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

Damla Senol Cali, Ph.D.

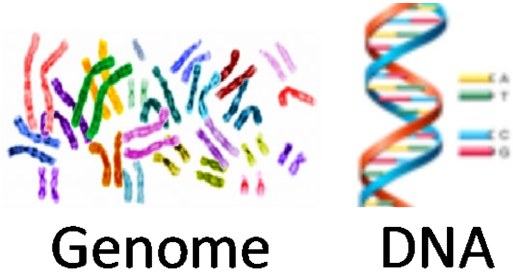
damlasenolcali@gmail.com
https://damlasenolcali.github.io/

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, Onur Mutlu

Carnegie Mellon
ETH zürich
Bilkent University
intel
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
SAFARI

Genome Sequencing

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



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

Genome Sequence Analysis

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

Linear Reference: ACGTACGT

Read: ACG**G**

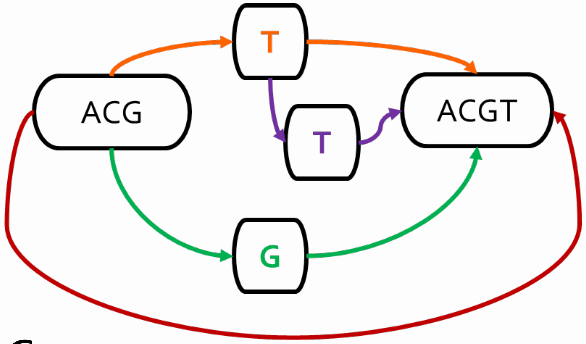
Alternative Sequence: ACG**G**ACGT

Alternative Sequence: ACG**TT**ACGT

Alternative Sequence: ACG–ACGT

Sequence-to-Sequence (S2S) Mapping

Graph-based Reference:



Read: ACGG

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**
  - ○ **Genome Graphs**
  - ○ **Sequence-to-Graph Mapping**
- ❑ SeGraM: Universal Genomic Mapping Accelerator
  - ○ High-Level Overview
  - ○ MinSeed
  - ○ BitAlign
  - ○ Use Cases
- ❑ Evaluation
- ❑ Conclusion

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

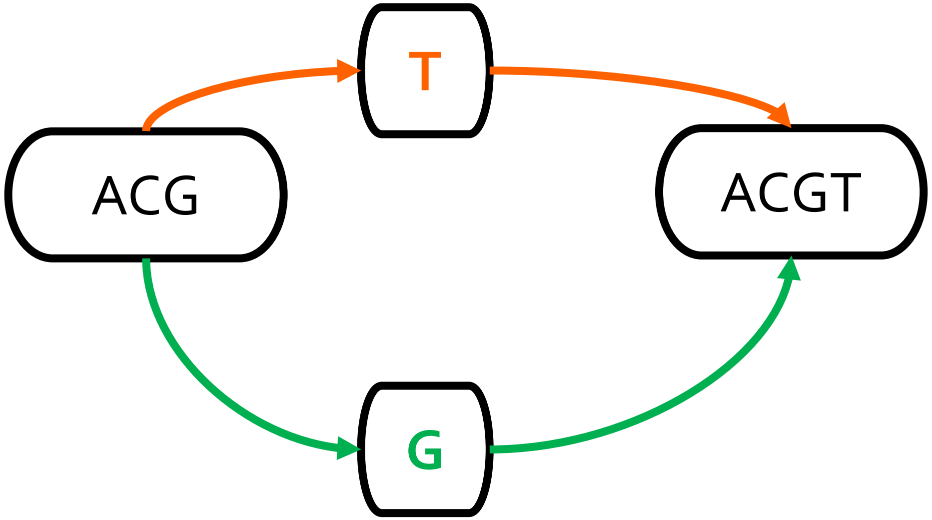
Genome Graphs

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

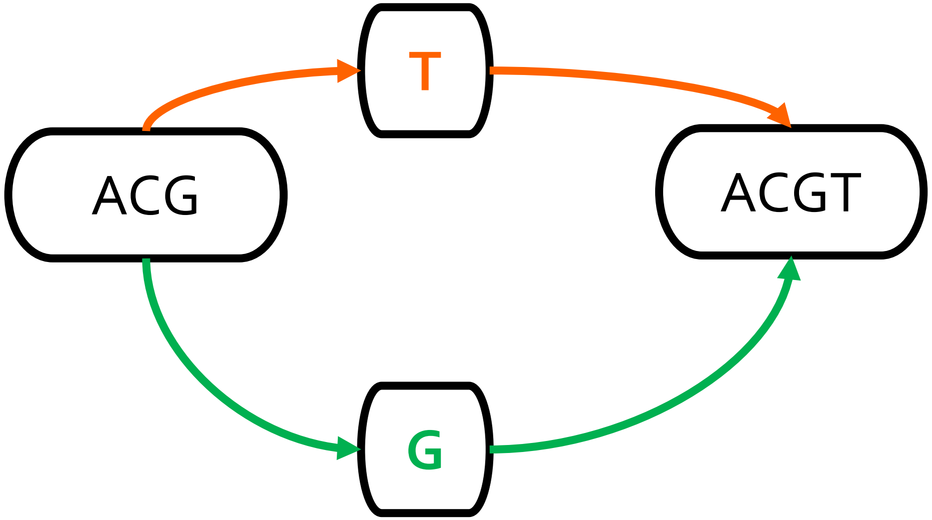
Genome graphs:

- ☐ 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: ACG**T**ACGT

Sequence #2: ACG**G**ACGT

Sequence #3: ACG**TT**ACGT



Genome Graphs

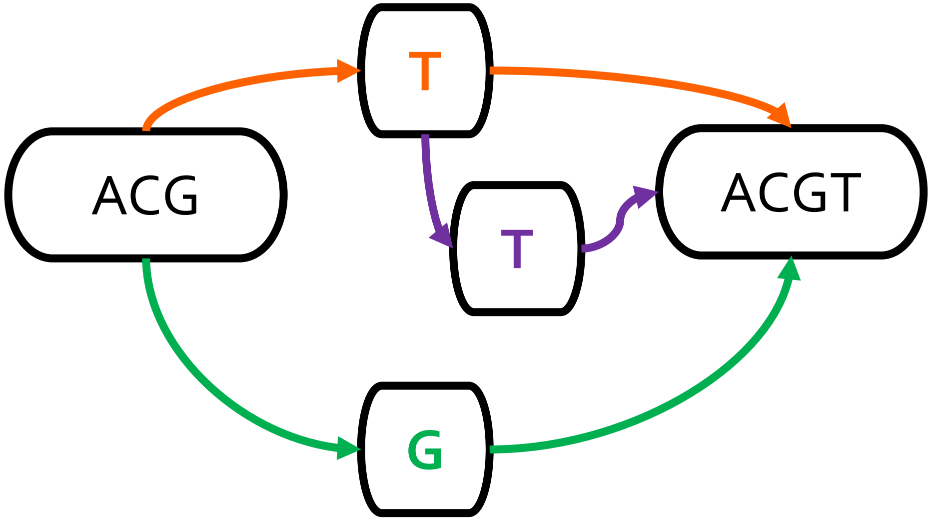
Genome graphs:

- ☐ 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:

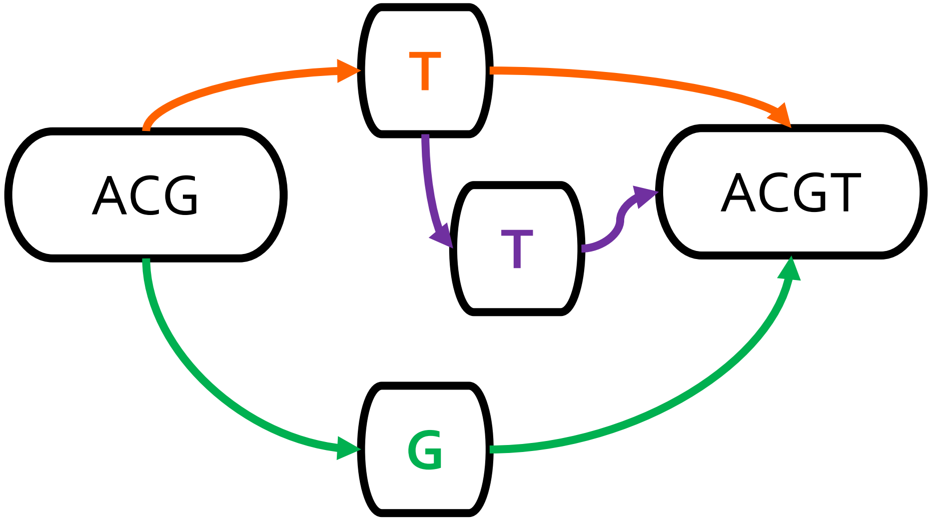
- ☐ Combine the linear reference genome with the known genetic variations in the entire population as a graph-based data structure
- ☐ Enable us to move away from aligning with a single linear reference genome (reference bias) and more accurately express the genetic diversity in a population

Sequence #1: ACGTACGT

Sequence #2: ACGGACGT

Sequence #3: ACGTTACGT

Sequence #4: ACGACGT



Genome Graphs

Genome graphs:

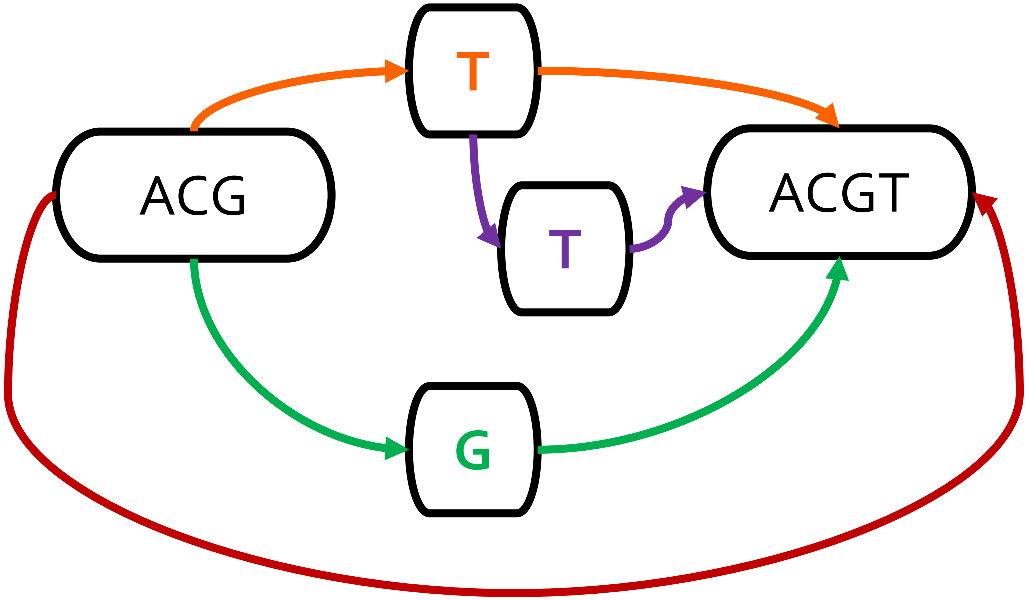
- ☐ Combine the linear reference genome with the known genetic variations in the entire population as a graph-based data structure
- ☐ Enable us to move away from aligning with a single linear reference genome (reference bias) and more accurately express the genetic diversity in a population

Sequence #1: ACGTACGT

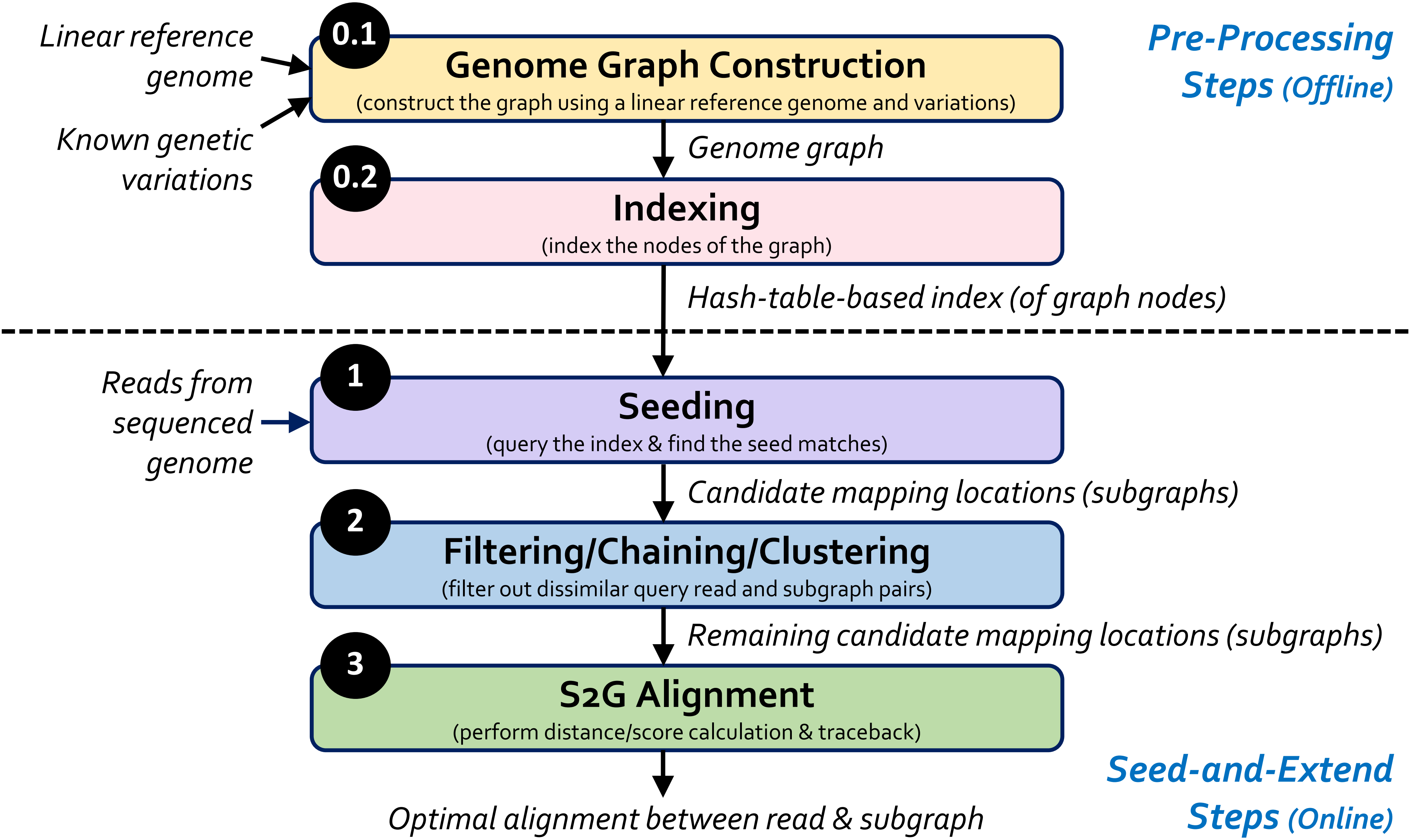
Sequence #2: ACGGACGT

Sequence #3: ACGTTACGT

Sequence #4: ACGACGT



Sequence-to-Graph Mapping Pipeline



Linear reference genome

Known genetic variations

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

Genome graph

0.2 **Indexing**
(index the nodes of the graph)

Hash-table-based index (of graph nodes)

***Pre-Processing Steps** (Offline)*

Reads from sequenced genome

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

Candidate mapping locations (subgraphs)

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

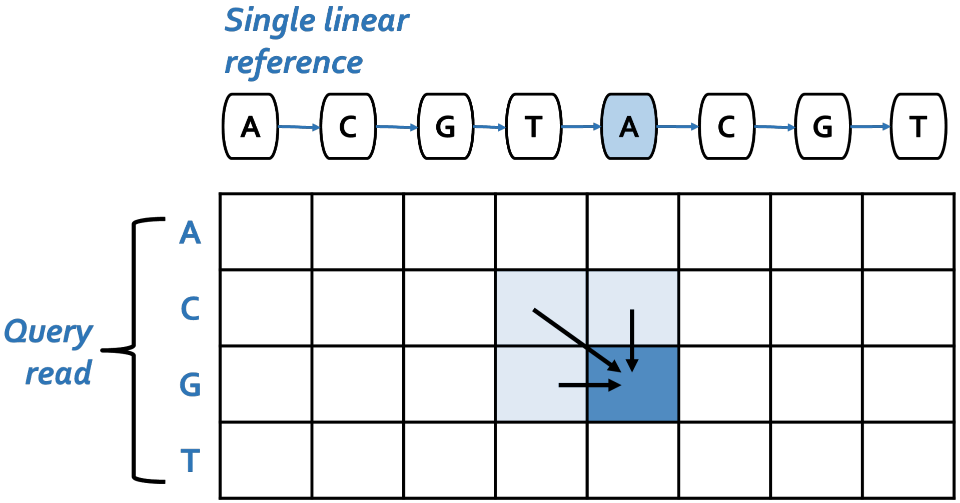
Remaining candidate mapping locations (subgraphs)

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

Optimal alignment between read & subgraph

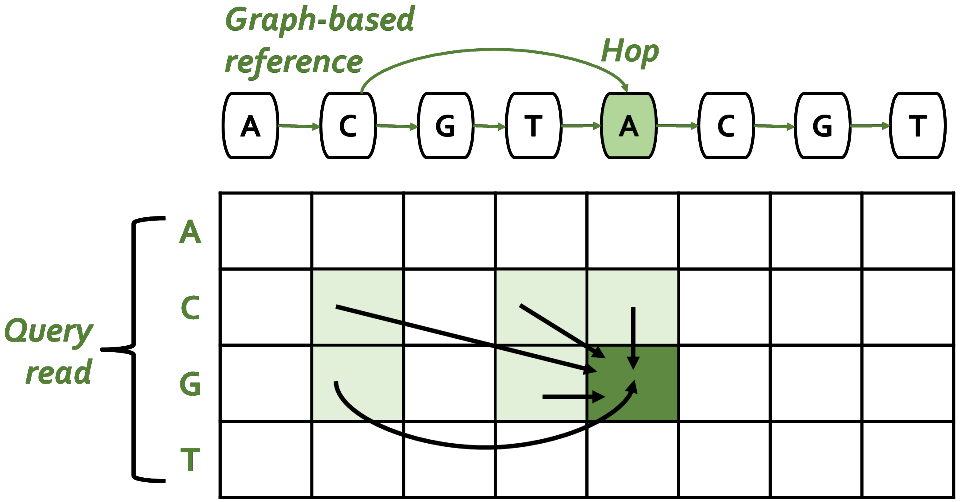
***Seed-and-Extend Steps** (Online)*

S2S vs. S2G Alignment



Sequence-to-Sequence (S2S) Alignment

S2S vs. S2G Alignment



Sequence-to-Graph (S2G) 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

SW

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

HW

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
  - ○ 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

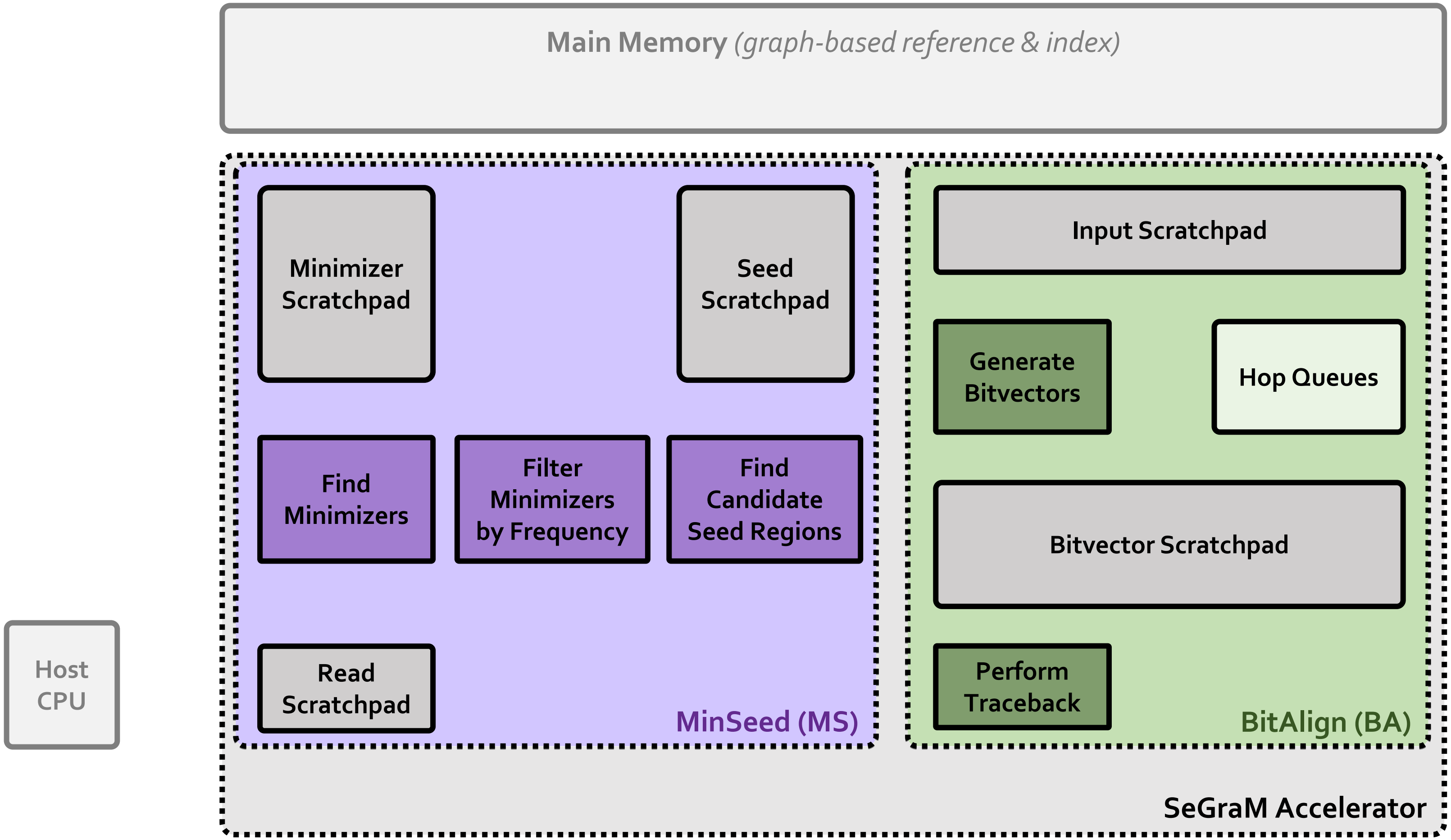
- ❑ We base SeGraM upon a minimizer-based seeding algorithm
- ❑ We propose a novel bitvector-based alignment algorithm to perform approximate string matching between a read and a graph-based reference genome

SW

- ❑ We co-design both algorithms with high-performance, scalable, and efficient hardware accelerators

HW

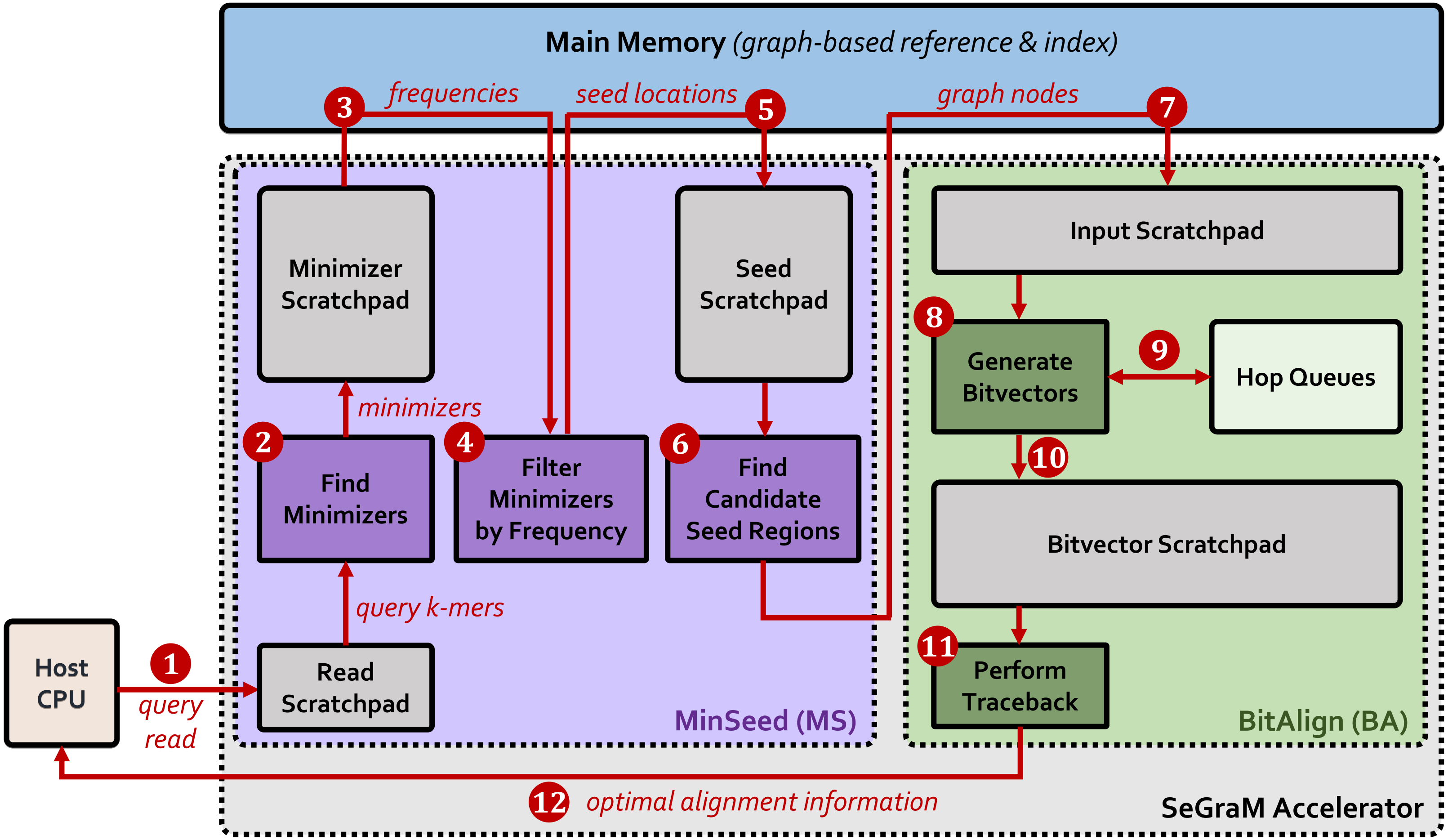
SeGraM Hardware Design



MinSeed: *first* hardware accelerator for **Min**imizer-based **Seed**ing

BitAlign: *first* hardware accelerator for (**Bit**vector-based) sequence-to-graph **Align**ment

SeGraM Hardware Design

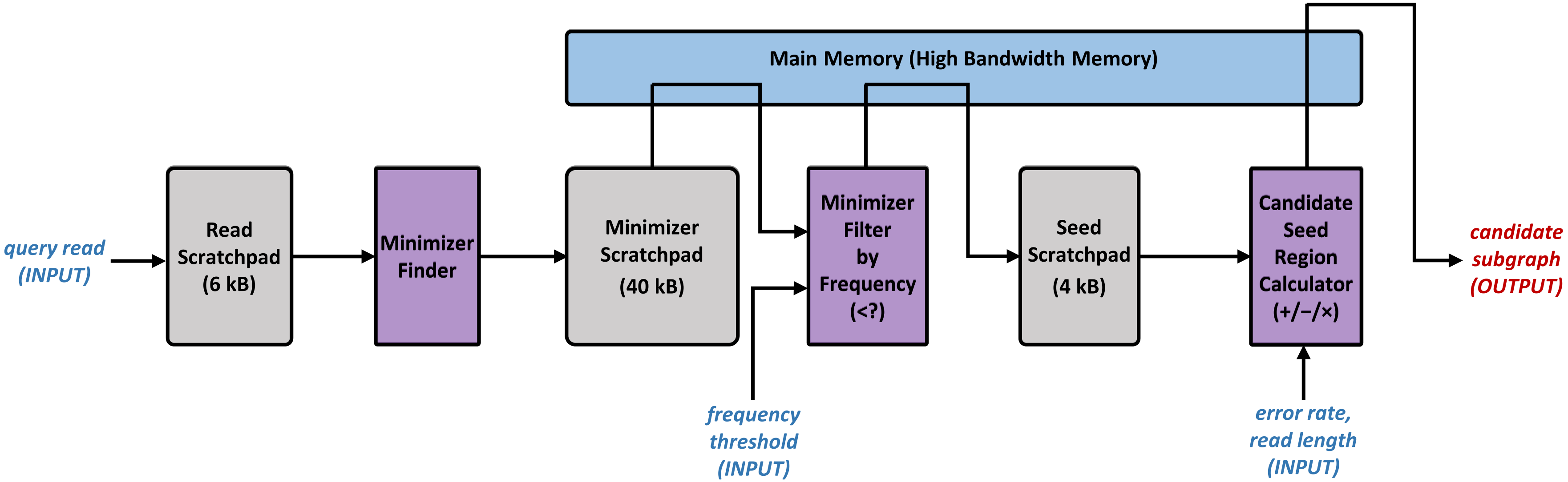


MinSeed: *first* hardware accelerator for **Min**imizer-based **Seed**ing

BitAlign: *first* hardware accelerator for (**Bit**vector-based) sequence-to-graph **Align**ment

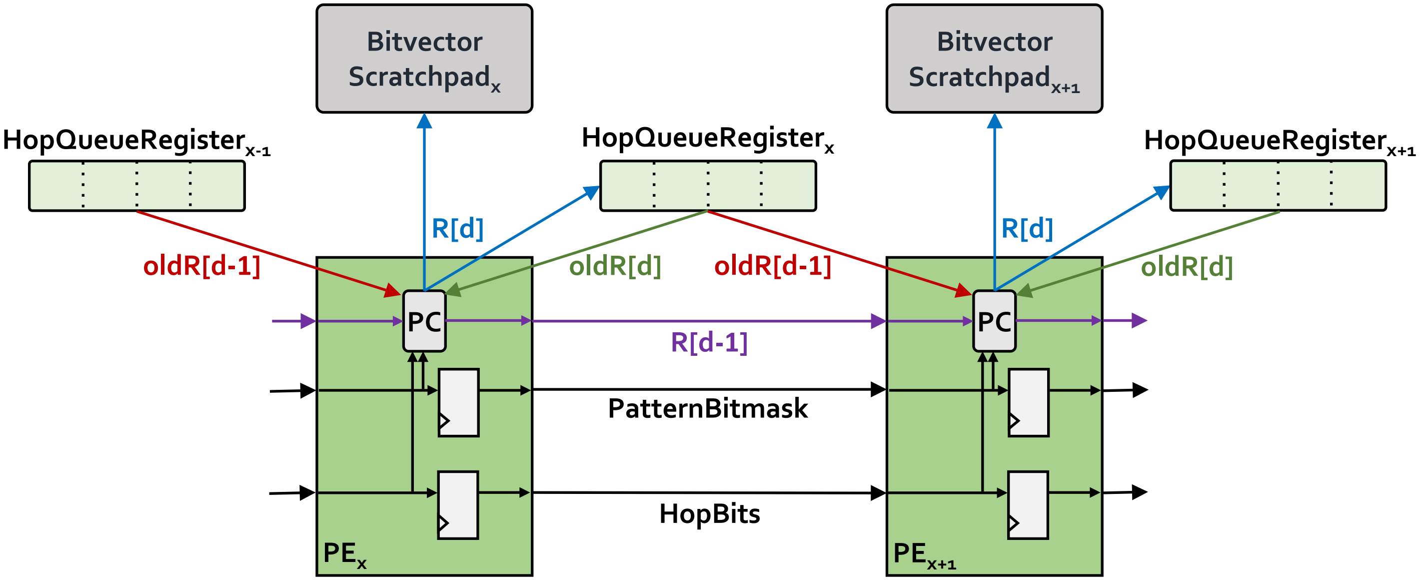
MinSeed HW

- ☐ 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



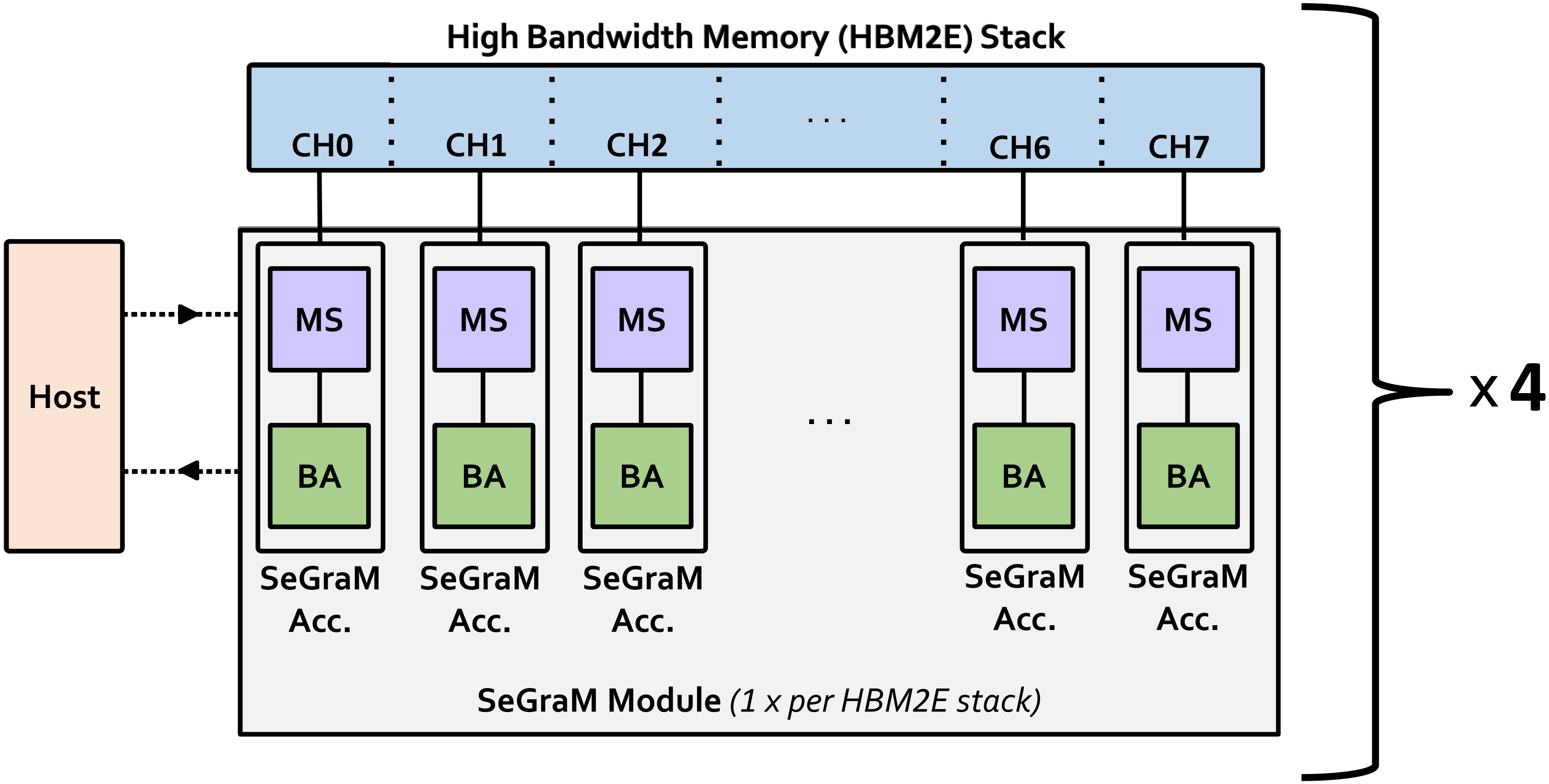
BitAlign HW

- **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*)

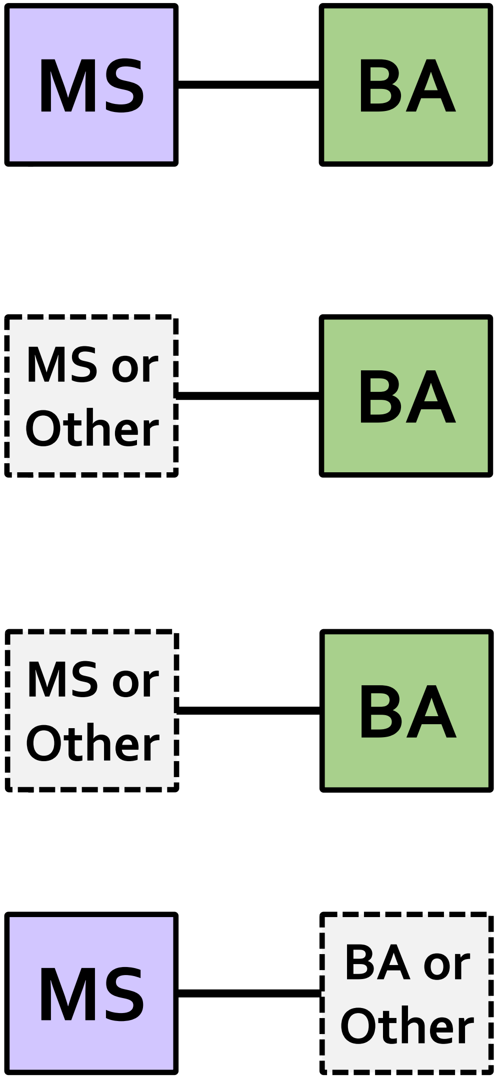


[*] D. Senol Cali *et al.* "GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis" (MICRO'20)

Overall System Design of SeGraM



Use Cases of SeGraM



(1) Sequence-to-Graph Mapping

(2) Sequence-to-Graph Alignment

(3) Sequence-to-Sequence Alignment

(4) Seeding

Outline

- ☐ Introduction
- ☐ Background
  - ○ 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

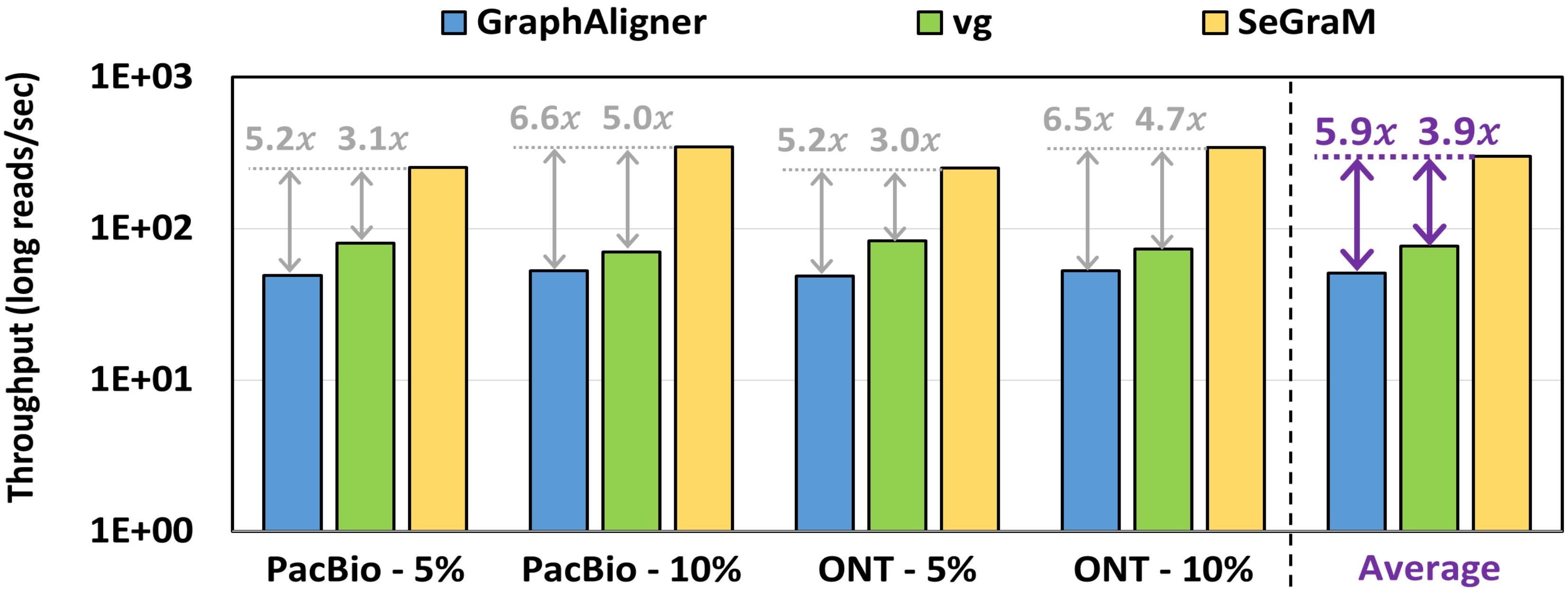
- **Datasets:**
  - Graph-based reference: GRCh38 + 7 VCF files for HG001-007
  - Simulated datasets for both short and long reads

Key Results – Area & Power

- Based on our **synthesis** of **MinSeed** and **BitAlign** accelerator datapaths using the Synopsys Design Compiler with a **28nm** process (**@ 1GHz):**

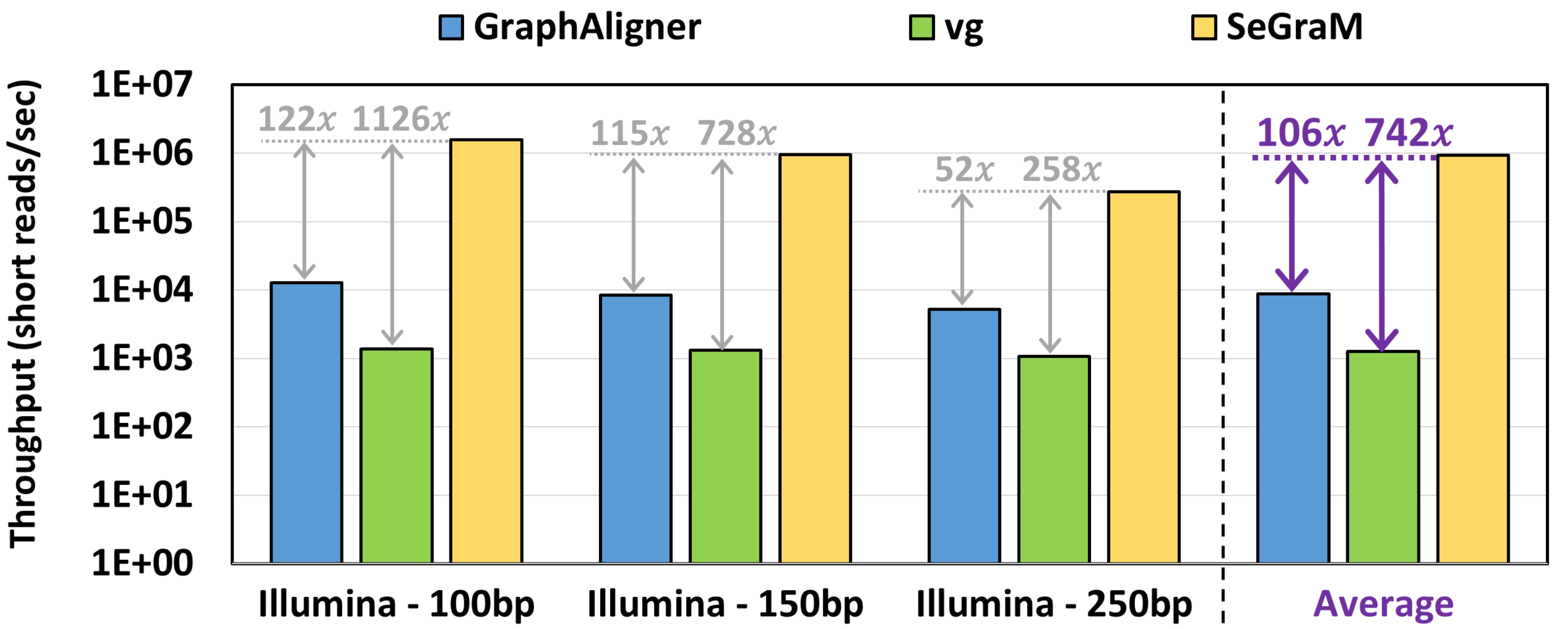
Component	**Area (mm^2)**	**Power (mW)**
MinSeed – Logic	0.017	10.8
Read Scratchpad (6 kB)	0.012	7.9
Minimizer Scratchpad (40 kB)	0.055	22.7
Seed Scratchpad (4 kB)	0.008	6.4
BitAlign – Edit Distance Calculation Logic with Hop Queue Registers (64 PEs)	0.393	378.0
BitAlign – Traceback Logic	0.020	2.7
Input Scratchpad (24 kB)	0.033	13.3
Bitvector Scratchpads (128 kB)	0.329	316.2
Total – 1 SeGraM Accelerator	**0.867**	**758.0 (0.8 W)**
Total – 4 SeGraM Modules (32 SeGraM Accelerators)	**27.744**	**24.3 W**
HBM2E (4 stacks)	**--**	**3.8 W**

Key Results – SeGraM with Long Reads



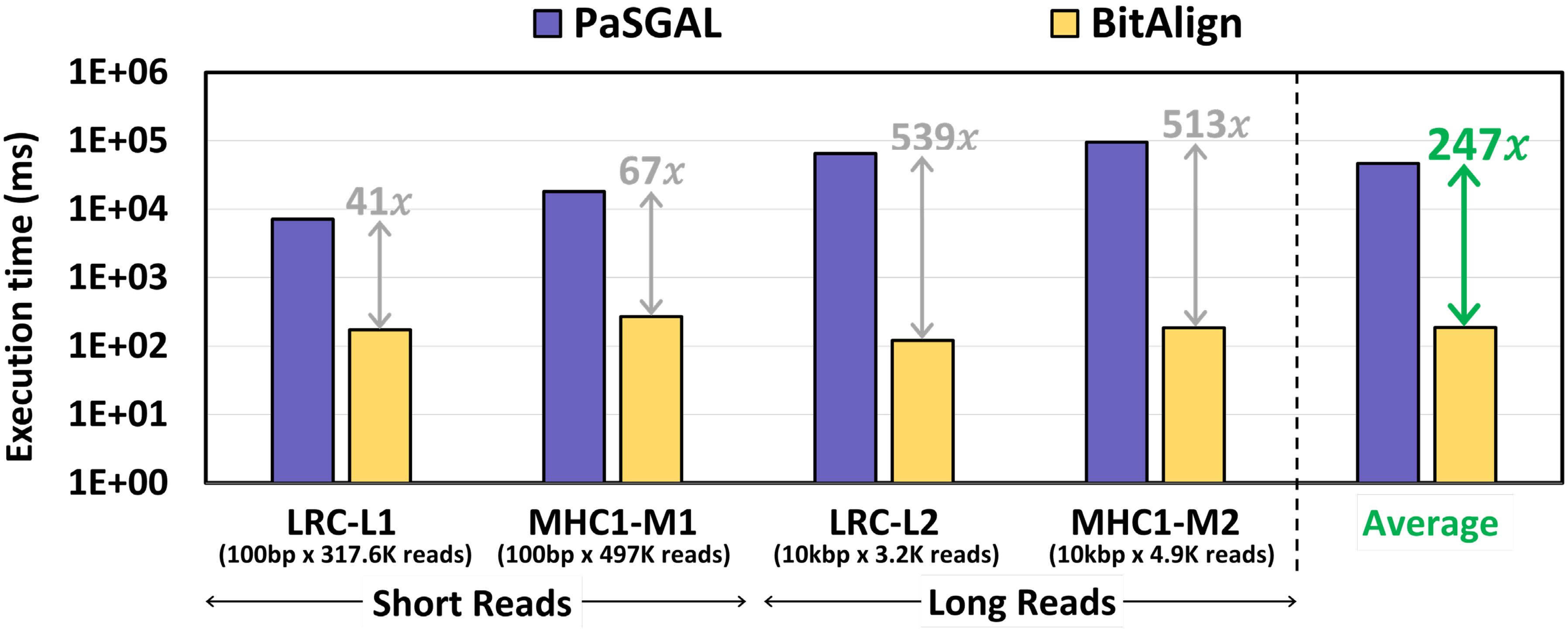
SeGraM provides **5.9× and 3.9× throughput improvement** over GraphAligner and vg, while **reducing the power consumption by 4.1× and 4.4×**

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×-539× 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
  - ○ 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

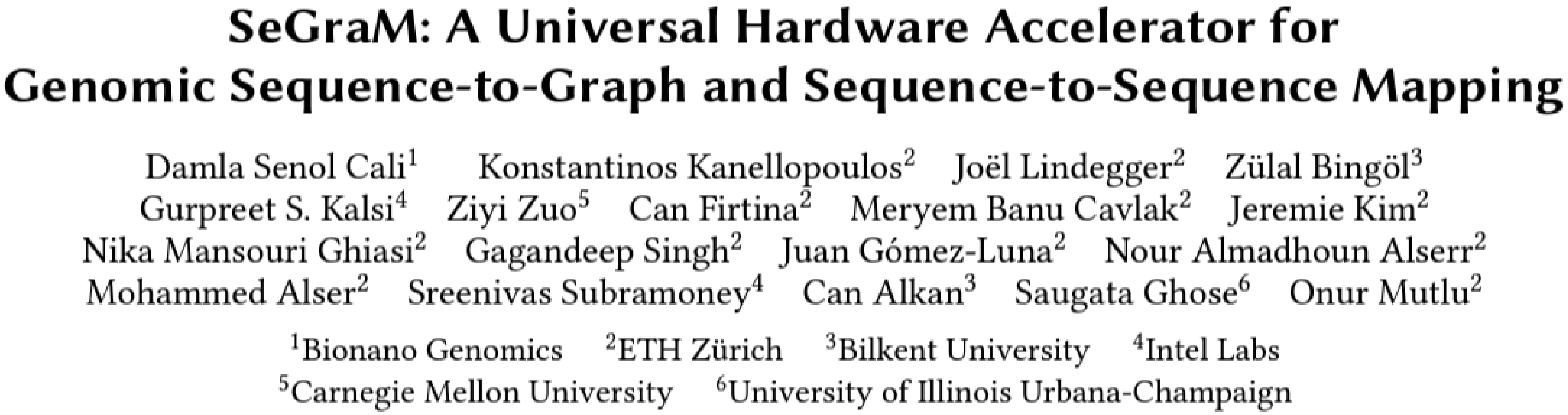
- **SeGraM:** *First universal algorithm/hardware co-designed genomic mapping accelerator that supports:*
  - Sequence-to-graph (S2G) & sequence-to-sequence (S2S) mapping
  - Short & long reads
  - **MinSeed:** *First minimizer-based seeding accelerator*
  - **BitAlign:** *First (bitvector-based) S2G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

Proceedings of the 49th International Symposium on Computer Architecture (ISCA), New York City, NY, June 2022.



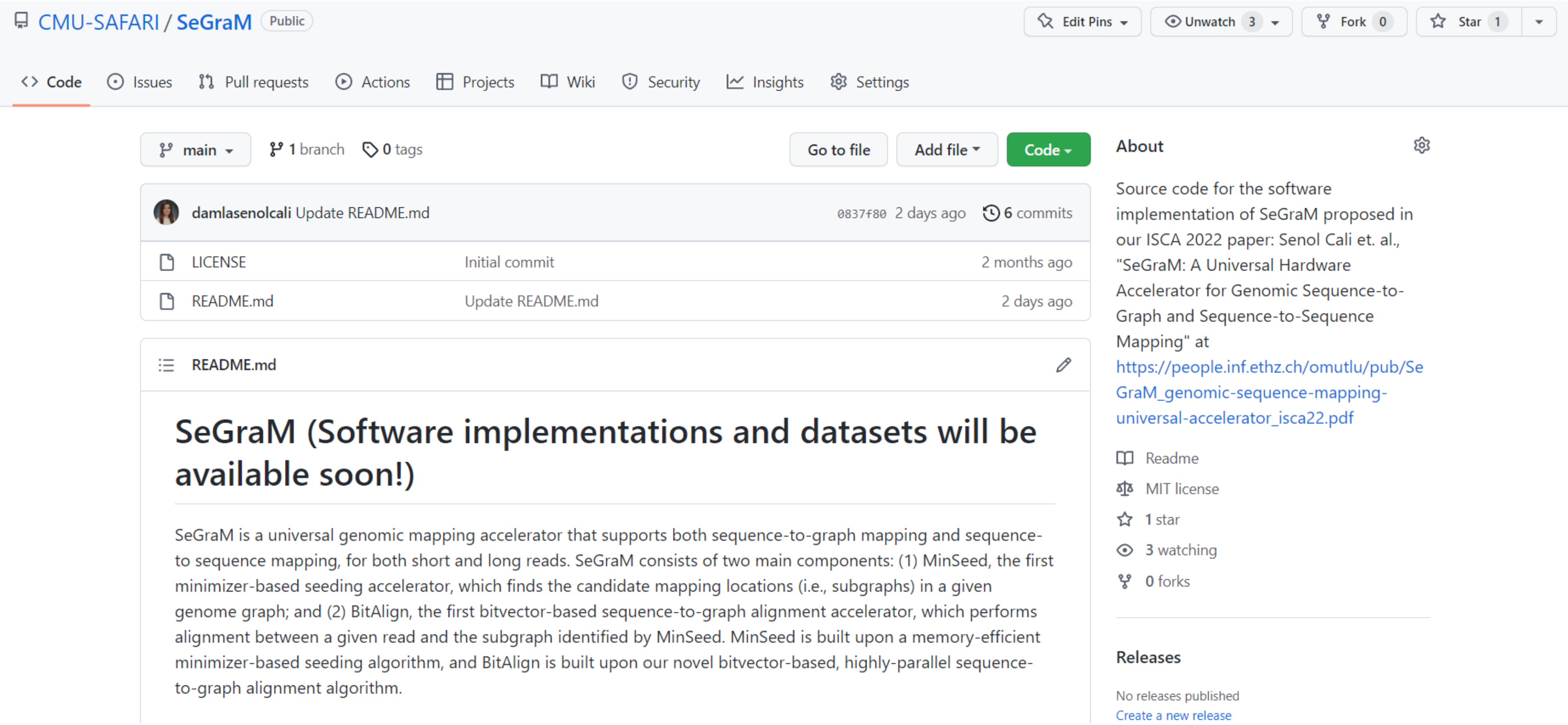
SeGraM: A Universal Hardware Accelerator for Genomic Sequence-to-Graph and Sequence-to-Sequence Mapping

Damla Senol Cali[1] Konstantinos Kanellopoulos[2] Joël Lindegger[2] Zülal Bingöl[3]
Gurpreet S. Kalsi[4] Ziyi Zuo[5] Can Firtina[2] Meryem Banu Cavlak[2] Jeremie Kim[2]
Nika Mansouri Ghiasi[2] Gagandeep Singh[2] Juan Gómez-Luna[2] Nour Almadhoun Alserr[2]
Mohammed Alser[2] Sreenivas Subramoney[4] Can Alkan[3] Saugata Ghose[6] Onur Mutlu[2]

[1]Bionano Genomics [2]ETH Zürich [3]Bilkent University [4]Intel Labs
[5]Carnegie Mellon University [6]University of Illinois Urbana-Champaign

SeGraM – GitHub Page

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



damlasenolcali Update README.md 0837f80 2 days ago 6 commits

LICENSE	Initial commit	2 months ago
README.md	Update README.md	2 days ago

README.md

SeGraM (Software implementations and datasets will be available soon!)

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 genome graph; and (2) BitAlign, the first bitvector-based sequence-to-graph alignment accelerator, which performs alignment between a given read and the subgraph identified by MinSeed. MinSeed is built upon a memory-efficient minimizer-based seeding algorithm, and BitAlign is built upon our novel bitvector-based, highly-parallel sequence-to-graph alignment algorithm.

About

Source code for the software implementation of SeGraM proposed in our ISCA 2022 paper: Senol Cali et. al., "SeGraM: A Universal Hardware Accelerator for Genomic Sequence-to-Graph and Sequence-to-Sequence Mapping" at https://people.inf.ethz.ch/omutlu/pub/SeGraM_genomic-sequence-mapping-universal-accelerator_isca22.pdf

Releases

No releases published
Create a new release

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

Damla Senol Cali, Ph.D.

damlasenolcali@gmail.com
https://damlasenolcali.github.io/

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, Onur Mutlu

Carnegie Mellon
ETH zürich
Bilkent University
intel
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
SAFARI

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**

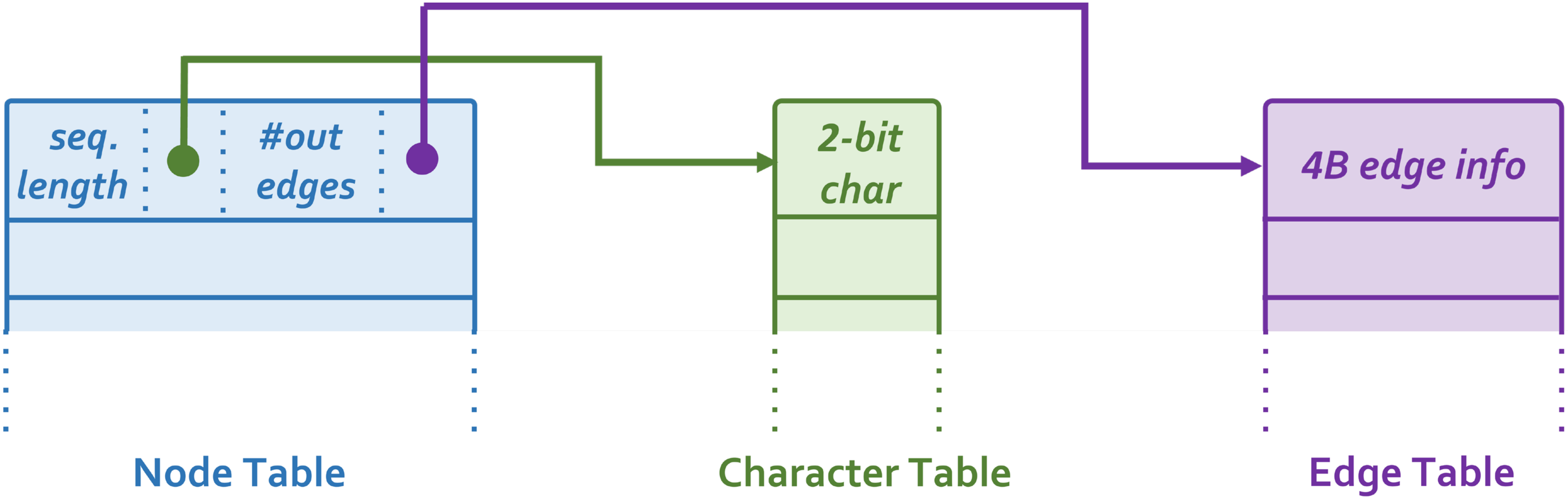
- ❑ **Well studied** with many available tools and accelerators

Sequence-to-Graph (S2G) Mapping

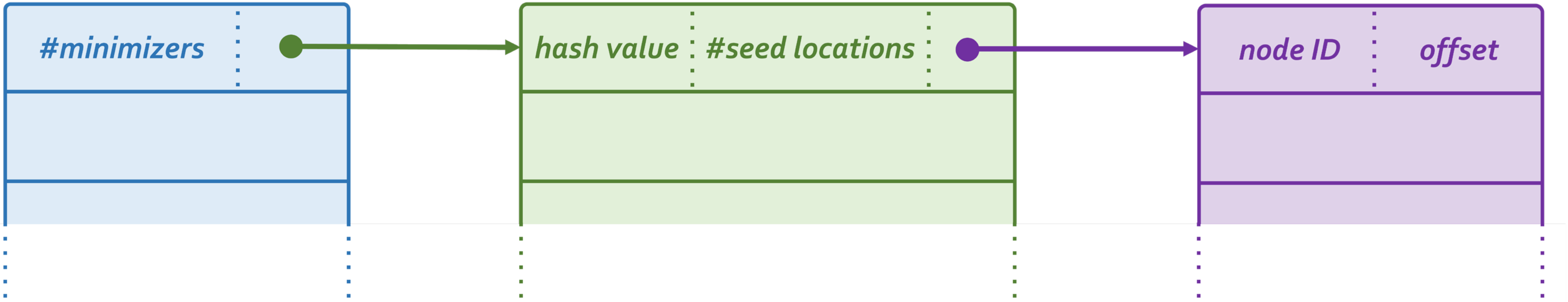
- ❑ Replaces the linear reference sequence with a ***graph-based representation of the reference genome (genome graph)***
- ❑ **Captures the genetic variations and diversity** across many individuals in a population
- ❑ Results in **notable quality improvements** in GSA

- ❑ **More difficult** computational problem
- ❑ **No prior hardware design** for graph-based GSA

SeGraM – Graph Structure



SeGraM – Index Structure

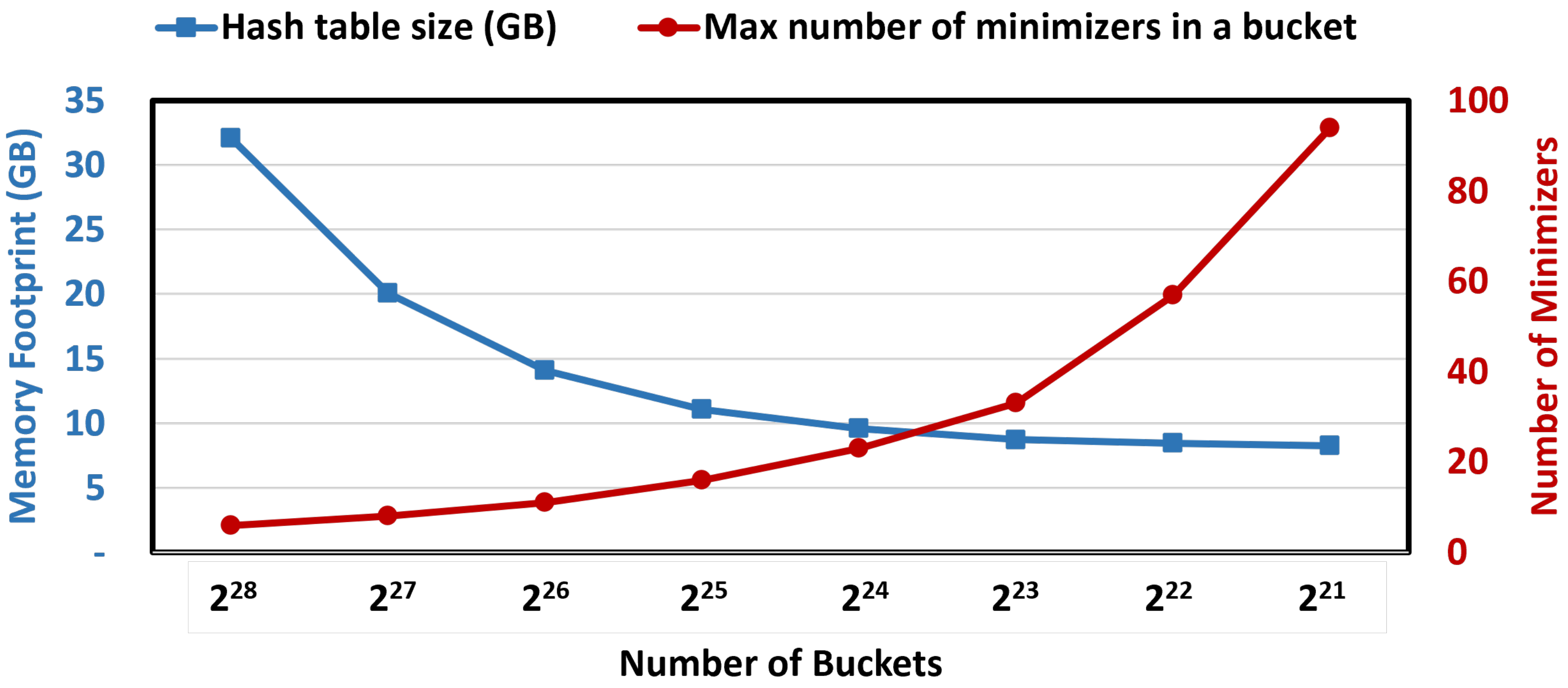


First Level: Buckets

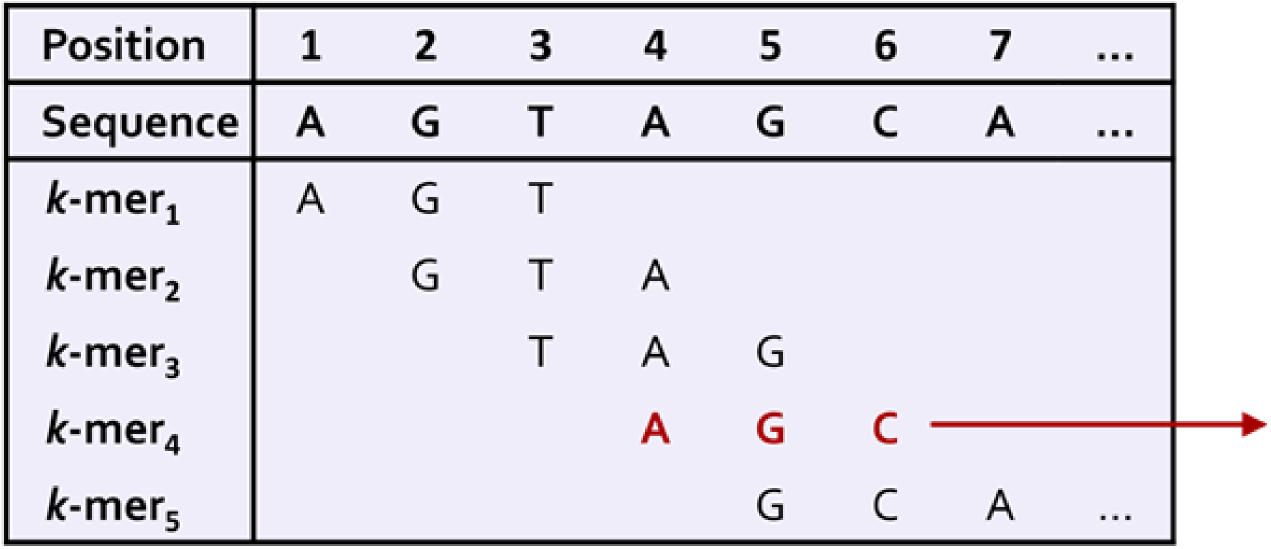
Second Level: Minimizers

Third Level: Seed Locations

SeGraM – Selection of #Buckets



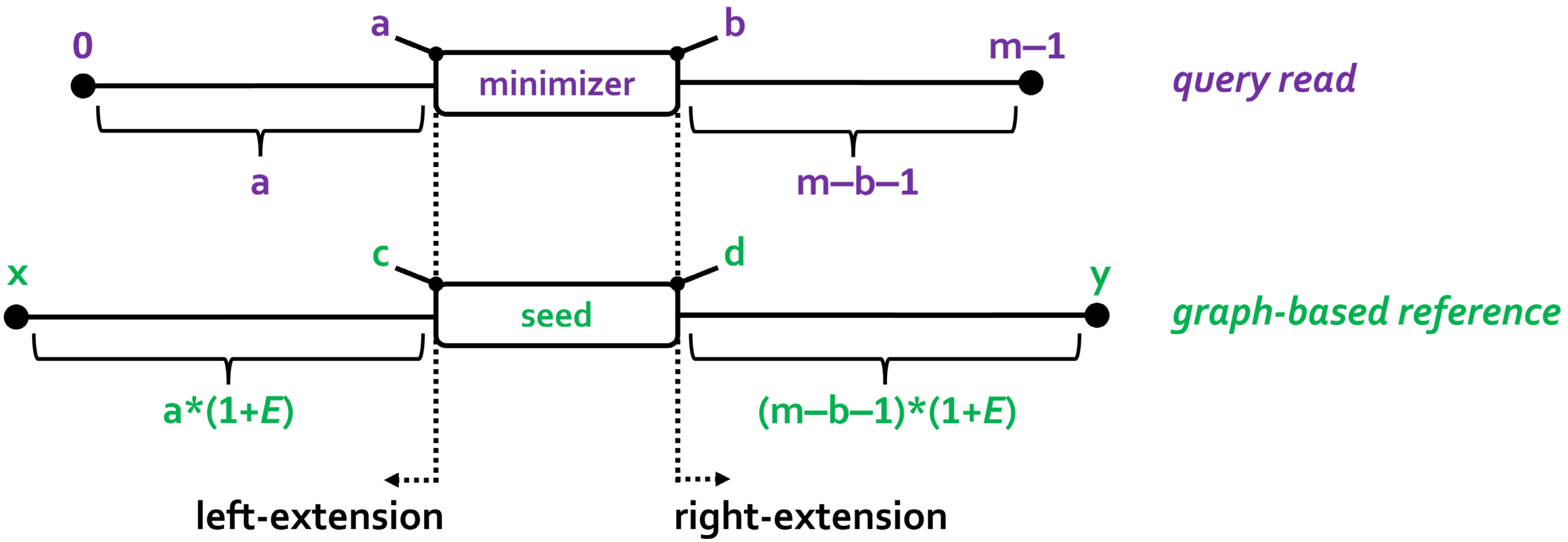
Minimizers



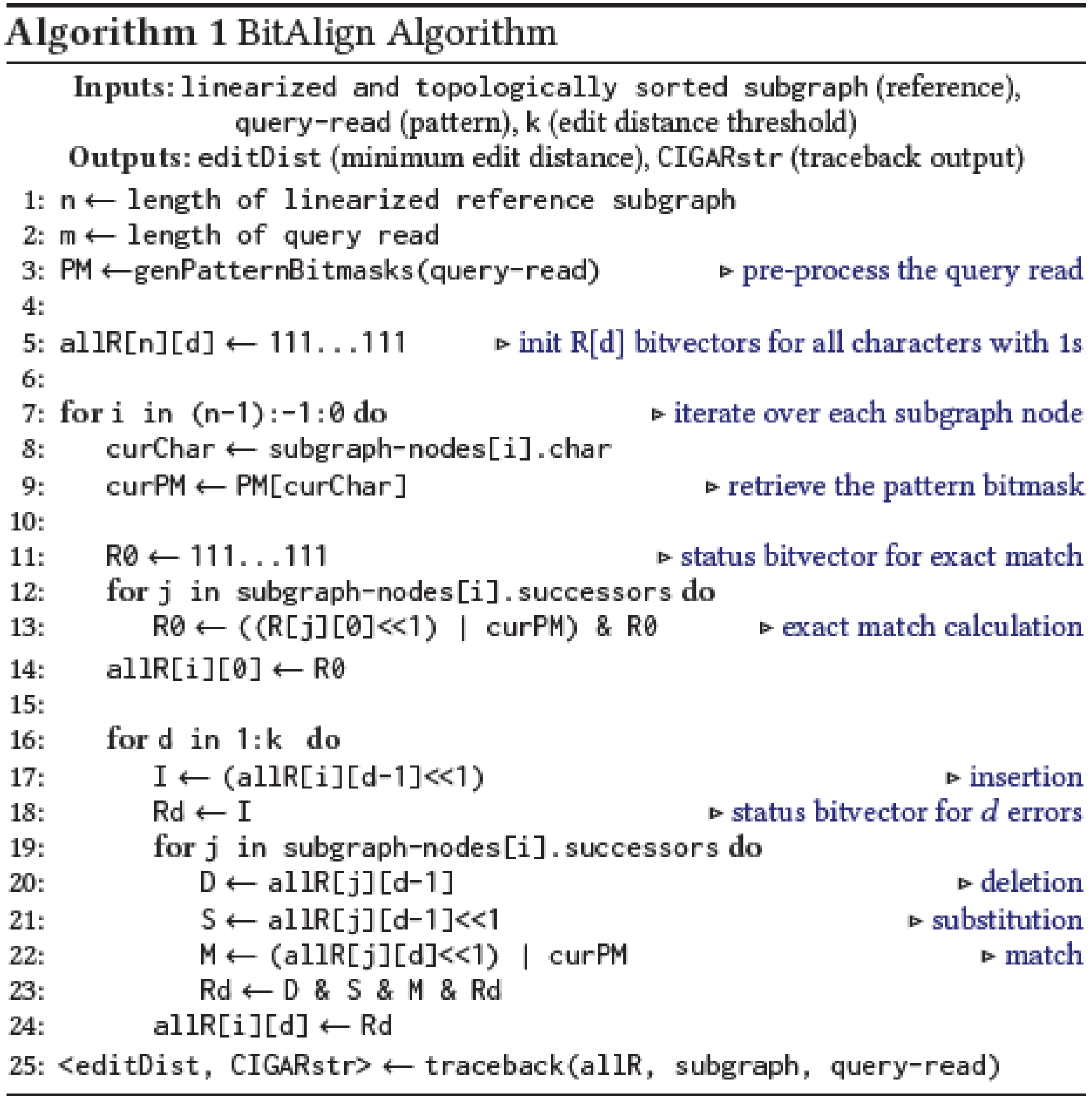
Position	1	2	3	4	5	6	7	...
Sequence	A	G	T	A	G	C	A	...
k-mer$_1$	A	G	T					
k-mer$_2$		G	T	A				
k-mer$_3$			T	A	G			
k-mer$_4$				A	G	C		
k-mer$_5$					G	C	A	...

lexicographically smallest k-mer

MinSeed – Region Calculation



BitAlign Algorithm



Algorithm 1 BitAlign Algorithm

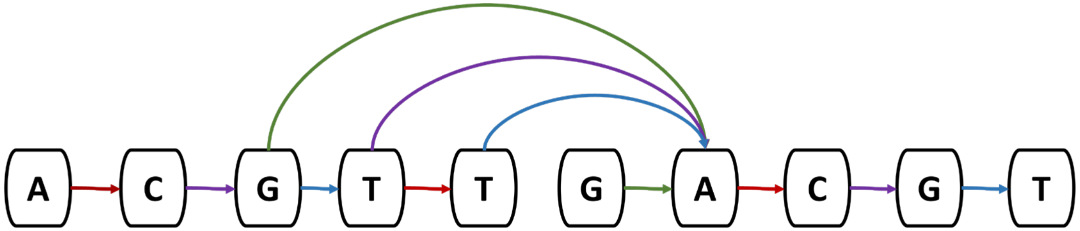
Inputs: `linearized and topologically sorted subgraph` (reference), `query-read` (pattern), `k` (edit distance threshold)

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

```
n ← length of linearized reference subgraph
m ← length of query read
PM ←genPatternBitmasks(query-read)            ▷ pre-process the query read

allR[n][d] ← 111...111        ▷ init R[d] bitvectors for all characters with 1s

for i in (n-1):-1:0 do                        ▷ iterate over each subgraph node
    curChar ← subgraph-nodes[i].char
    curPM ← PM[curChar]                       ▷ retrieve the pattern bitmask

    R0 ← 111...111                            ▷ status bitvector for exact match
    for j in subgraph-nodes[i].successors do
        R0 ← ((R[j][0]<<1) | curPM) & R0       ▷ exact match calculation
    allR[i][0] ← R0

    for d in 1:k  do
        I ← (allR[i][d-1]<<1)                              ▷ insertion
        Rd ← I                                  ▷ status bitvector for d errors
        for j in subgraph-nodes[i].successors do
            D ← allR[j][d-1]                                ▷ deletion
            S ← allR[j][d-1]<<1                             ▷ substitution
            M ← (allR[j][d]<<1) | curPM                     ▷ match
            Rd ← D & S & M & Rd
        allR[i][d] ← Rd
<editDist, CIGARstr> ← traceback(allR, subgraph, query-read)
```

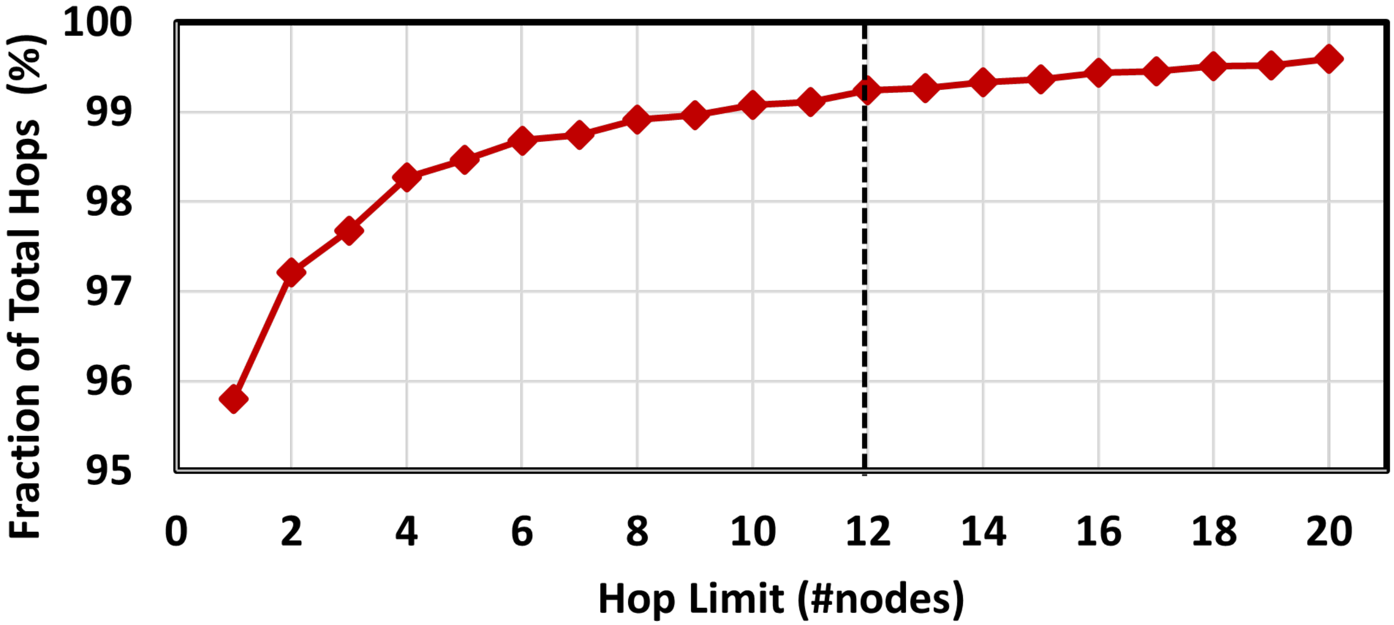
BitAlign – HopBits

Linearized Graph



NodeID	1	2	3	4	5	6	7	8	9	10
1	•	•	•	•	•	•	•	•	•	•
2	1	•	•	•	•	•	•	•	•	•
3	•	1	•	•	•	•	•	•	•	•
4	•	•	1	•	•	•	•	•	•	•
5	•	•	•	1	•	•	•	•	•	•
6	•	•	•	•	•	•	•	•	•	•
7	•	•	1	1	1	1	•	•	•	•
8	•	•	•	•	•	•	1	•	•	•
9	•	•	•	•	•	•	•	1	•	•
10	•	•	•	•	•	•	•	•	1	•

BitAlign – Hop Length Selection



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: A A A A T G T T T A G T G C T A C   T G
     Read: A A A   T G T T T A C T G C T A C T T G
```

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

[1] R. A. Baeza-Yates and G. H. Gonnet. "A New Approach to Text Searching." *CACM,* 1992.
[2] S. Wu and U. Manber. "Fast Text Searching: Allowing Errors." *CACM,* 1992.

Bitap Algorithm (cont'd.)

- **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.
```

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] = (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.
```

Data dependency between iterations (i.e., no parallelization)

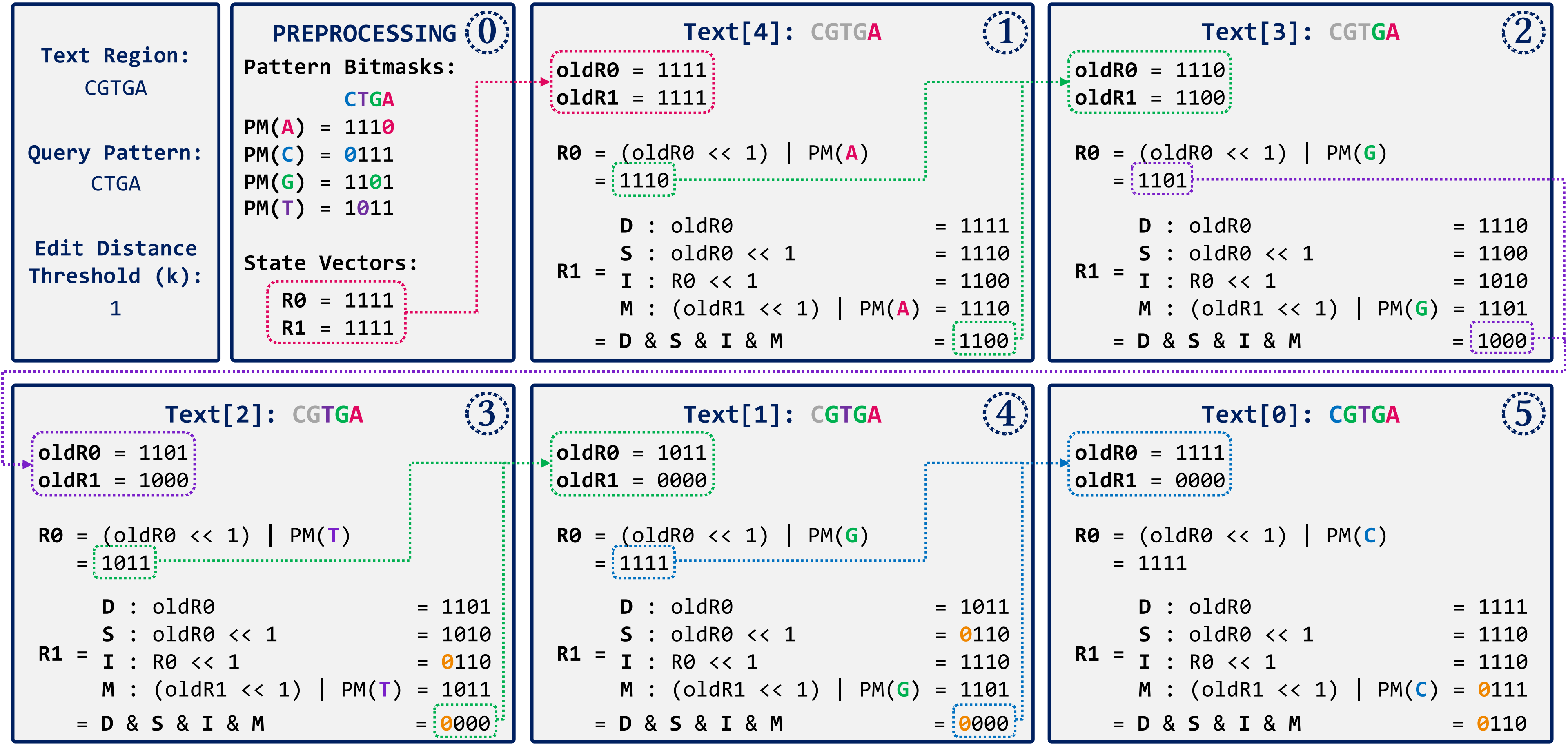
Bitap Algorithm (cont'd.)

- **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



Text Region:
CGTGA

Query Pattern:
CTGA

Edit Distance Threshold (k):
1

PREPROCESSING ⓪

```
Pattern Bitmasks:
          CTGA
PM(A) = 1110
PM(C) = 0111
PM(G) = 1101
PM(T) = 1011

State Vectors:
  R0 = 1111
  R1 = 1111
```

Text[4]: CGTGA ①

```
oldR0 = 1111
oldR1 = 1111

R0 = (oldR0 << 1) | PM(A)
   = 1110

       D : oldR0                    = 1111
       S : oldR0 << 1               = 1110
R1 =   I : R0 << 1                  = 1100
       M : (oldR1 << 1) | PM(A)     = 1110
   = D & S & I & M                  = 1100
```

Text[3]: CGTGA ②

```
oldR0 = 1110
oldR1 = 1100

R0 = (oldR0 << 1) | PM(G)
   = 1101

       D : oldR0                    = 1110
       S : oldR0 << 1               = 1100
R1 =   I : R0 << 1                  = 1010
       M : (oldR1 << 1) | PM(G)     = 1101
   = D & S & I & M                  = 1000
```

Text[2]: CGTGA ③

```
oldR0 = 1101
oldR1 = 1000

R0 = (oldR0 << 1) | PM(T)
   = 1011

       D : oldR0                    = 1101
       S : oldR0 << 1               = 1010
R1 =   I : R0 << 1                  = 0110
       M : (oldR1 << 1) | PM(T)     = 1011
   = D & S & I & M                  = 0000
```

Alignment Found @ Location=2

Text[1]: CGTGA ④

```
oldR0 = 1011
oldR1 = 0000

R0 = (oldR0 << 1) | PM(G)
   = 1111

       D : oldR0                    = 1011
       S : oldR0 << 1               = 0110
R1 =   I : R0 << 1                  = 1110
       M : (oldR1 << 1) | PM(G)     = 1101
   = D & S & I & M                  = 0000
```

Alignment Found @ Location=1

Text[0]: CGTGA ⑤

```
oldR0 = 1111
oldR1 = 0000

R0 = (oldR0 << 1) | PM(C)
   = 1111

       D : oldR0                    = 1111
       S : oldR0 << 1               = 1110
R1 =   I : R0 << 1                  = 1110
       M : (oldR1 << 1) | PM(C)     = 0111
   = D & S & I & M                  = 0110
```

Alignment Found @ Location=0

Limitations of Bitap

Algorithm

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

Hardware

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

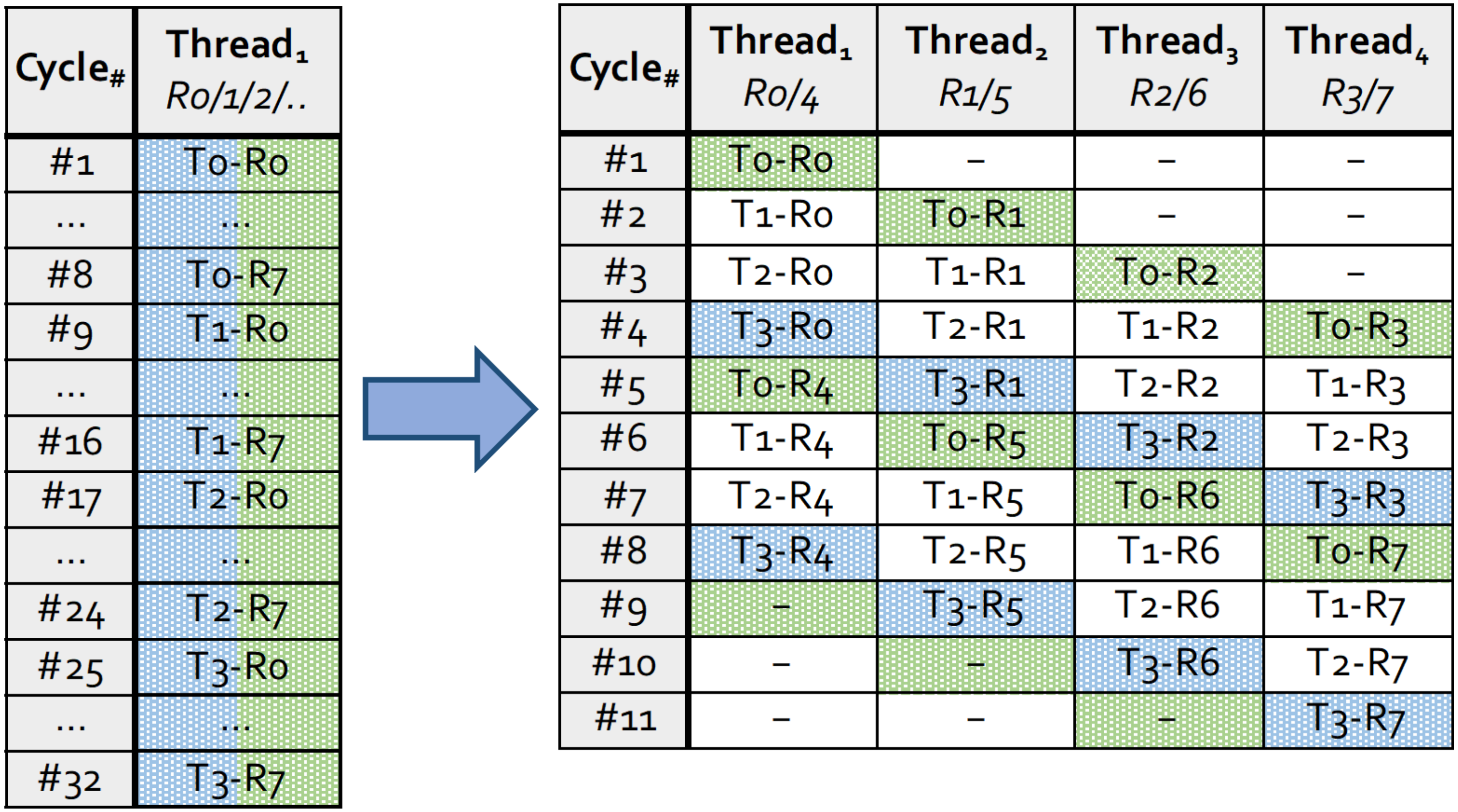
- ❑ **GenASM:** *First* ASM acceleration framework for GSA
  - ○ Approximate string matching (ASM) acceleration framework based on the Bitap algorithm
- ❑ We overcome the five limitations that hinder Bitap's use in GSA:
  - ○ Modified and extended ASM algorithm **SW**
    - ▪ Highly-parallel Bitap with 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 **HW**

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



Cycle$_{\#}$	Thread$_1$ *R0/1/2/..*
#1	T0-R0
…	…
#8	T0-R7
#9	T1-R0
…	…
#16	T1-R7
#17	T2-R0
…	…
#24	T2-R7
#25	T3-R0
…	…
#32	T3-R7

Cycle$_{\#}$	Thread$_1$ *R0/4*	Thread$_2$ *R1/5*	Thread$_3$ *R2/6*	Thread$_4$ *R3/7*
#1	🟩 T0-R0	–	–	–
#2	T1-R0	🟩 T0-R1	–	–
#3	T2-R0	T1-R1	🟩 T0-R2	–
#4	🟦 T3-R0	T2-R1	T1-R2	🟩 T0-R3
#5	🟩 T0-R4	🟦 T3-R1	T2-R2	T1-R3
#6	T1-R4	🟩 T0-R5	🟦 T3-R2	T2-R3
#7	T2-R4	T1-R5	🟩 T0-R6	🟦 T3-R3
#8	🟦 T3-R4	T2-R5	T1-R6	🟩 T0-R7
#9	🟩 –	🟦 T3-R5	T2-R6	T1-R7
#10	–	🟩 –	🟦 T3-R6	T2-R7
#11	–	–	🟩 –	🟦 T3-R7

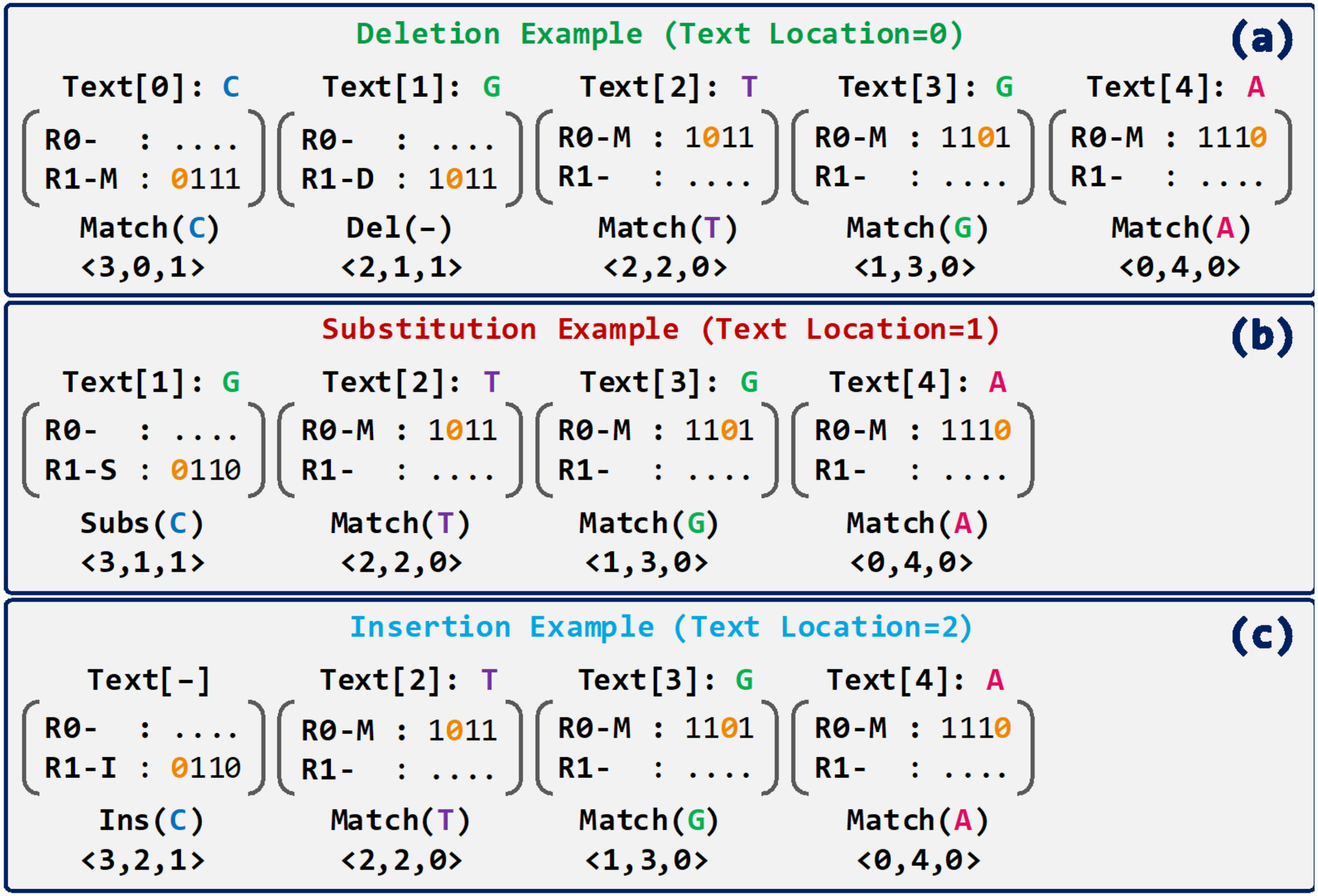
🟦 data *written to memory*

🟩 data *read from memory*

target cell (R_d)

cells target cell depends on ($oldR_d$, R_{d-1}, $oldR_{d-1}$)

Traceback Example with GenASM-TB



Deletion Example (Text Location=0) **(a)**

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>

Substitution Example (Text Location=1) **(b)**

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>

Insertion Example (Text Location=2) **(c)**

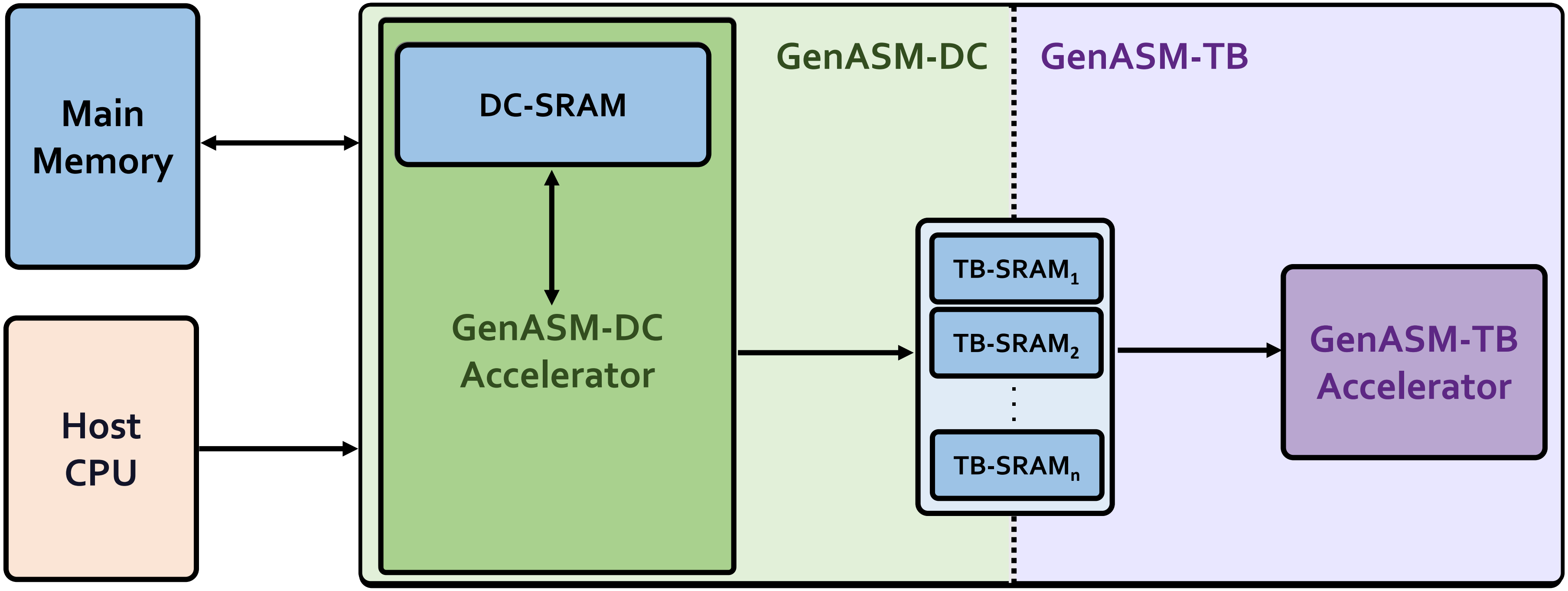
Text[-]
R0- : ....
R1-I : 0110
Ins(C)
<3,2,1>

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>

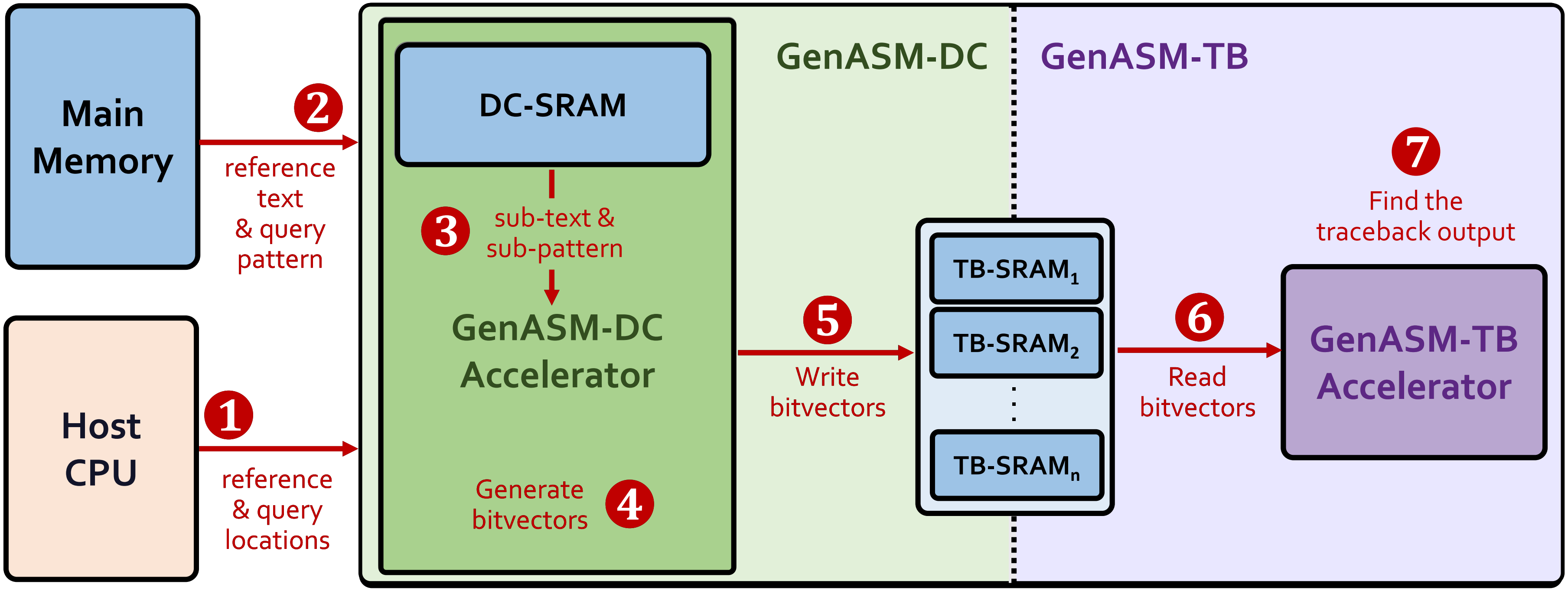
GenASM Hardware Design



GenASM-DC:
generates bitvectors
and performs edit
Distance **C**alculation

GenASM-TB:
performs **T**race**B**ack
and assembles the
optimal alignment

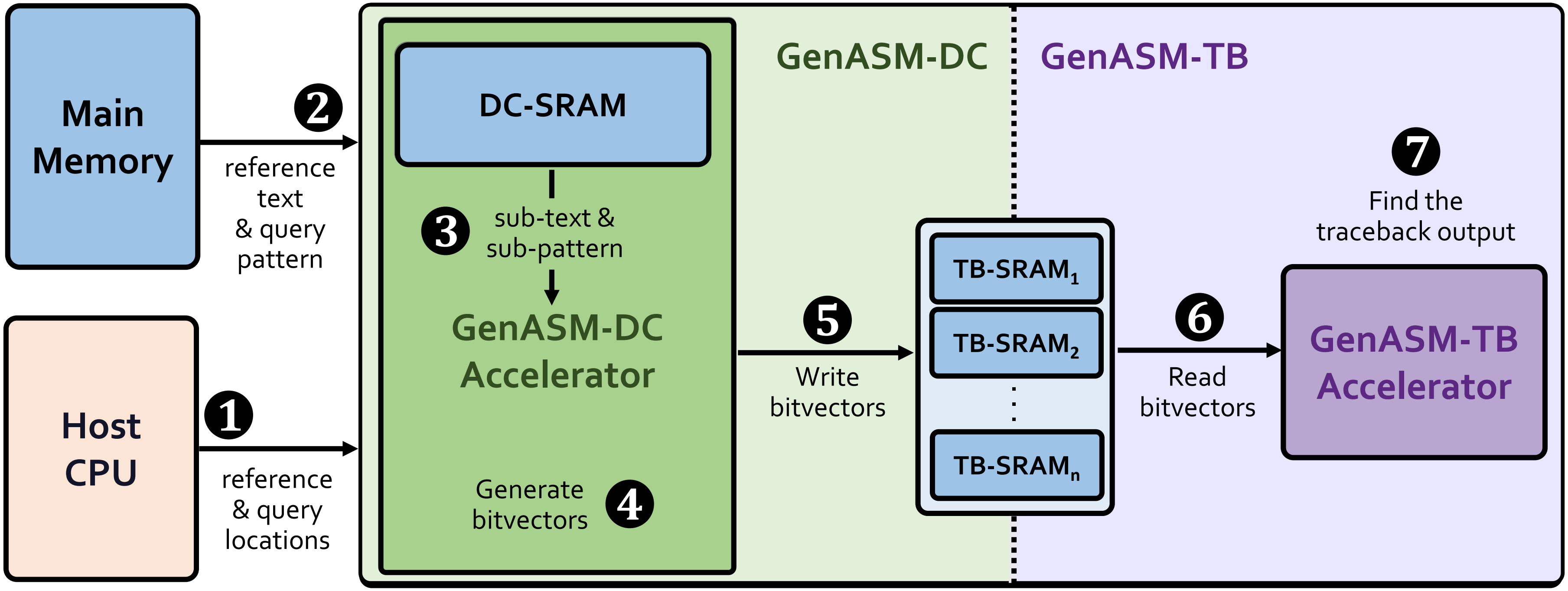
GenASM Hardware Design



GenASM-DC:
generates bitvectors and performs edit **D**istance **C**alculation

GenASM-TB:
performs **T**race**B**ack and assembles the optimal alignment

GenASM Hardware Design



Our *specialized compute units* and *on-chip SRAMs* help us to:

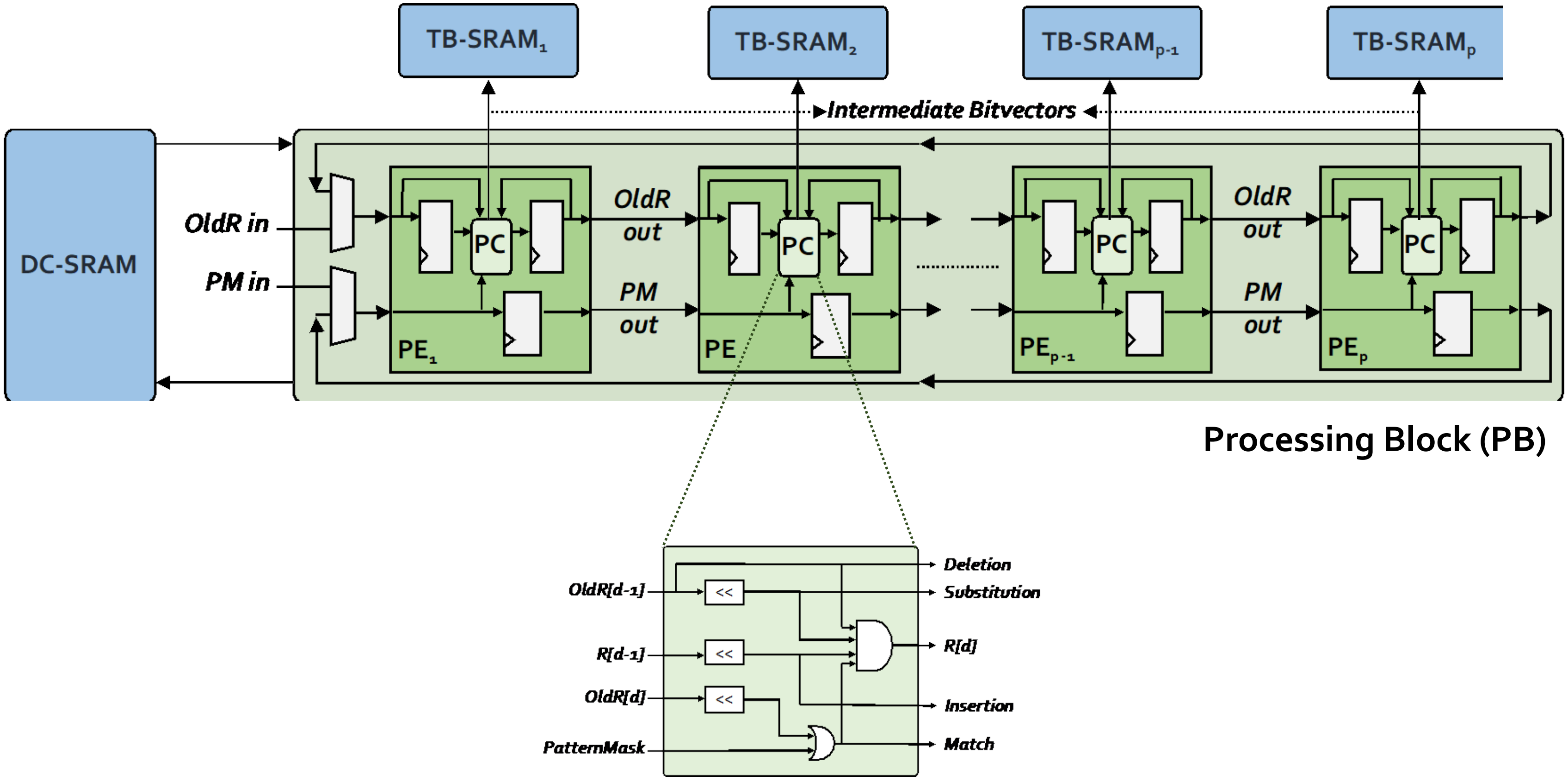
→ Match the rate of computation with memory capacity and bandwidth

→ 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

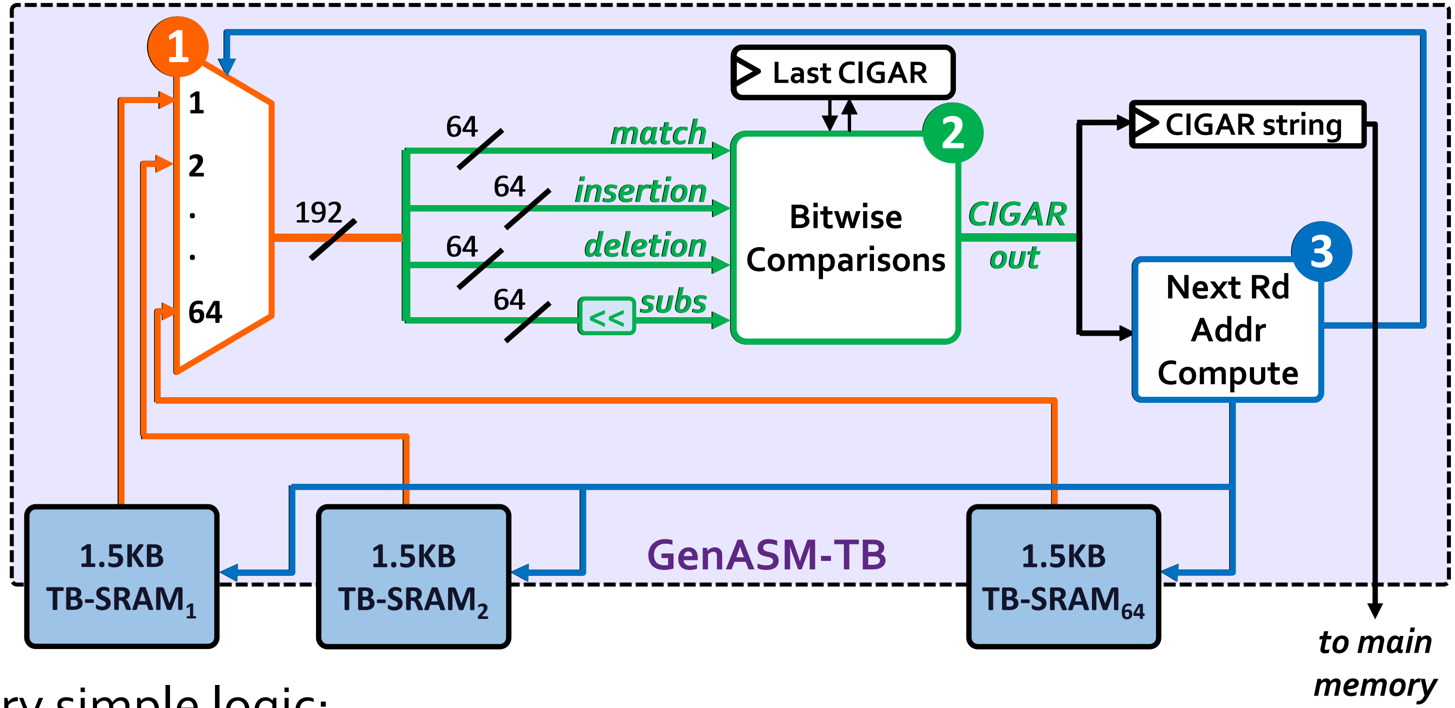
- **Linear cyclic systolic array** based accelerator
  - Designed to maximize parallelism and minimize memory bandwidth and memory footprint



Processing Block (PB)

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.)

	SW Baselines	**HW Baselines**
Read Alignment	Minimap2[1] BWA-MEM[2]	GACT (Darwin)[3] SillaX (GenAx)[4]
Pre-Alignment Filtering	–	Shouji[5]
Edit Distance Calculation	Edlib[6]	ASAP[7]

[1] H. Li. "Minimap2: Pairwise Alignment for Nucleotide Sequences." In *Bioinformatics,* 2018.
[2] H. Li. "Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM." In *arXiv,* 2013.
[3] Y. Turakhia et al. "Darwin: A genomics co-processor provides up to 15,000 x acceleration on long read assembly." In *ASPLOS,* 2018.
[4] D. Fujiki et al. "GenAx: A genome sequencing accelerator." In *ISCA*, 2018.
[5] M. Alser. "Shouji: A fast and efficient pre-alignment filter for sequence alignment." In *Bioinformatics,* 2019.
[6] M. Šošić et al. "Edlib: A C/C++ library for fast, exact sequence alignment using edit distance." In *Bioinformatics,* 2017.
[7] S.S. Banerjee et al. "ASAP: Accelerated short-read alignment on programmable hardware." In *TC*, 2018.

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

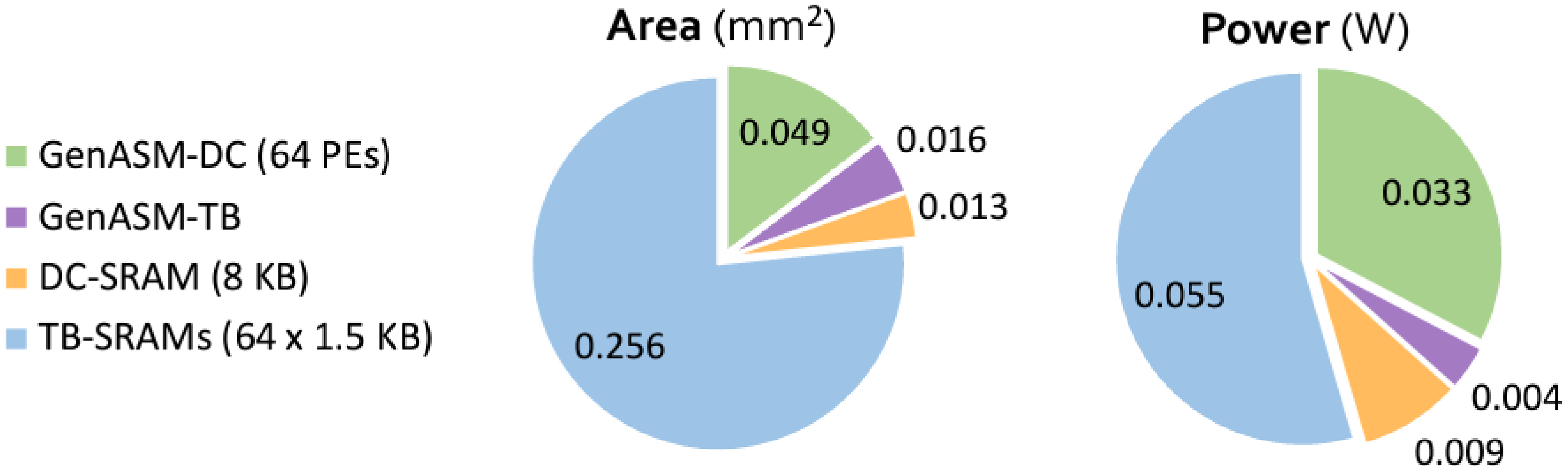
Evaluation Methodology (cont'd.)

- **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

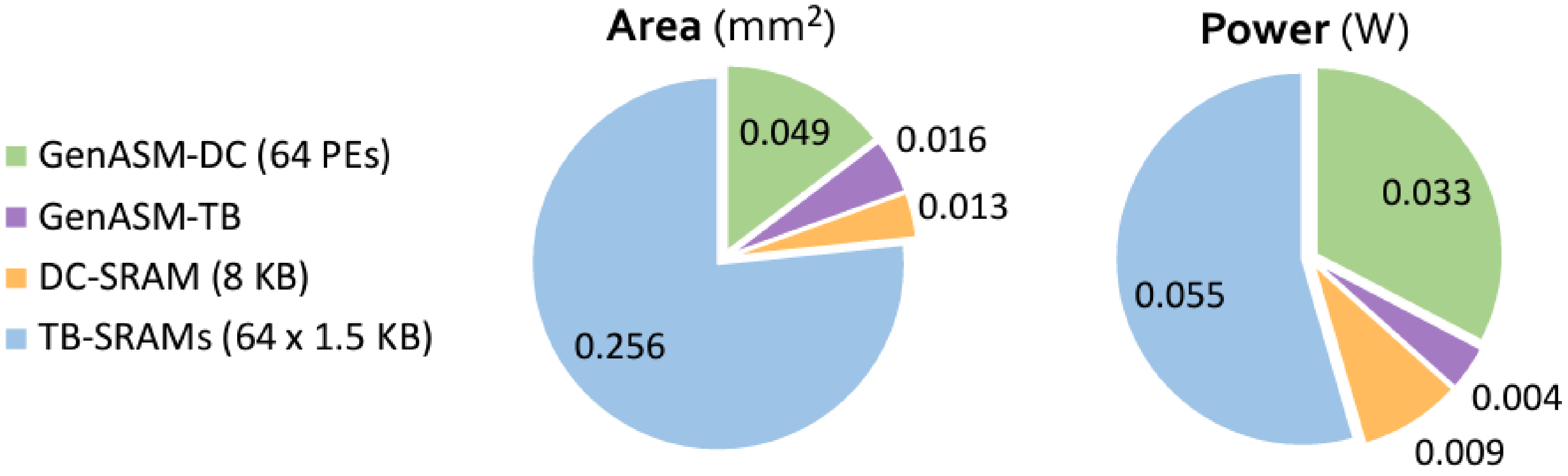
- Based on our **synthesis** of **GenASM-DC** and **GenASM-TB** accelerator datapaths using the Synopsys Design Compiler with a **28nm** process:
  - Both GenASM-DC and GenASM-TB operate **@ 1GHz**



Total (1 vault):	0.334 mm^2	0.101 W
Total (32 vaults):	10.69 mm^2	3.23 W
% of a Xeon CPU core:	**1%**	**1%**

Key Results – Area and Power

- Based on our **synthesis** of **GenASM-DC** and **GenASM-TB** accelerator datapaths using the Synopsys Design Compiler with a **28nm** 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

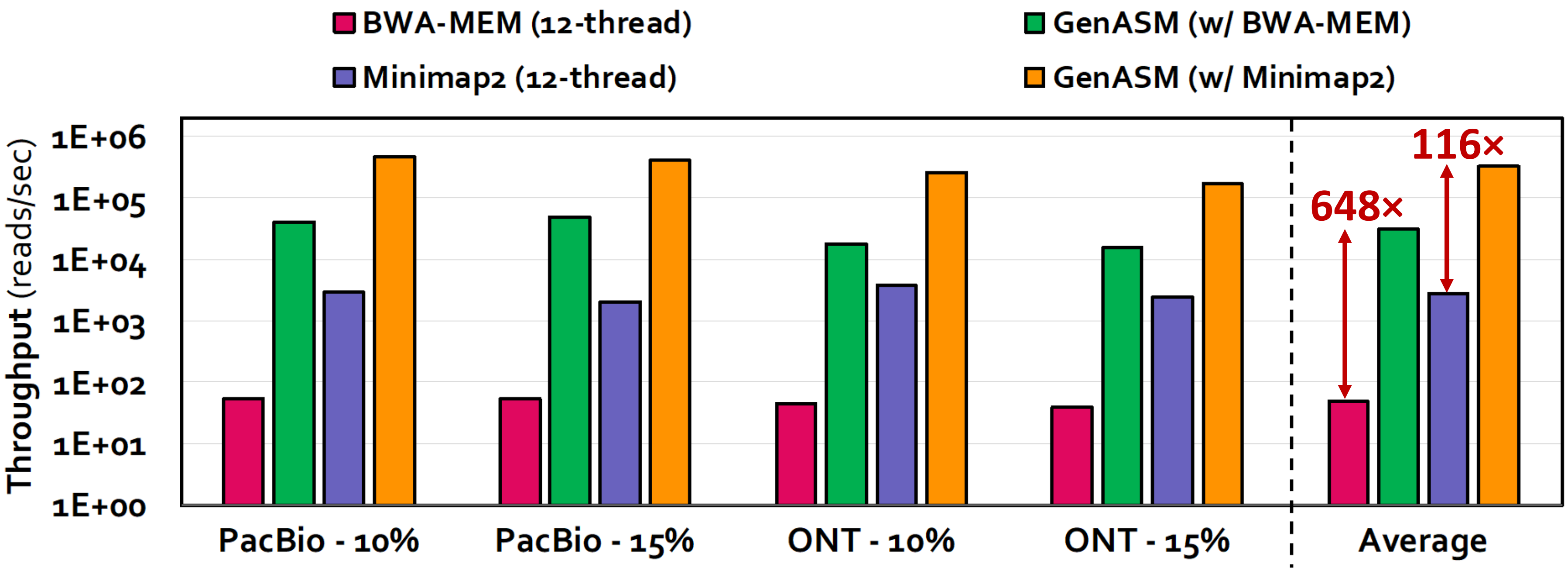
(2) Pre-Alignment Filtering for Short Reads

- 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

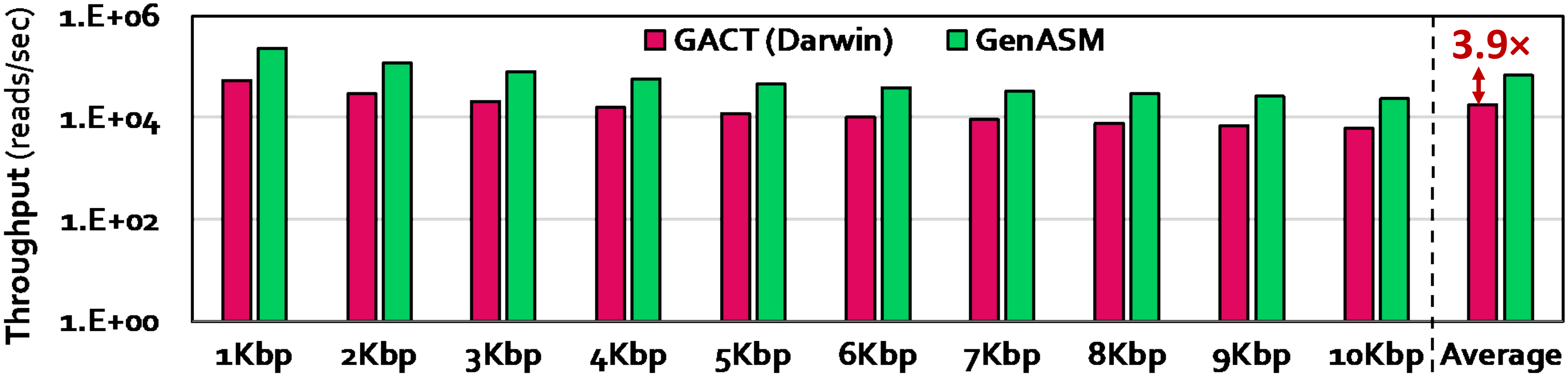
Key Results – Use Case 1 (Long Reads)



SW

GenASM **achieves 648× and 116× speedup** over 12-thread runs of BWA-MEM and Minimap2, while **reducing power consumption by 34× and 37×**

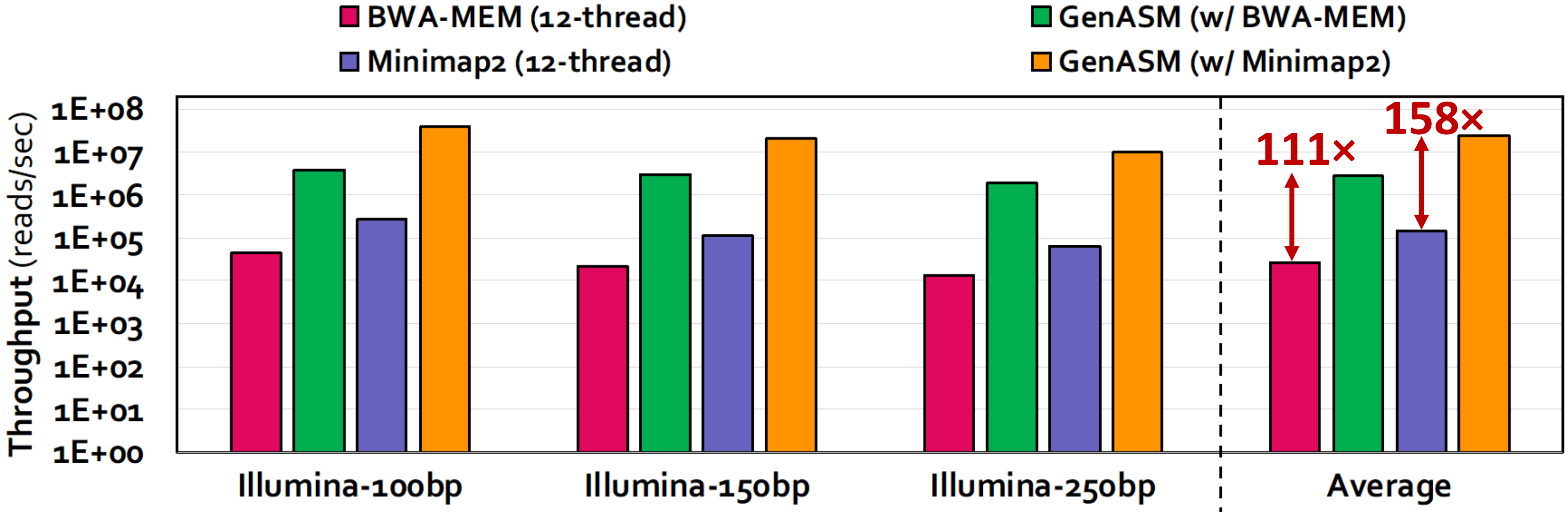
Key Results – Use Case 1 (Long Reads)



HW

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

HW

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

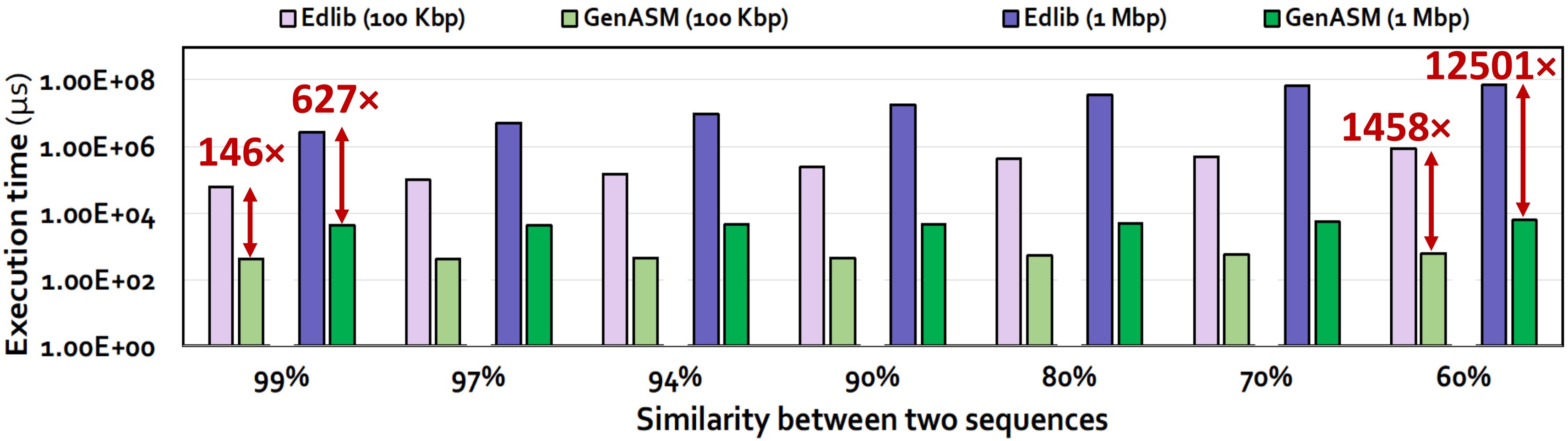
(2) Pre-Alignment Filtering for Short Reads

- 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



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

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

(2) Pre-Alignment Filtering

- ☐ **3.7×** speedup, **1.7×** less power than **Shouji** (state-of-the-art **HW**)

(3) Edit Distance Calculation

- ☐ **22–12501×** speedup, **548–582×** less power than **Edlib** (state-of-the-art **SW**)
- ☐ **9.3–400×** speedup, **67×** less power than **ASAP** (state-of-the-art **HW**)

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

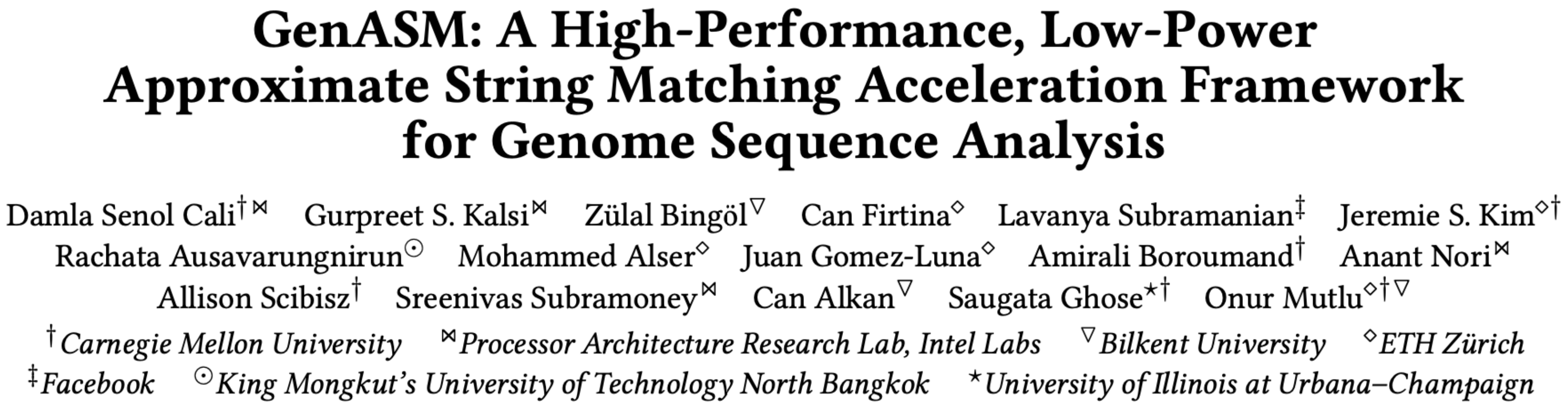
- **Key Results:** GenASM is significantly more efficient for all the three use cases (in terms of throughput and throughput per unit power) than state-of-the-art software and hardware baselines

GenASM [MICRO 2020]

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

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

Proceedings of the 53rd International Symposium on Microarchitecture (***MICRO***), Virtual, October 2020.

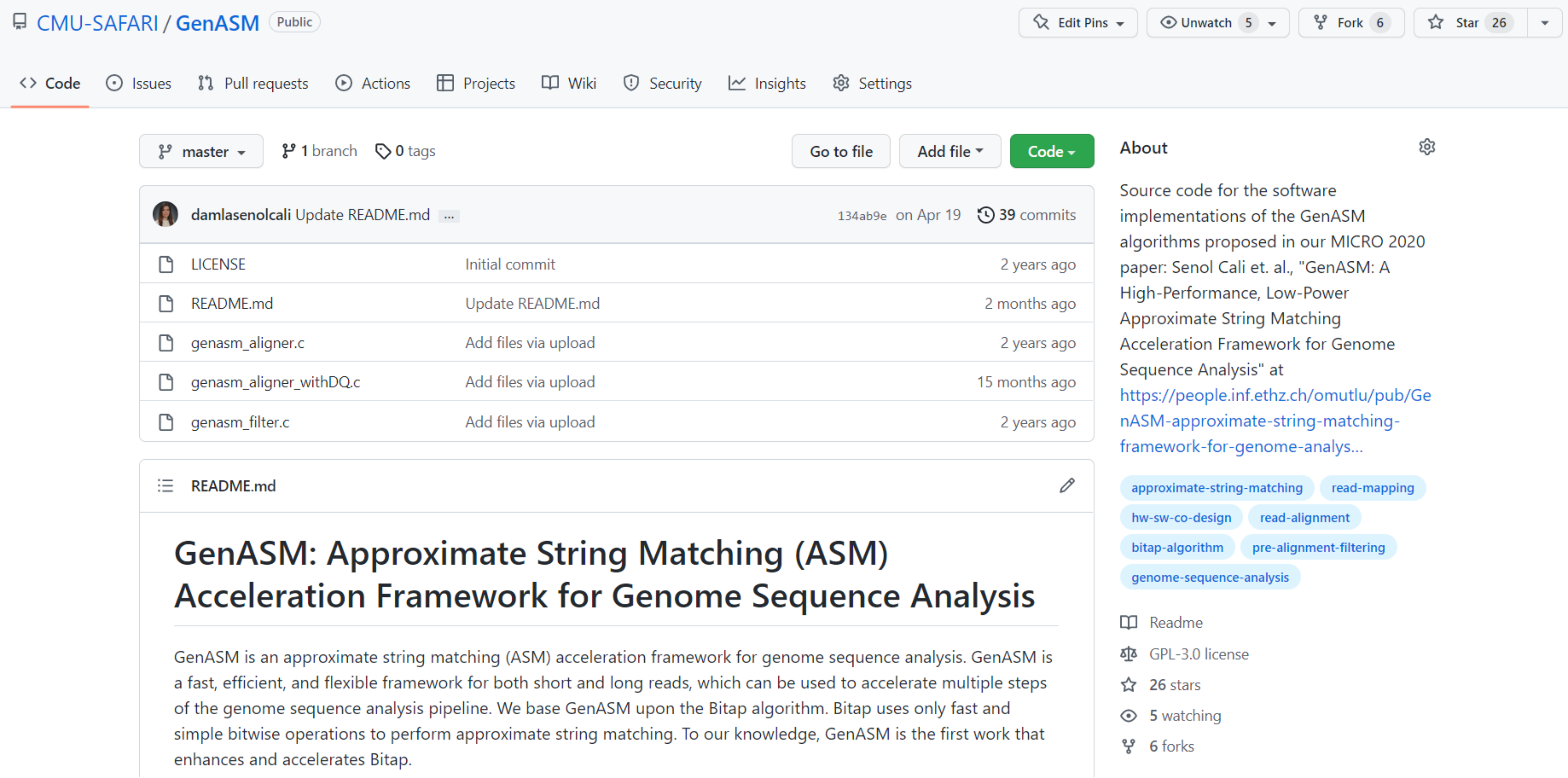


GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis

Damla Senol Cali[†⋈] Gurpreet S. Kalsi[⋈] Zülal 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*

GenASM – GitHub Page

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



damlasenolcali Update README.md … 134ab9e on Apr 19 39 commits

LICENSE	Initial commit	2 years ago
README.md	Update README.md	2 months ago
genasm_aligner.c	Add files via upload	2 years ago
genasm_aligner_withDQ.c	Add files via upload	15 months ago
genasm_filter.c	Add files via upload	2 years ago

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.

Source code for the software implementations of the GenASM algorithms proposed in our MICRO 2020 paper: Senol Cali et. al., "GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis" at https://people.inf.ethz.ch/omutlu/pub/GenASM-approximate-string-matching-framework-for-genome-analys...

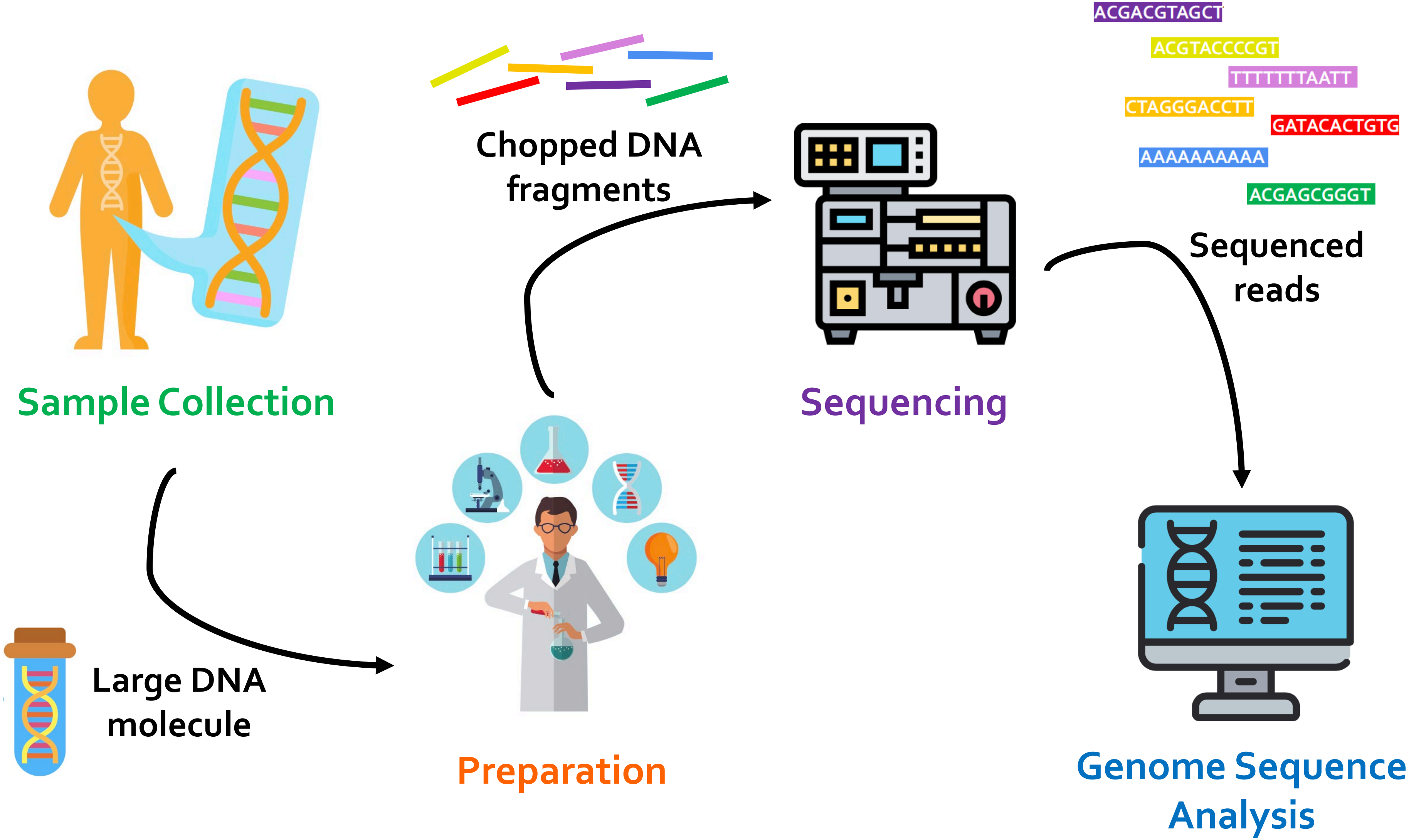
approximate-string-matching
read-mapping
hw-sw-co-design
read-alignment
bitap-algorithm
pre-alignment-filtering
genome-sequence-analysis

Readme
GPL-3.0 license
26 stars
5 watching
6 forks

Backup Slides

(Sequencing)

Genome Sequencing



Sample Collection

Large DNA molecule

Preparation

Chopped DNA fragments

Sequencing

ACGACGTAGCT
ACGTACCCCGT
TTTTTTTAATT
CTAGGGACCTT
GATACACTGTG
AAAAAAAAAA
ACGAGCGGGT

Sequenced reads

Genome Sequence Analysis

Sequencing Technologies

Oxford Nanopore
(ONT)

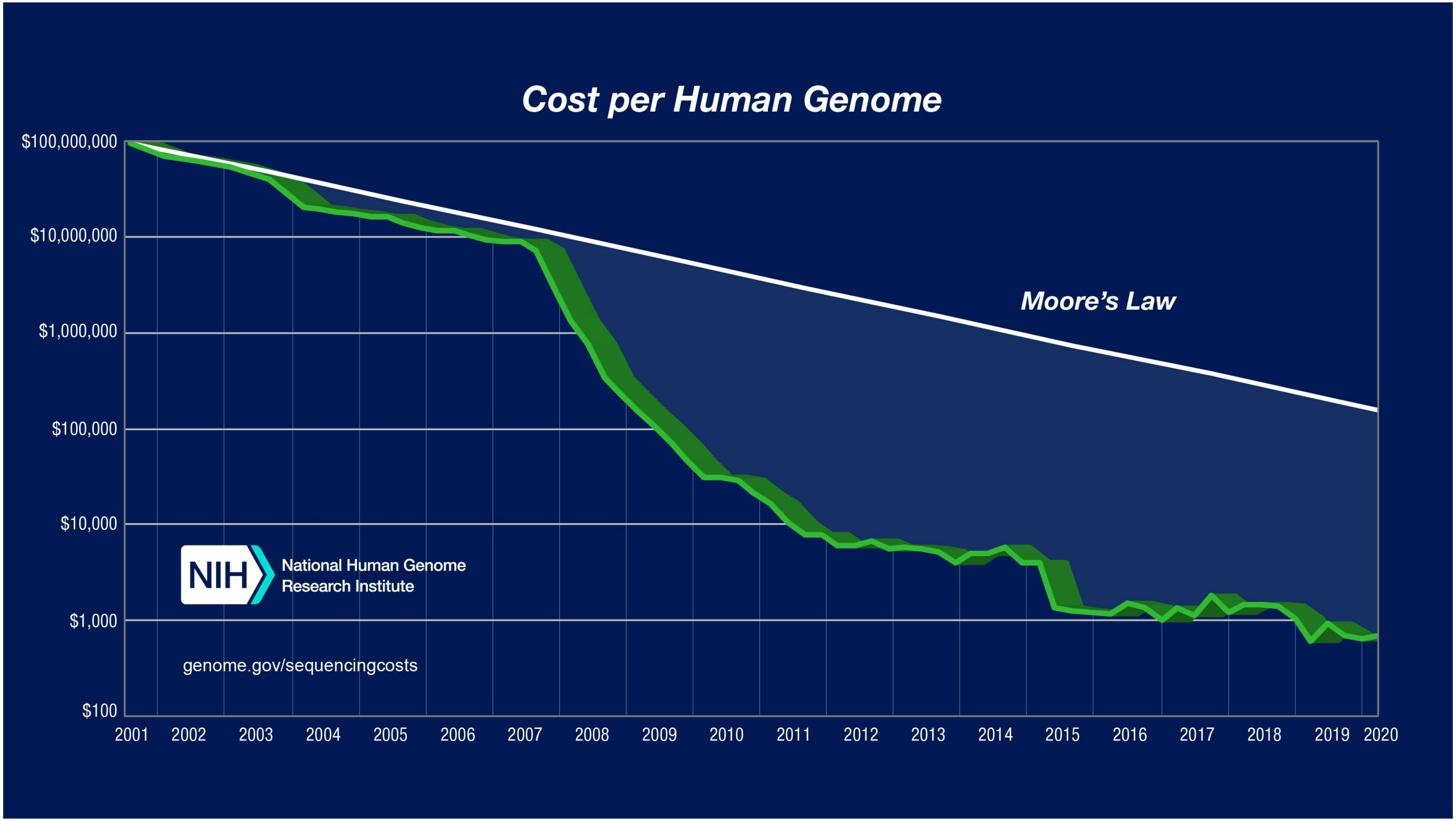
PacBio

Illumina

Short reads: **a few hundred base pairs** and **error rate of ~0.1%**

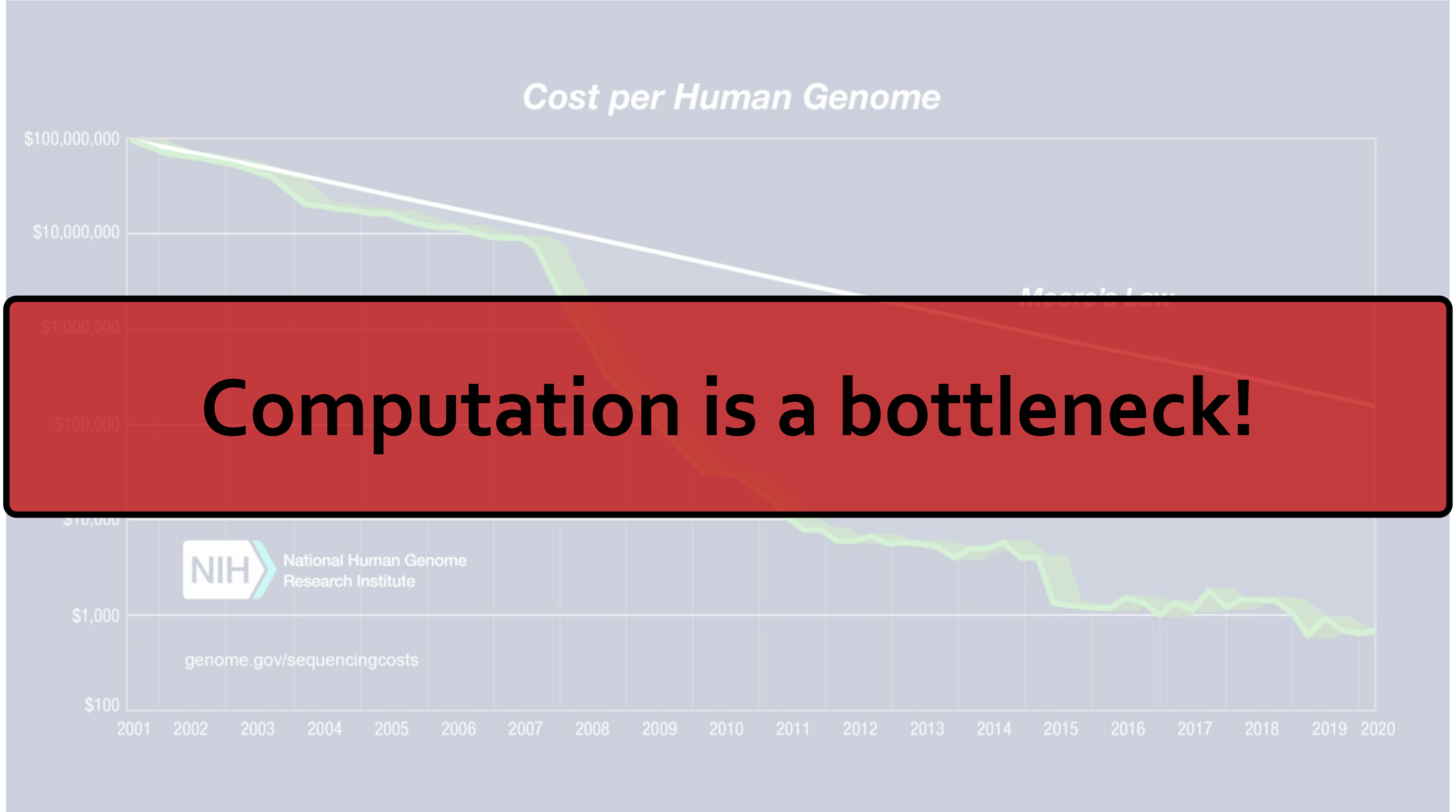
Long reads: **thousands to millions of base pairs** and **error rate of 5–10%**

Current State of Sequencing (cont'd.)



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

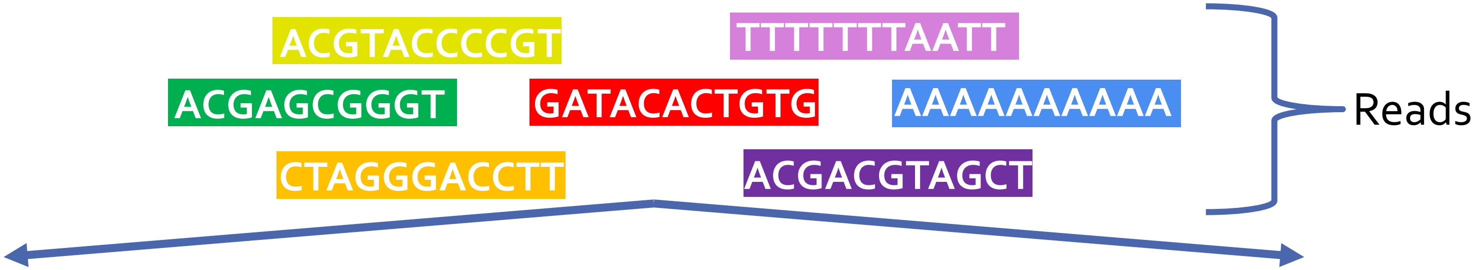
Current State of Sequencing (cont'd.)



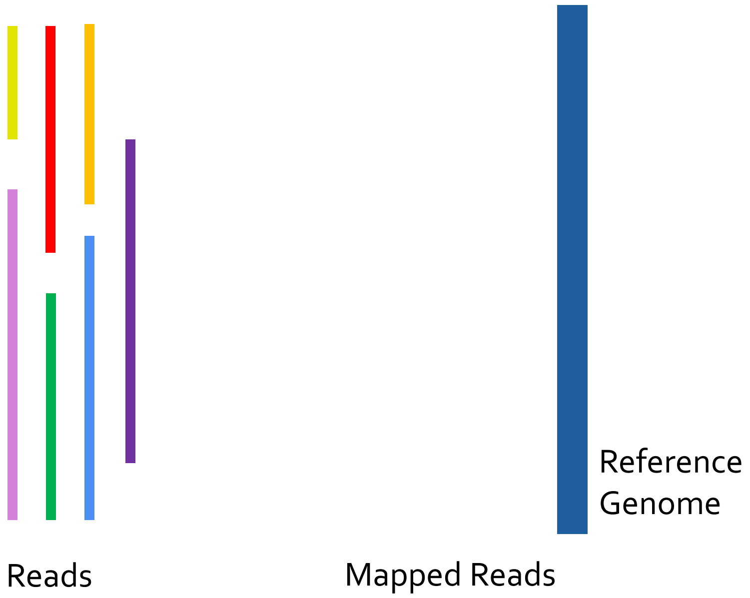
Computation is a bottleneck!

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

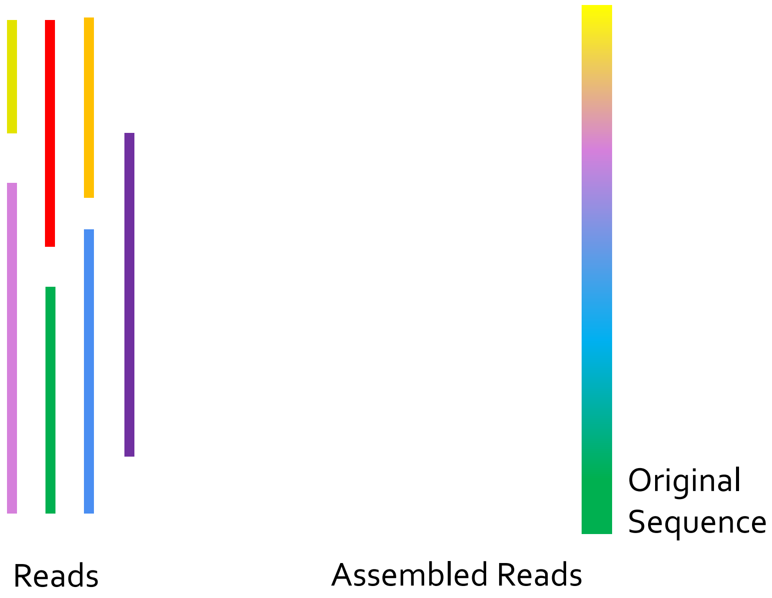
Genome Sequence Analysis



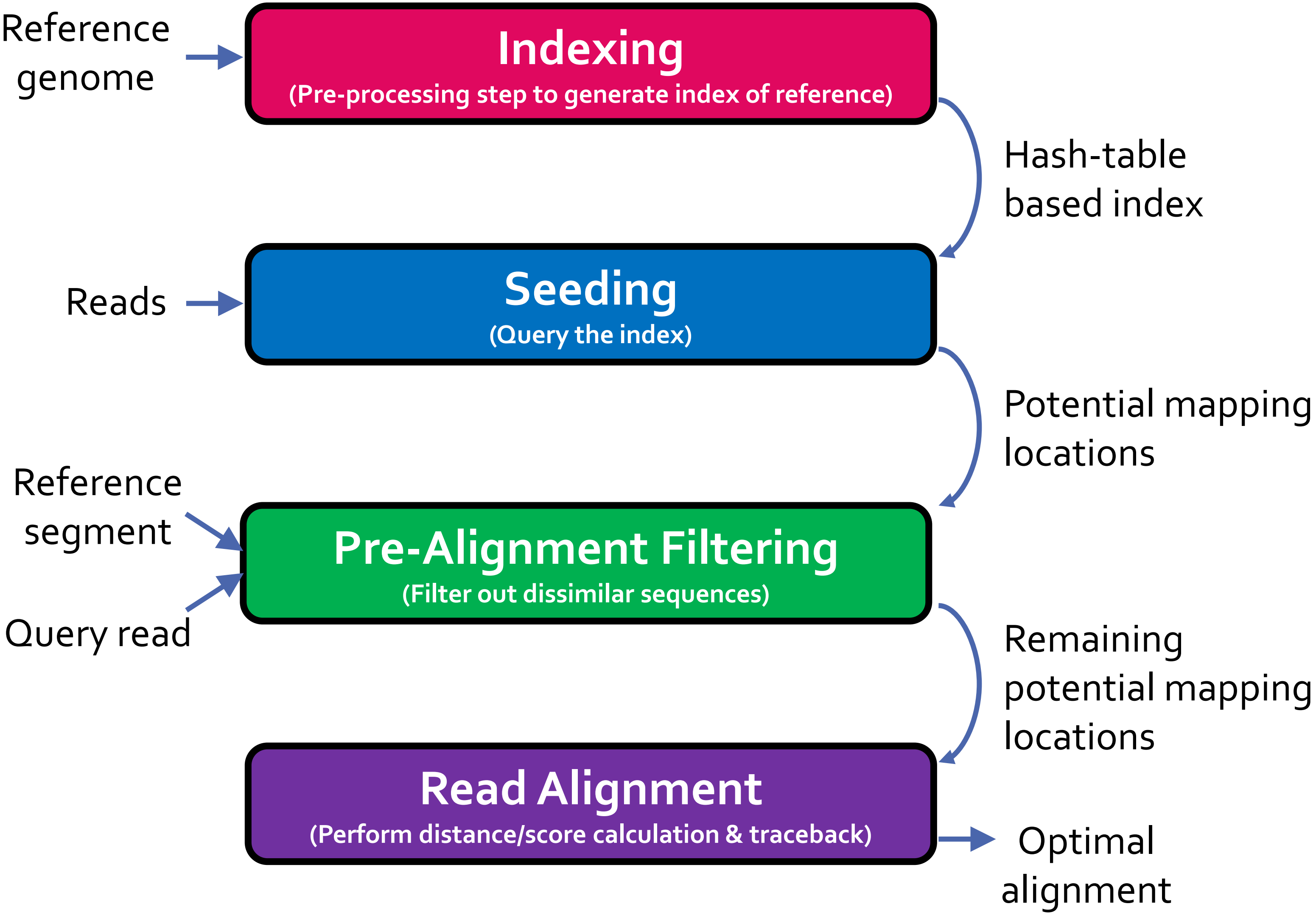
Read Mapping, method of aligning the reads against the reference genome in order to **detect matches and variations.**



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



Read Mapping Pipeline



Reference genome

Indexing

(Pre-processing step to generate index of reference)

Hash-table based index

Reads

Seeding

(Query the index)

Potential mapping locations

Reference segment

Query read

Pre-Alignment Filtering

(Filter out dissimilar sequences)

Remaining potential mapping locations

Read Alignment

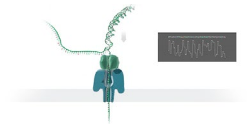
(Perform distance/score calculation & traceback)

Optimal alignment

Genome Assembly Pipeline Using Long Reads

- With the emergence of long read sequencing technologies, *de novo* assembly becomes a promising way of constructing the original genome.

Raw signal data →



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

DNA reads

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

Overlaps

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

← Assembly

Draft assembly

Read Mapping
(Maps the reads to the draft assembly)

Mappings of reads against draft assembly

Polishing
(Polishes the draft assembly & increases the accuracy)

← 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

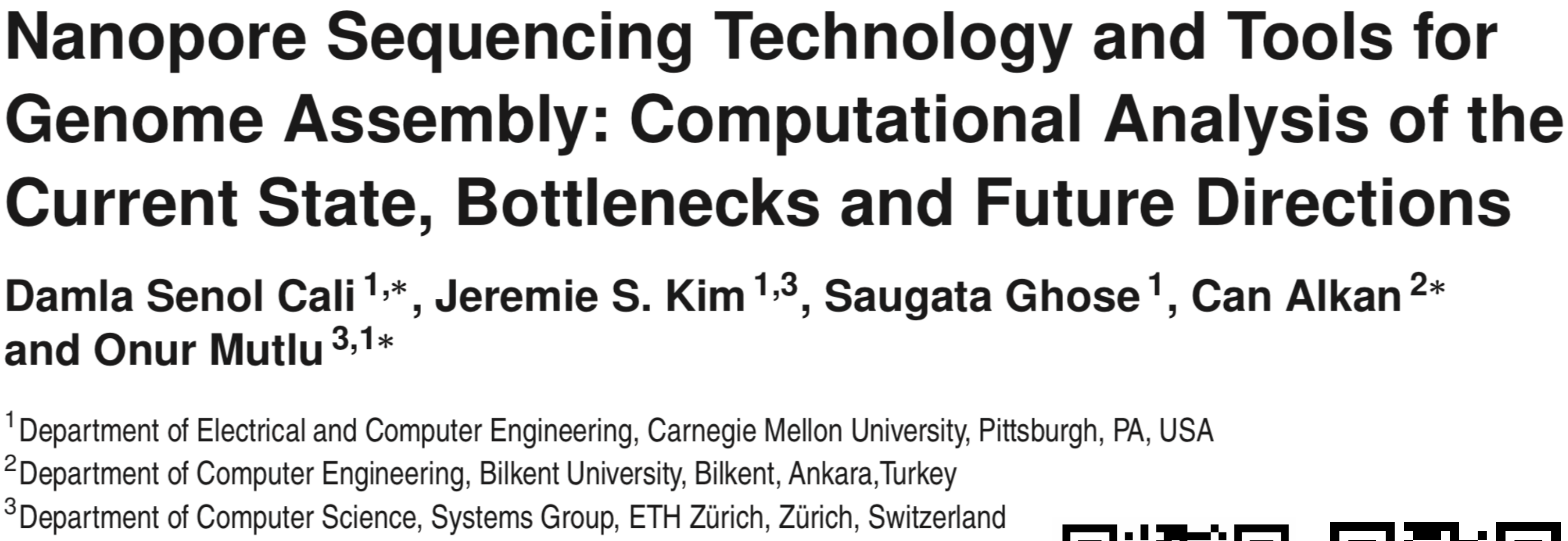
Key Findings

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 Cali [1,*], Jeremie S. Kim [1,3], Saugata Ghose [1], Can Alkan [2*] and Onur Mutlu [3,1*]

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

Damla Senol Cali, Jeremie S. Kim, Saugata Ghose, Can Alkan, and Onur Mutlu. **"Nanopore Sequencing Technology and Tools for Genome Assembly: Computational Analysis of the Current State, Bottlenecks and Future Directions."** *Briefings in Bioinformatics* (2018).

BiB Version

arXiv Version