</td>
<td>627×</td>
<td>12501×</td>
<td>1458×</td>
</tr>
<tr>
<td>70%</td>
<td>146×</td>
<td>627×</td>
<td>12501×</td>
<td>1458×</td>
</tr>
<tr>
<td>60%</td>
<td>146×</td>
<td>627×</td>
<td>12501×</td>
<td>1458×</td>
</tr>
</tbody>
</table>

**SW**

GenASM provides **146 – 1458×** and **627 – 12501×** speedup, while reducing power consumption by **548×** and **582×** for 100Kbp and 1Mbp sequences, respectively, compared to Edlib.

**HW**

GenASM provides **9.3 – 400×** speedup over ASAP, while consuming **67×** less power.
Key Results – Summary

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

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

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

- **Problem:**
  - Genome sequence analysis is bottlenecked by the computational power and memory bandwidth limitations of existing systems
  - This bottleneck is particularly an issue for approximate string matching

- **Key Contributions:**
  - GenASM: An approximate string matching (ASM) acceleration framework to accelerate multiple steps of genome sequence analysis
    - *First* to enhance and accelerate Bitap for ASM with genomic sequences
    - *Co-design* of our modified scalable and memory-efficient algorithms with low-power and area-efficient hardware accelerators
    - Evaluation of three different use cases: read alignment, pre-alignment filtering, edit distance calculation

- **Key Results:** GenASM is significantly more efficient for all the three use cases (in terms of throughput and throughput per unit power) than state-of-the-art software and hardware baselines
GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis

GenASM – GitHub Page

https://github.com/CMU-SAFARI/GenASM

GenASM: Approximate String Matching (ASM) Acceleration Framework for Genome Sequence Analysis

GenASM is an approximate string matching (ASM) acceleration framework for genome sequence analysis. GenASM is a fast, efficient, and flexible framework for both short and long reads, which can be used to accelerate multiple steps of the genome sequence analysis pipeline. We base GenASM upon the Bitap algorithm. Bitap uses only fast and simple bitwise operations to perform approximate string matching. To our knowledge, GenASM is the first work that enhances and accelerates Bitap.
Backup Slides
(Sequencing)
Genome Sequencing

Sample Collection

Large DNA molecule

Preparation

Chopped DNA fragments

Sequencing

Sequenced reads

Genome Sequence Analysis

Joël Lindegger
Sequencing Technologies

**Short reads:** a few hundred base pairs and error rate of $\sim0.1\%$

**Long reads:** thousands to millions of base pairs and error rate of 5–10\%
Current State of Sequencing (cont’d.)

Cost per Human Genome

Moore’s Law

*From NIH (https://www.genome.gov/about-genomics/fact-sheets/DNA-Sequencing-Costs-Data)
Current State of Sequencing (cont’d.)

Computation is a bottleneck!

*From NIH (https://www.genome.gov/about-genomics/fact-sheets/DNA-Sequencing-Costs-Data)
**Genome Sequence Analysis**

Read Mapping, method of aligning the reads against the reference genome in order to detect matches and variations.

De novo Assembly, method of merging the reads in order to construct the original sequence.
Read Mapping Pipeline

Indexing
(Pre-processing step to generate index of reference)

Seeding
(Query the index)

Pre-Alignment Filtering
(Filter out dissimilar sequences)

Read Alignment
(Perform distance/score calculation & traceback)

Reference genome → Hash-table based index

Reads → Potential mapping locations

Reference segment → Remaining potential mapping locations

Query read → Optimal alignment
With the emergence of long read sequencing technologies, de novo assembly becomes a promising way of constructing the original genome.
Our Contributions

- Analyze the tools in multiple dimensions: accuracy, performance, memory usage, and scalability

- Reveal new bottlenecks and trade-offs

- First study on bottleneck analysis of nanopore sequence analysis pipeline on real machines

- Provide guidelines for practitioners

- Provide guidelines for tool developers
Key Findings

- **Laptops** are becoming a popular platform for running genome assembly tools, as the *portability* of a laptop makes it a good fit for *in-field analysis*
  - Greater memory constraints
  - Lower computational power
  - Limited battery life

- **Memory usage** is an important factor that greatly affects the performance and the usability of the tool
  - Data structure choices that increase the memory requirements
  - Algorithms that are not cache-efficient
  - Not keeping memory usage in check with the number of threads

- **Scalability of the tool** with the number of cores is an important requirement. However, parallelizing the tool can *increase the memory usage*
  - Not dividing the input data into batches
  - Not limiting the memory usage of each thread
  - Dividing the dataset instead of the computation between simultaneous threads
Key Findings

- **Goal 1:** High-performance and low-power
- **Goal 2:** Memory-efficient
- **Goal 3:** Scalable/highly-parallel
Nanopore Sequencing & Tools

Nanopore Sequencing Technology and Tools for Genome Assembly: Computational Analysis of the Current State, Bottlenecks and Future Directions

Damla Senol Cali¹,*, Jeremie S. Kim¹,³, Saugata Ghose¹, Can Alkan²* and Onur Mutlu³,¹*

¹Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, USA
²Department of Computer Engineering, Bilkent University, Bilkent, Ankara, Turkey
³Department of Computer Science, Systems Group, ETH Zürich, Zürich, Switzerland