Accelerating Genome Sequence Analysis via Efficient Hardware/Algorithm Co-Design

Damla Senol Cali, Ph.D.
https://damlasenolcali.github.io/
damlasenolcali@gmail.com

Staff Software Engineer, Hardware Acceleration
bionano

BIO-Arch Workshop @ RECOMB 2023
April 14, 2023

Carnegie Mellon  SAFARI  ETH Zürich
Genome Sequencing

Sample Collection

Large DNA molecule

Chopped DNA fragments

Preparation

Sequencing

Sequenced reads

Genome Sequence Analysis
Current State of Sequencing

*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](https://www.genome.gov/about-genomics/fact-sheets/DNA-Sequencing-Costs-Data))
Our Goal & Approach

- **Our Goal:**
  Accelerating genome sequence analysis by **efficient** hardware/algorithm co-design

- **Our Approach:**
  1. Analyze the *multiple steps* and the *associated tools* in the genome sequence analysis pipeline,
  2. Expose the *tradeoffs* between accuracy, performance, memory usage and scalability, and
  3. Co-design *fast and efficient algorithms* along with scalable and energy-efficient customized hardware accelerators for the key bottleneck steps of the pipeline
### Research Contributions

<table>
<thead>
<tr>
<th>Project</th>
<th>Description</th>
<th>Status</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>BitMAc</td>
<td>FPGA-based near-memory acceleration of bitvector-based sequence alignment</td>
<td>Ongoing</td>
<td></td>
</tr>
<tr>
<td>GenASM</td>
<td>Approximate string matching framework for genome sequence analysis</td>
<td></td>
<td>[MICRO 2020]</td>
</tr>
<tr>
<td>SeGraM</td>
<td>Universal genomic mapping accelerator for both sequence-to-graph and sequence-to-sequence mapping</td>
<td></td>
<td>[ISCA 2022]</td>
</tr>
<tr>
<td>Bottleneck analysis of genome assembly pipeline for long reads</td>
<td></td>
<td></td>
<td>[Briefings in Bioinformatics, 2018]</td>
</tr>
</tbody>
</table>
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

Key Findings

Goal 1: High-performance and low-power

Goal 2: Memory-efficient

Goal 3: Scalable/highly-parallel
Research Contributions

Bottleneck analysis of genome assembly pipeline for long reads

[Briefings in Bioinformatics, 2018]

**GenASM**: Approximate string matching framework for genome sequence analysis

[MICRO 2020]

BitMAc: FPGA-based near-memory acceleration of bitvector-based sequence alignment

[Ongoing]

SeGraM: Universal genomic mapping accelerator for both sequence-to-graph and sequence-to-sequence mapping

[ISCA 2022]
Read Mapping Pipeline

1. **Indexing**
   - (Pre-processing step to generate index of reference)
   - Hash-table based index

2. **Seeding**
   - (Query the index)
   - Potential mapping locations

3. **Pre-Alignment Filtering**
   - (Filter out dissimilar sequences)
   - Remaining potential mapping locations

4. **Read Alignment**
   - (Perform distance/score calculation & traceback)
   - Optimal alignment
GSA with Read Mapping

- **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
Approximate String Matching

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

  **Reference:** AAAATGTTTAGTGCCTACTTG
  **Read:** AAAATGTTTACTGCTACTTGT
  - 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 differences)
    - 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

A Bitap version 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

Limitations of Bitap

1) Data Dependency Between Iterations:
   - Two-level data dependency forces the consecutive iterations to take place sequentially
Step 2: Edit Distance Calculation

For each character of the text (char):

Copy previous R bitvectors as oldR

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

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

- deletion = oldR[\(d-1\)]
- substitution = oldR[\(d-1\)] \ll 1
- insertion = R[\(d-1\)] \ll 1
- match = (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.
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 \( d = 1 \ldots k \):
    
    \[
    \begin{align*}
    \text{deletion} & = \text{oldR}[d-1] \\
    \text{substitution} & = \text{oldR}[d-1] \ll 1 \\
    \text{insertion} & = R[d-1] \ll 1 \\
    \text{match} & = (\text{oldR}[d] \ll 1) | \text{PM}[\text{char}] \\
    \end{align*}
    \]

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

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

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

For each character of the text (char):

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

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

- deletion = oldR\[d-1\]
- substitution = oldR\[d-1\] \ll 1
- insertion = R\[d-1\] \ll 1
- match = (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

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 Hardware Design

GenASM-DC: generates bitvectors and performs edit Distance Calculation

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

1. Host CPU
   - Reference text & query locations

2. Main Memory
   - Reference text & query pattern

3. GenASM-DC Accelerator
   - Sub-text & sub-pattern

4. Generate bitvectors

5. Write bitvectors

6. Read bitvectors

7. Find the traceback output

GenASM-DC: generates bitvectors and performs edit Distance Calculation

GenASM-TB: performs TraceBack and assembles the optimal alignment
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
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&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>

---

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
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 chart showing area and power comparison]

<table>
<thead>
<tr>
<th>Component</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>0.009</td>
</tr>
<tr>
<td>DC-SRAM (8 KB)</td>
<td>0.013</td>
<td>0.004</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%
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 (cont’d.)

(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), while significantly improving the accuracy of pre-alignment filtering

(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)
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

- **GenASM**: *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

- GenASM supports three different use cases: read alignment, pre-alignment filtering, edit distance calculation

- GenASM is *significantly more efficient* for all the three use cases than state-of-the-art software and hardware baselines
GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis

Damla Senol Cali†, Gurpreet S. Kalsi‡, Zülaı BingölVen Can Firtina◊, Lavanya Subramanian‡, Jeremie S. Kim◊, Rachata Ausavarungnirun◊, Mohammed Alser◊, Juan Gomez-Luna◊, Amirali Boroumand‡, Anant Nori◊, Allison Scibisz†, Sreenivas Subramoney◊, Can AlkanVen, Saugata Ghose*†, Onur Mutlu*Ven

†Carnegie Mellon University  ‡Processor Architecture Research Lab, Intel Labs  VenBilkent University  ◊ETH Zürich  *Facebook  ◁King Mongkut’s University of Technology North Bangkok  *University of Illinois at Urbana-Champaign

MICRO’20 Paper

MICRO’20 Talk
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.

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

Bottleneck analysis of genome assembly pipeline for long reads

[Briefings in Bioinformatics, 2018]

GenASM: Approximate string matching framework for genome sequence analysis

[MICRO 2020]

BitMAc: FPGA-based near-memory acceleration of bitvector-based sequence alignment

[Ongoing]

SeGraM: Universal genomic mapping accelerator for both sequence-to-graph and sequence-to-sequence mapping

[ISCA 2022]
Mapping the reads to a reference genome (i.e., *read mapping*) is a critical step in genome sequence analysis. Sequence-to-Sequence (S2S) Mapping results in notable quality improvements.

However, it is a more difficult computational problem, with no prior hardware design.

Linear Reference: ACGTACGT
Read: ACGG
Alternative Sequence: ACGGACGT
Alternative Sequence: ACGTTACGT
Alternative Sequence: ACG‒ACGT

Graph-based Reference:

Read: ACGG

Sequence-to-Graph (S2G) Mapping
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: ACCTACGT
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: ACAGACGT
Sequence #3: ACGTGACGT
Sequence #4: ACAGACGT
Sequence-to-Graph Mapping Pipeline

Pre-Processing Steps (Offline)

0.1 Genome Graph Construction
   (construct the graph using a linear reference genome and variations)

0.2 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)

Optimal alignment between read & subgraph

Steps:

- Linear reference genome
- Known genetic variations
- Reads from sequenced genome
- Genome graph
- Hash-table-based index (of graph nodes)
- Candidate mapping locations (subgraphs)
- Remaining candidate mapping locations (subgraphs)
- Optimal alignment between read & subgraph
S\textsubscript{2}S vs. S\textsubscript{2}G Alignment

**Single linear reference**

\[
\begin{array}{cccccc}
A & C & G & T & A & C \\
\end{array}
\]

**Query read**

\[
\begin{array}{cccc}
A & C & G & T \\
\end{array}
\]

Sequence-to-Sequence (S\textsubscript{2}S) 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
SeGraM: Universal Genomic 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 support both sequence-to-graph and sequence-to-sequence mapping, for both short and long reads

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

- We base SeGraM upon a **minimizer-based seeding algorithm** and a **novel bitvector-based alignment algorithm**

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

Main Memory (graph-based reference & index)

MinSeed (MS)
- Minimizer Scratchpad
- Find Minimizers
- Read Scratchpad
- Find Candidate Seed Regions

BitAlign (BA)
- Input Scratchpad
- Generate Bitvectors
- Bitvector Scratchpad
- Perform Traceback
- Hop Queues

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*
- 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

SeGraM Module (1 x per HBM2E stack)
Use Cases of SeGraM

(1) Sequence-to-Graph Mapping

(2) Sequence-to-Graph Alignment

(3) Sequence-to-Sequence Alignment

(4) Seeding
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
## Key Results – Area & Power

- 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><strong>BitAlign – Edit Distance Calculation Logic with Hop Queue Registers</strong></td>
<td><strong>0.393</strong></td>
<td><strong>378.0</strong></td>
</tr>
<tr>
<td><strong>BitAlign – Traceback Logic</strong></td>
<td><strong>0.020</strong></td>
<td><strong>2.7</strong></td>
</tr>
<tr>
<td>Input Scratchpad (24 kB)</td>
<td>0.033</td>
<td>13.3</td>
</tr>
<tr>
<td><strong>Bitvector Scratchpads (128 kB)</strong></td>
<td><strong>0.329</strong></td>
<td><strong>316.2</strong></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>--</td>
<td><strong>3.8 W</strong></td>
</tr>
</tbody>
</table>
Key Results (cont’d.)

(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)
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
Summary of SeGraM

- **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: 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 Alser²
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 – Source Code & Datasets

Running SeGraM

Call the following two functions in src/graph.c and src/align.c files in your C code, respectively, or update the existing main() function in src/main.c file:

```c
struct SeqNode* generateGraphFromGFA(char *filename, int *numNodes, int *numEdges);
bitalign_aligner(struct SeqNode *nodes, char *pattern, int startNode, int startOffset, int endNode, int endOffset);
```

Datasets

We evaluate SeGraM using the latest major release of the human genome assembly, GRCh38, as the starting reference genome. To incorporate known genetic variations and thus form a genome graph, we use 7 VCF files for HG001-007 from the GIAB project (v3.3.2). Across the 24 graphs generated (one for each chromosome: 1-22, X, Y), in total, we have 20.4M nodes, 27.9 M edges, 3.1B sequence characters, and 7.1M variations.

For the read datasets, we generate four sets of long reads (i.e., PacBio and ONT datasets) using PBSIM2 and three sets of short reads (i.e., Illumina datasets) using Mason. For the PacBio and ONT datasets, we have reads of length 10kbp, each simulated with 5% and 10% error rates. The Illumina datasets have reads of length 100bp, 150bp, and 250bp, each simulated with a 1% error rate. Each dataset has 10,000 reads.

All our prepared datasets can be downloaded from this link. The unzipped directory has the following structure:

```
- datasets
  - graphs
    - gfa_files: our graph files (1 for each chromosome: 1-22, X, Y) in GFA format
    - vg_files: our graph files (1 for each chromosome: 1-22, X, Y) in VG format
    - gfa_files: our graph files (1 for each chromosome: 1-22, X, Y) in FASTA format (i.e., each node is
      - reads
        - illumina_reads: our simulated short reads with 1% error rate and 100/150/250bp length
        - pacbio_ont_reads: our simulated 10k-length long reads with 5% and 10% error rates and different
```

https://github.com/CMU-SAFARI/SeGraM
Accelerating Genome Sequence Analysis via Efficient Hardware/Algorithm Co-Design

Damla Senol Cali, Ph.D.
https://damlasenolcali.github.io/
damlasenolcali@gmail.com

Staff Software Engineer, Hardware Acceleration
bionano

BIO-Arch Workshop @ RECOMB 2023
April 14, 2023
Backup Slides

(BiB Paper)
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.
With the emergence of long read sequencing technologies, de novo assembly becomes a promising way of constructing the original genome.

Genome Assembly Pipeline Using Long Reads

- **Basecalling**
  - (Translates signal data into bases: A, C, G, T)

- **Read-to-Read Overlap Finding**
  - (Finds pairwise read alignments for each pair of read)

- **Assembly**
  - (Traverses the overlap graph & constructs the draft assembly)

- **Read Mapping**
  - (Maps the reads to the draft assembly)

- **Polishing**
  - (Polishes the draft assembly & increases the accuracy)

DNA reads → Overlaps → Draft assembly → Mappings of reads against draft assembly

Raw signal data → DNA reads

Assembly → Improved assembly
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
Backup Slides
(GenASM)
Example for the Bitap Algorithm

Text Region: CGTGA
Query Pattern: CTGA
Edit Distance Threshold (k): 1

Text[4]: CGTGA
oldR0 = 1111
oldR1 = 1111
R0 = (oldR0 << 1) | PM(A) = 1110
D : oldR0 = 1111
S : oldR0 << 1 = 1110
I : R0 << 1 = 1100
M : (oldR1 << 1) | PM(A) = 1110 = D & S & I & M
R1 = 1111

Text[3]: CGTGA
oldR0 = 1110
oldR1 = 1100
R0 = (oldR0 << 1) | PM(G) = 1101
D : oldR0 = 1110
S : oldR0 << 1 = 1100
I : R0 << 1 = 1010
M : (oldR1 << 1) | PM(G) = 1101 = D & S & I & M

Text[2]: CGTGA
oldR0 = 1101
oldR1 = 1000
R0 = (oldR0 << 1) | PM(T) = 1011
D : oldR0 = 1101
S : oldR0 << 1 = 1010
I : R0 << 1 = 0110
M : (oldR1 << 1) | PM(T) = 1011 = D & S & I & M

Text[1]: CGTGA
oldR0 = 1011
oldR1 = 0000
R0 = (oldR0 << 1) | PM(G) = 1111
D : oldR0 = 1011
S : oldR0 << 1 = 0110
I : R0 << 1 = 1100
M : (oldR1 << 1) | PM(G) = 1101 = D & S & I & M

Text[0]: CGTGA
oldR0 = 1111
oldR1 = 0000
R0 = (oldR0 << 1) | PM(C) = 1111
D : oldR0 = 1111
S : oldR0 << 1 = 1110
I : R0 << 1 = 1110
M : (oldR1 << 1) | PM(C) = 0111 = D & S & I & M

Alignment Found @ Location=2
Alignment Found @ Location=1
Alignment Found @ Location=0
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 1: Thread Execution Schedule

<table>
<thead>
<tr>
<th>Cycle #</th>
<th>Thread 1 (R0/1/2/..)</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>T0-R0</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>#8</td>
<td>T0-R7</td>
</tr>
<tr>
<td>#9</td>
<td>T1-R0</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>#16</td>
<td>T1-R7</td>
</tr>
<tr>
<td>#17</td>
<td>T2-R0</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>#24</td>
<td>T2-R7</td>
</tr>
<tr>
<td>#25</td>
<td>T3-R0</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>#32</td>
<td>T3-R7</td>
</tr>
</tbody>
</table>

### Table 2: Thread Data Execution

<table>
<thead>
<tr>
<th>Cycle #</th>
<th>Thread 1 (R0/4)</th>
<th>Thread 2 (R1/5)</th>
<th>Thread 3 (R2/6)</th>
<th>Thread 4 (R3/7)</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>T0-R0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#2</td>
<td>T1-R0</td>
<td>T0-R1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#3</td>
<td>T2-R0</td>
<td>T1-R1</td>
<td>T0-R2</td>
<td></td>
</tr>
<tr>
<td>#4</td>
<td>T3-R0</td>
<td>T2-R1</td>
<td>T1-R2</td>
<td>T0-R3</td>
</tr>
<tr>
<td>#5</td>
<td>T0-R4</td>
<td>T3-R1</td>
<td>T2-R2</td>
<td>T1-R3</td>
</tr>
<tr>
<td>#6</td>
<td>T1-R4</td>
<td>T0-R5</td>
<td>T3-R2</td>
<td>T2-R3</td>
</tr>
<tr>
<td>#7</td>
<td>T2-R4</td>
<td>T1-R5</td>
<td>T0-R6</td>
<td>T3-R3</td>
</tr>
<tr>
<td>#8</td>
<td>T3-R4</td>
<td>T2-R5</td>
<td>T1-R6</td>
<td>T0-R7</td>
</tr>
<tr>
<td>#9</td>
<td></td>
<td>T3-R5</td>
<td>T2-R6</td>
<td>T1-R7</td>
</tr>
<tr>
<td>#10</td>
<td></td>
<td></td>
<td>T3-R6</td>
<td>T2-R7</td>
</tr>
<tr>
<td>#11</td>
<td></td>
<td></td>
<td></td>
<td>T3-R7</td>
</tr>
</tbody>
</table>

- **Data written to memory**
- **Data read from memory**

<table>
<thead>
<tr>
<th>Target cell (R_d)</th>
<th>Cells target cell depends on (oldR_{d-1}, R_{d-1}, oldR_{d-1})</th>
</tr>
</thead>
</table>

---

Damla Senol Cali

SAFARI
Traceback Example with GenASM-TB

### Deletion Example (Text Location=0)

<table>
<thead>
<tr>
<th>Text[0]: C</th>
<th>Text[1]: G</th>
<th>Text[2]: T</th>
<th>Text[3]: G</th>
<th>Text[4]: A</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1-M : 0111</td>
<td>R1-D : 1011</td>
<td>R1- : .....</td>
<td>R1- : .....</td>
<td>R1- : .....</td>
</tr>
<tr>
<td>Match(C)</td>
<td>Del(-)</td>
<td>Match(T)</td>
<td>Match(G)</td>
<td>Match(A)</td>
</tr>
<tr>
<td>&lt;3,0,1&gt;</td>
<td>&lt;2,1,1&gt;</td>
<td>&lt;2,2,0&gt;</td>
<td>&lt;1,3,0&gt;</td>
<td>&lt;0,4,0&gt;</td>
</tr>
</tbody>
</table>

### Substitution Example (Text Location=1)

<table>
<thead>
<tr>
<th>Text[1]: G</th>
<th>Text[2]: T</th>
<th>Text[3]: G</th>
<th>Text[4]: A</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1-S : 0110</td>
<td>R1- : .....</td>
<td>R1- : .....</td>
<td>R1- : .....</td>
</tr>
<tr>
<td>Subs(C)</td>
<td>Match(T)</td>
<td>Match(G)</td>
<td>Match(A)</td>
</tr>
<tr>
<td>&lt;3,1,1&gt;</td>
<td>&lt;2,2,0&gt;</td>
<td>&lt;1,3,0&gt;</td>
<td>&lt;0,4,0&gt;</td>
</tr>
</tbody>
</table>

### Insertion Example (Text Location=2)

<table>
<thead>
<tr>
<th>Text[-]</th>
<th>Text[2]: T</th>
<th>Text[3]: G</th>
<th>Text[4]: A</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1-I : 0110</td>
<td>R1- : .....</td>
<td>R1- : .....</td>
<td>R1- : .....</td>
</tr>
<tr>
<td>Ins(C)</td>
<td>Match(T)</td>
<td>Match(G)</td>
<td>Match(A)</td>
</tr>
<tr>
<td>&lt;3,2,1&gt;</td>
<td>&lt;2,2,0&gt;</td>
<td>&lt;1,3,0&gt;</td>
<td>&lt;0,4,0&gt;</td>
</tr>
</tbody>
</table>
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 $648\times$ and $116\times$ speedup over 12-thread runs of BWA-MEM and Minimap2, while reducing power consumption by $34\times$ and $37\times$
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)

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

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
  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
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

**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.

**GenASM** provides **9.3 – 400×** speedup over ASAP, while consuming **67× less power**.
Sources of Improvement in GenASM

- **Very simple computations** GenASM performs

- **Divide-and-conquer approach** we follow, which makes our design efficient for both short and long reads despite their different error profiles

- **Very high degree of parallelism** obtained with the help of:
  - Specialized compute units, dedicated SRAMs for both GenASM-DC and GenASM-TB, and
  - Vault-level parallelism provided by processing in the logic layer of 3D-stacked memory
Backup Slides
(SeGraM)
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

Hash table size (GB) vs Max number of minimizers in a bucket

Number of Buckets:
- $2^{28}$
- $2^{27}$
- $2^{26}$
- $2^{25}$
- $2^{24}$
- $2^{23}$
- $2^{22}$
- $2^{21}$

Memory Footprint (GB):
- 35
- 30
- 25
- 20
- 15
- 10
- 5
- 0

Number of Minimizers:
- 100
- 80
- 60
- 40
- 20
- 0

Graph shows the relationship between the number of buckets and memory footprint, as well as the maximum number of minimizers in a bucket.
**Minimizers**

<table>
<thead>
<tr>
<th>Position</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence</td>
<td>A</td>
<td>G</td>
<td>T</td>
<td>A</td>
<td>G</td>
<td>C</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>$k$-mer$_1$</td>
<td>A</td>
<td>G</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$k$-mer$_2$</td>
<td>G</td>
<td>T</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$k$-mer$_3$</td>
<td>T</td>
<td>A</td>
<td>G</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$k$-mer$_4$</td>
<td>A</td>
<td>G</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$k$-mer$_5$</td>
<td></td>
<td></td>
<td></td>
<td>G</td>
<td>C</td>
<td>A</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The table shows a sequence of nucleotides (A, G, T, C) and different $k$-mers. The lexicographically smallest $k$-mer is indicated with a red arrow.
MinSeed – Region Calculation

Diagram:

- **Minimizer**: Node labeled "minimizer" with inputs and outputs labeled 0, a, b, m−1.
- **Seed**: Node labeled "seed" with inputs labeled c, d.
- **Left-extension**: Path from x to c labeled "a*(1+E)."
- **Right-extension**: Path from d to y labeled 
  \[(m−b−1)*(1+E)\].

Annotations:
- **Query Read**: "query read" beside the right-extension.
- **Graph-based Reference**: "graph-based reference" beside the left-extension.
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: \( n \leftarrow \text{length of linearized reference subgraph} \)
2: \( m \leftarrow \text{length of query read} \)
3: \( \text{PM} \leftarrow \text{genPatternBitmasks(query-read)} \) \hspace{1cm} \( \triangleright \text{pre-process the query read} \)
4: 
5: \( \text{allR}[n][d] \leftarrow 111 \ldots 111 \) \hspace{1cm} \( \triangleright \text{init R}[d] bitvectors for all characters with 1s} \)
6: 
7: for \( i \) in \( (n-1):-1:0 \) do \hspace{1cm} \( \triangleright \text{iterate over each subgraph node} \)
8: \( \text{curChar} \leftarrow \text{subgraph-nodes}[i].\text{char} \)
9: \( \text{curPM} \leftarrow \text{PM[curChar]} \) \hspace{1cm} \( \triangleright \text{retrieve the pattern bitmask} \)
10: 
11: \( \text{R}0 \leftarrow 111 \ldots 111 \) \hspace{1cm} \( \triangleright \text{status bitvector for exact match} \)
12: for \( j \) in \( \text{subgraph-nodes}[i].\text{successors} \) do
13: \( \text{R}0 \leftarrow ((\text{R}[j][0] \ll 1) | \text{curPM}) \& \text{R}0 \) \hspace{1cm} \( \triangleright \text{exact match calculation} \)
14: \( \text{allR}[i][0] \leftarrow \text{R}0 \)
15: 
16: for \( d \) in \( 1:k \) do \hspace{1cm} \( \triangleright \text{insertion} \)
17: \( \text{I} \leftarrow (\text{allR}[i][d-1] \ll 1) \)
18: \( \text{R}d \leftarrow \text{I} \) \hspace{1cm} \( \triangleright \text{status bitvector for } d \text{ errors} \)
19: for \( j \) in \( \text{subgraph-nodes}[i].\text{successors} \) do
20: \( \text{D} \leftarrow \text{allR}[j][d-1] \) \hspace{1cm} \( \triangleright \text{deletion} \)
21: \( \text{S} \leftarrow \text{allR}[j][d-1] \ll 1 \) \hspace{1cm} \( \triangleright \text{substitution} \)
22: \( \text{M} \leftarrow (\text{allR}[j][d] \ll 1) | \text{curPM} \) \hspace{1cm} \( \triangleright \text{match} \)
23: \( \text{R}d \leftarrow \text{D} \& \text{S} \& \text{M} \& \text{R}d \)
24: \( \text{allR}[i][d] \leftarrow \text{R}d \)
25: \( \text{<editDist, CIGARstr>} \leftarrow \text{traceback(allR, subgraph, query-read)} \)
BitAlign – Hop Length Selection

The graph shows the fraction of total hops (%) as a function of the hop limit (number of nodes). As the hop limit increases, the fraction of total hops increases, approaching 100%.
## 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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</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>3</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>4</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>5</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>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</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>1</td>
<td></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>1</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>1</td>
<td></td>
</tr>
</tbody>
</table>
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*)
SeGraM provides 5.9× and 3.9× throughput improvement over GraphAligner and vg, while reducing the power consumption by 4.1× and 4.4×.
Key Results – SeGraM with Short Reads

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
Accelerating Genome Sequence Analysis via Efficient Hardware/Algorithm Co-Design

Damla Senol Cali, Ph.D.
https://damlasenolcali.github.io/
damlasenolcali@gmail.com

Staff Software Engineer, Hardware Acceleration
bionano®

BIO-Arch Workshop @ RECOMB 2023
April 14, 2023