

SeGraM

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

Damla Senol Cali, Konstantinos Kanellopoulos, Joël 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

intel®

ETH zürich

UNIVERSITY OF
ILLINOIS
URBANA-CHAMPAIGN



Bilkent University

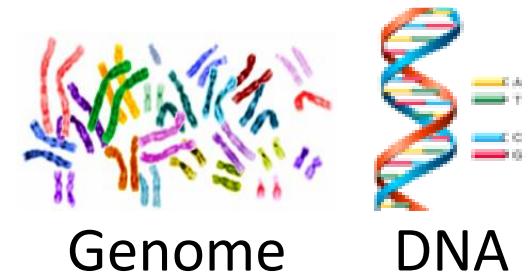
SAFARI

December 12 2023

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: ACGG

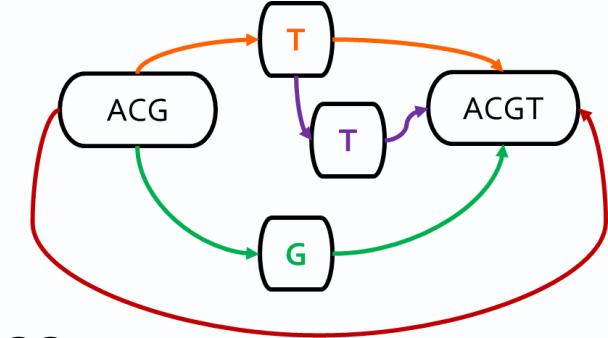
Alternative Sequence: ACGGACGT

Alternative Sequence: ACGTTACGT

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

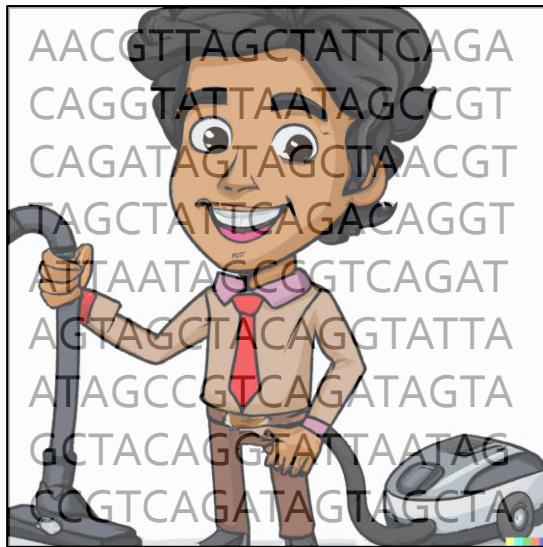
- ❑ **1.2×/4.8×** higher throughput than **GenASM** and **GACT** of **Darwin**
for long reads (state-of-the-art **HW**)
- ❑ **1.3×/2.4×** higher throughput than **GenASM** and **SillaX** of **GenAX**
for short reads (state-of-the-art **HW**)

Outline

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

Solving the Puzzle (S2S Mapping)

Reference Genome



For a Human:
3 Billion Characters (3GB)

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

Solving the Puzzle (S2S Mapping)

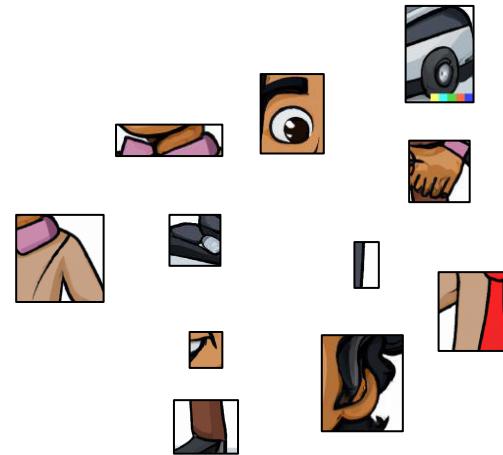
Reference Genome



For a Human:
3 Billion Characters (3GB)

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

Reads



150 – 2,000,000 Characters Each

Origin Locations are Unknown

Solving the Puzzle (S₂S Mapping)

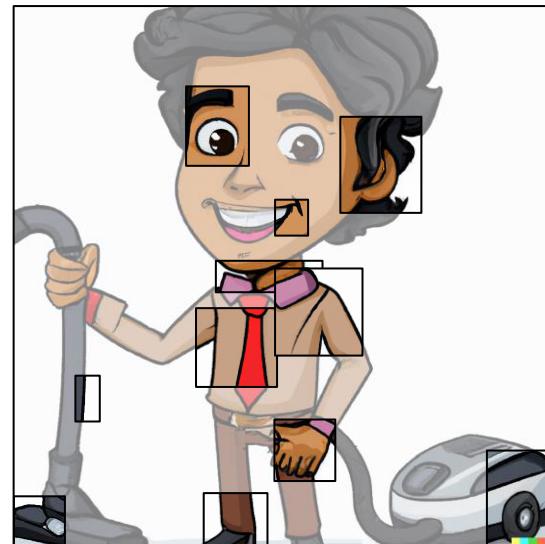
Reference Genome



For a Human:
3 Billion Characters (3GB)

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

Reads



150 – 2,000,000 Characters Each

Origin Locations are Unknown

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

S2S Alignment/Mapping Accelerators

SeedEx: A Genome Sequencing Accelerator for Optimal Alignments in Subminimal Space

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

GenAx: A Genome Sequencing Accelerator

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

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

Yatish Turakhia
Stanford University
yatish@stanford.edu

Gill Bejerano
Stanford University
bejerano@stanford.edu

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

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

Damla Senol Cali^{†✉} Gurpreet S. Kalsi[✉] Zülał Bingöl[▽] Can Firtina[◊] Lavanya Subramanian[‡] Jeremie S. Kim^{◊†}
Rachata Ausavarungnirun[○] Mohammed Alser[◊] Juan Gomez-Luna[◊] Amirali Boroumand[†] Anant Nori[✉]
Allison Scibisz[†] Sreenivas Subramoney[✉] Can Alkan[▽] Saugata Ghose^{*,†} Onur Mutlu^{◊†▽}

[†]Carnegie Mellon University [✉]Processor Architecture Research Lab, Intel Labs [▽]Bilkent University [◊]ETH Zürich
[‡]Facebook [○]King Mongkut's University of Technology North Bangkok ^{*}University of Illinois at Urbana-Champaign

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

Safaa Diab¹, Amir Nassereldine¹, Mohammed Alser², Juan Gómez Luna², Onur Mutlu², Izzat El Hajj¹

¹American University of Beirut, Lebanon ²ETH Zürich, Switzerland

Genome analysis

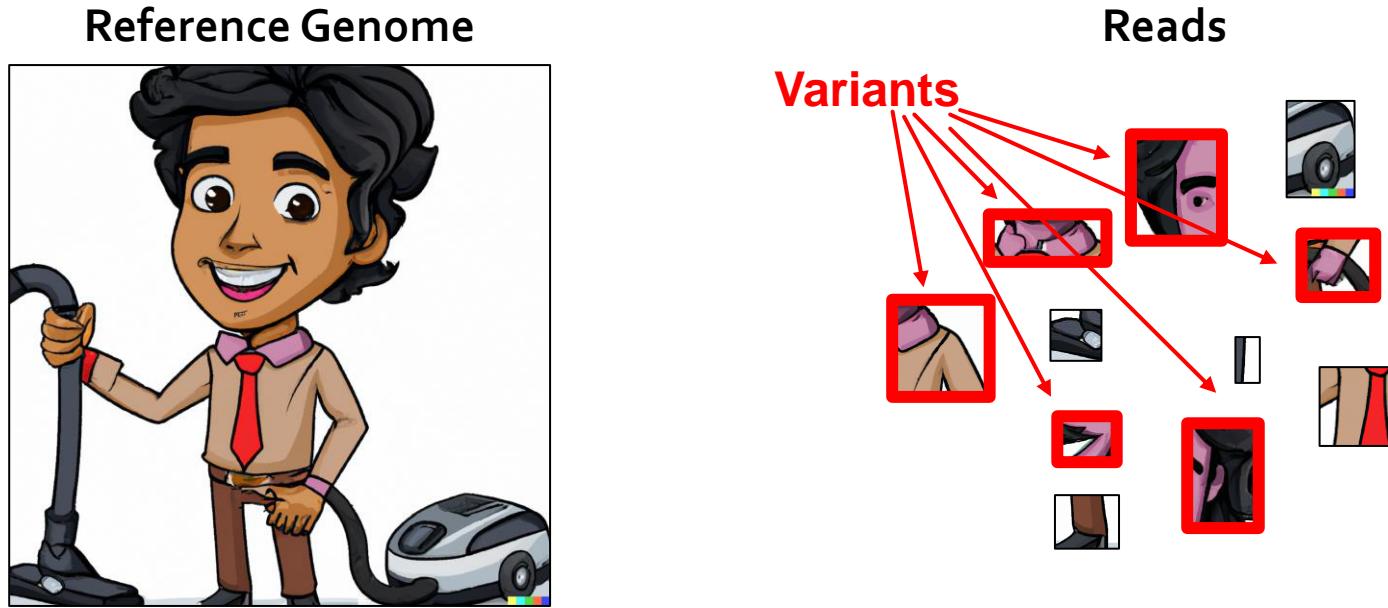
Scrooge: A Fast and Memory-Frugal Genomic Sequence Aligner for CPUs, GPUs, and ASICs

Joël Lindegger^{1,*}, Damla Senol Cali², Mohammed Alser¹, Juan Gómez-Luna¹, Nika Mansouri Ghiasi¹ and Onur Mutlu^{1,*}

¹Department of Information Technology and Electrical Engineering, ETH Zurich, Zurich 8006, Switzerland and

²Bionano Genomics, San Diego, CA 92121, USA.

Solving the Puzzle (S₂S Mapping)

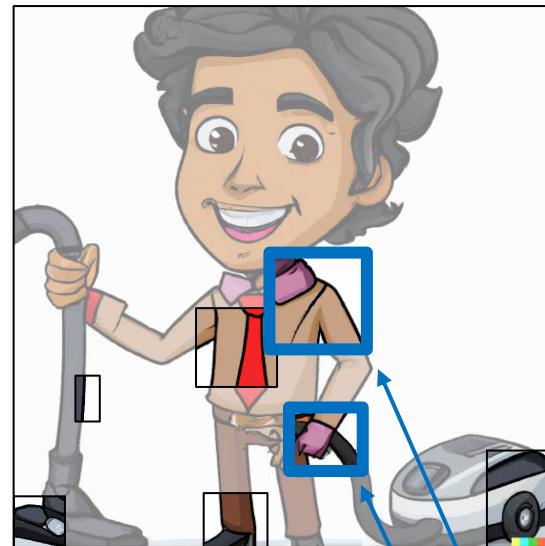


Solving the Puzzle (S₂S Mapping)

Reference Genome



Reads



Some Reads Can Be Mapped due to Sufficient Context

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



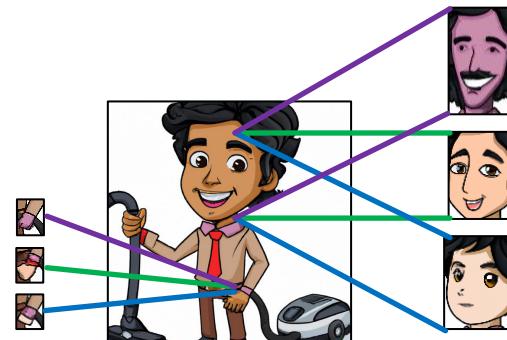
Reference Bias!

Avoiding Reference Bias in Read Mapping

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



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



Genome Graphs

Genome graphs:

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

Sequence #1: ACGTACGT

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

ACGTACGT

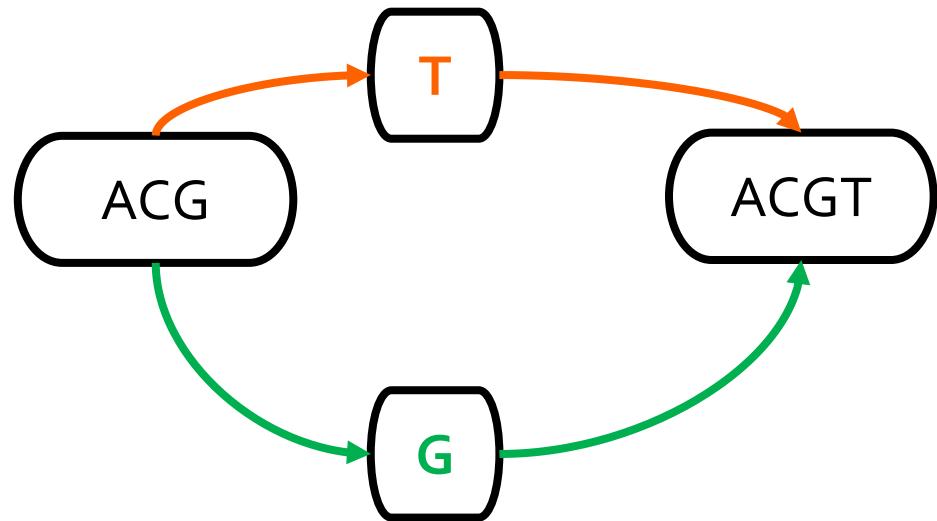
Genome Graphs

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

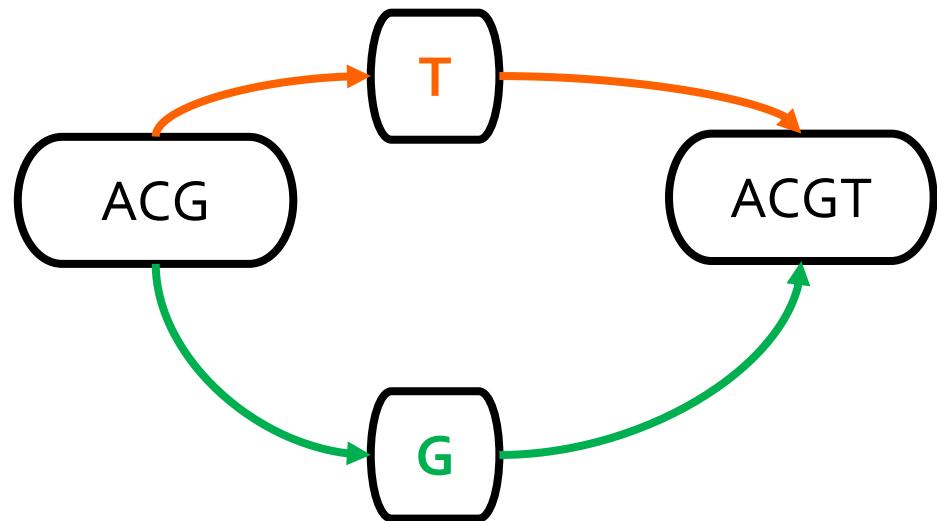
Genome graphs:

- 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

Sequence #3: ACGTTACGT



Genome Graphs

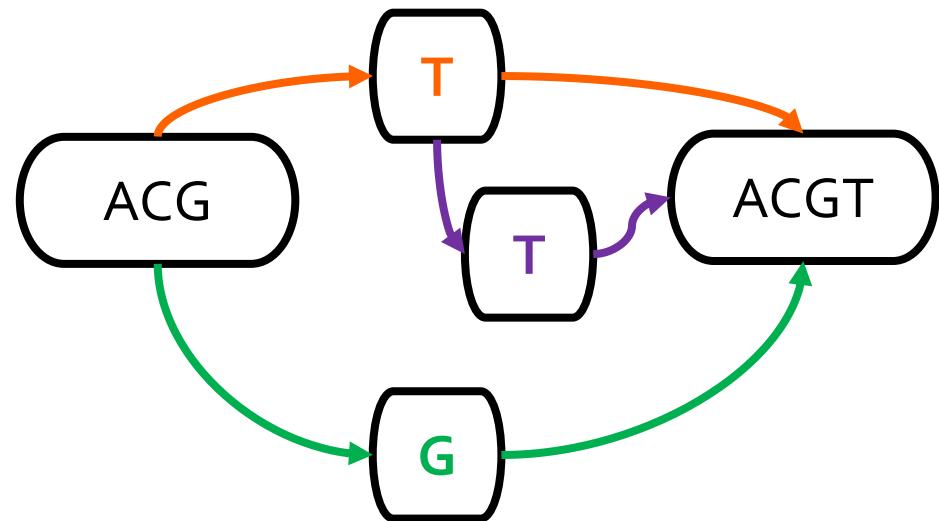
Genome graphs:

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

Sequence #1: ACGTACGT

Sequence #2: ACGGACGT

Sequence #3: ACGTTACGT



Genome Graphs

Genome graphs:

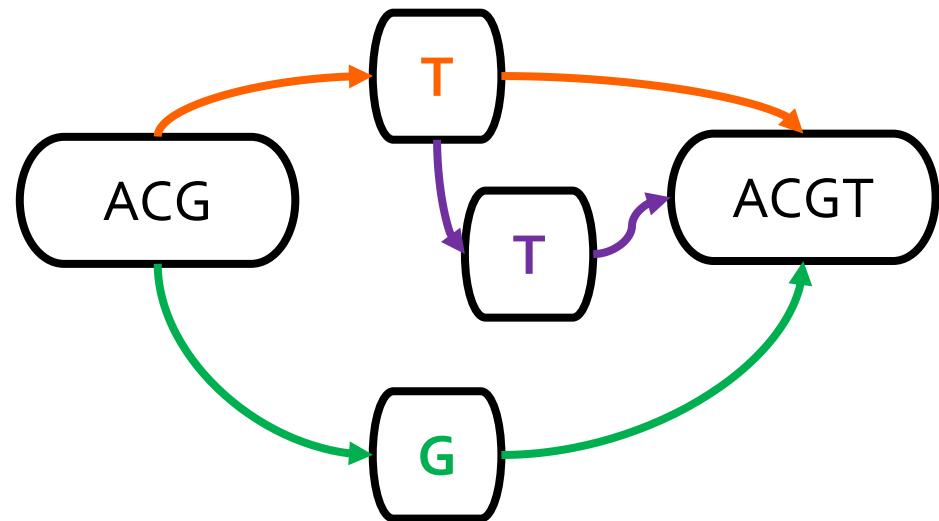
- Combine the **linear reference genome** with the **known genetic variations** in the entire population as a graph-based data structure
- Enable us to move away from aligning with a single linear reference genome (**reference bias**) and **more accurately express the genetic diversity** in a population

Sequence #1: ACGTACGT

Sequence #2: ACGGACGT

Sequence #3: ACGTTACGT

Sequence #4: ACGACGT



Genome Graphs

Genome graphs:

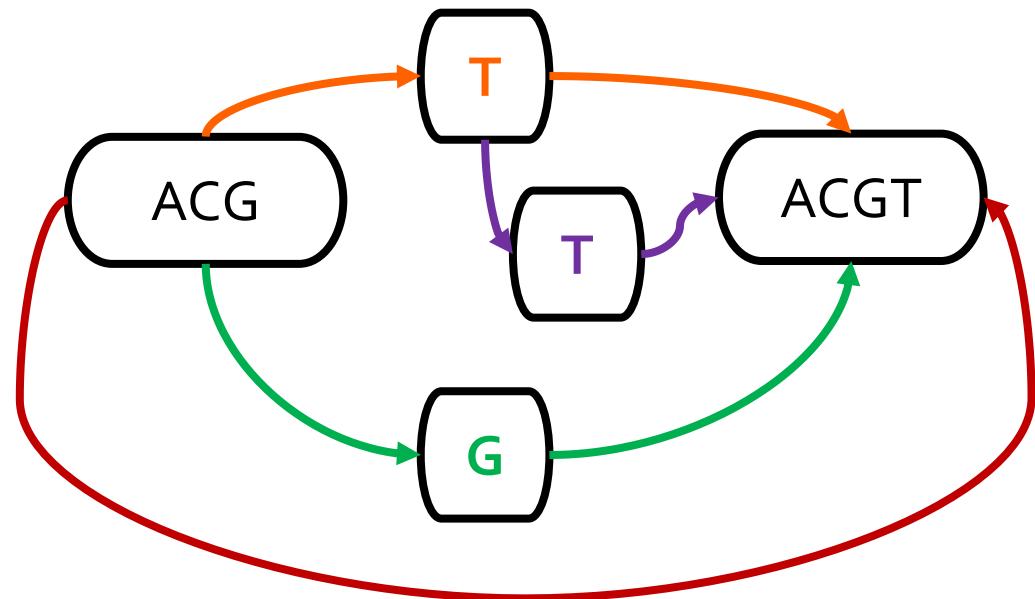
- Combine the **linear reference genome** with the **known genetic variations** in the entire population as a graph-based data structure
- Enable us to move away from aligning with a single linear reference genome (**reference bias**) and **more accurately express the genetic diversity** in a population

Sequence #1: ACGTACGT

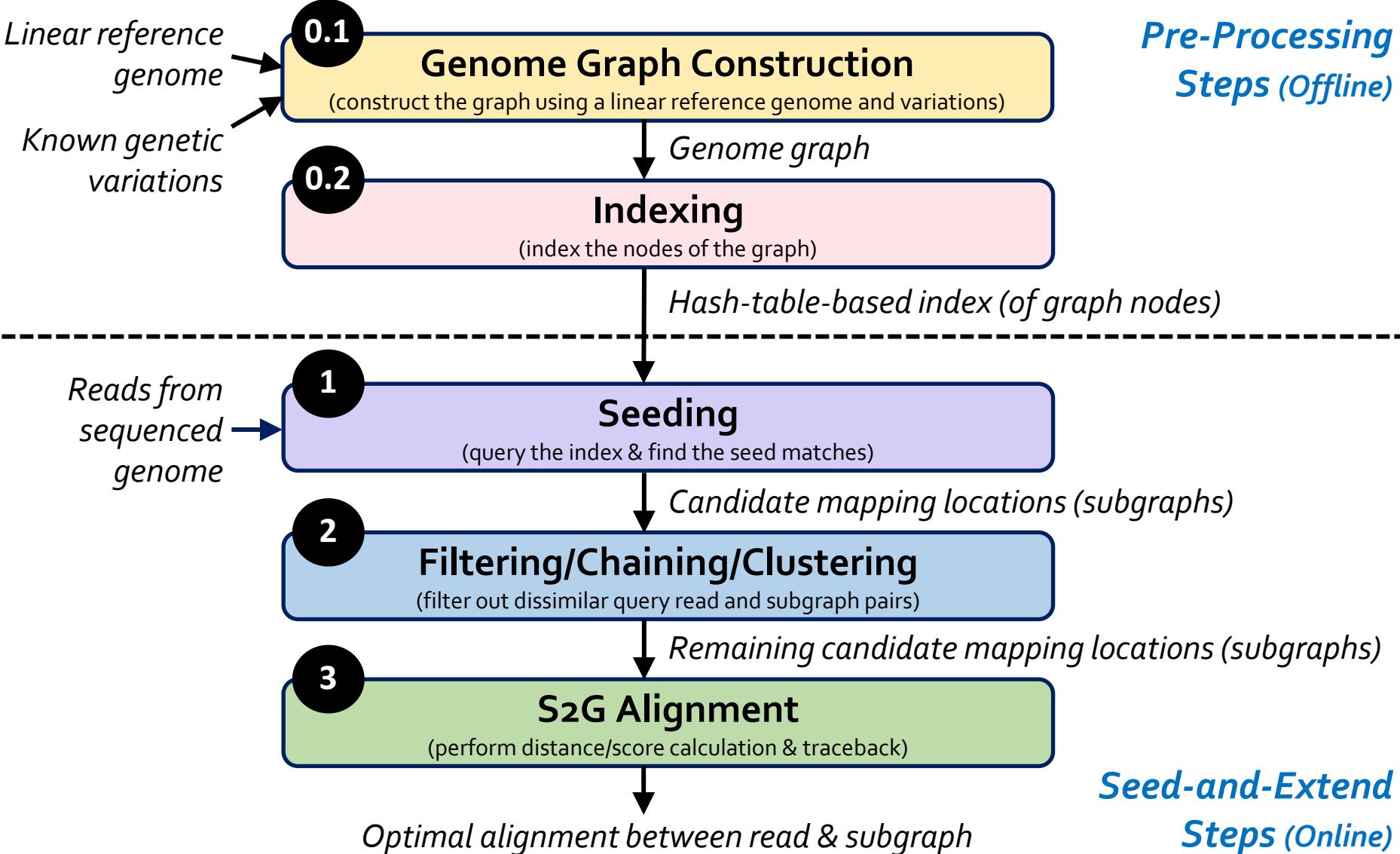
Sequence #2: ACGGACGT

Sequence #3: ACGTTACGT

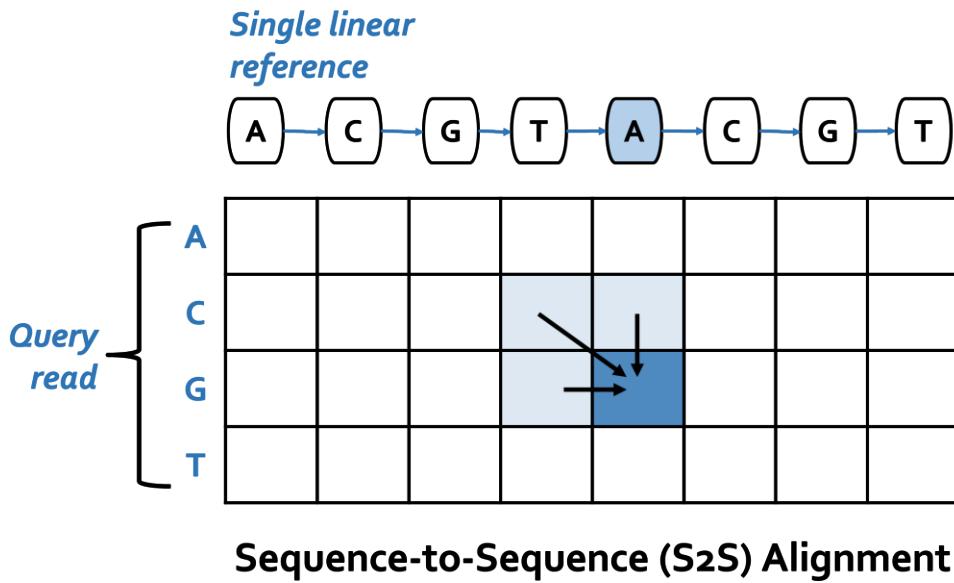
Sequence #4: ACGACGT



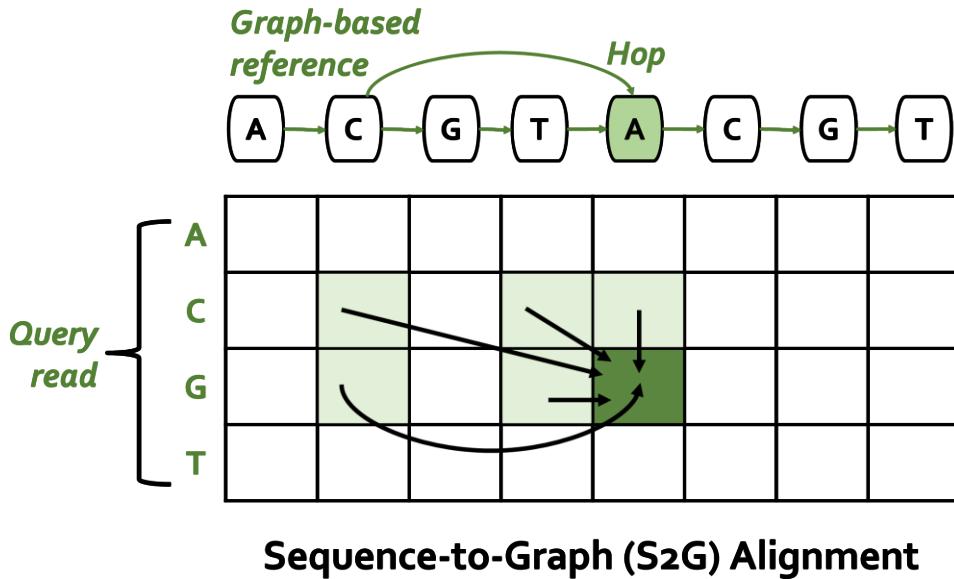
Sequence-to-Graph Mapping Pipeline



S₂S vs. S₂G Alignment



S₂S vs. 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

Based on our analysis with **GraphAligner** and **vg**:

SW

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

HW

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

Outline

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

SeGraM: Universal Genomic Mapping Accelerator

- ❑ *First universal genomic mapping accelerator* that can support *both* sequence-to-graph mapping and sequence-to-sequence mapping, for *both* short and long reads
- ❑ *First algorithm/hardware co-design* for accelerating sequence-to-graph mapping

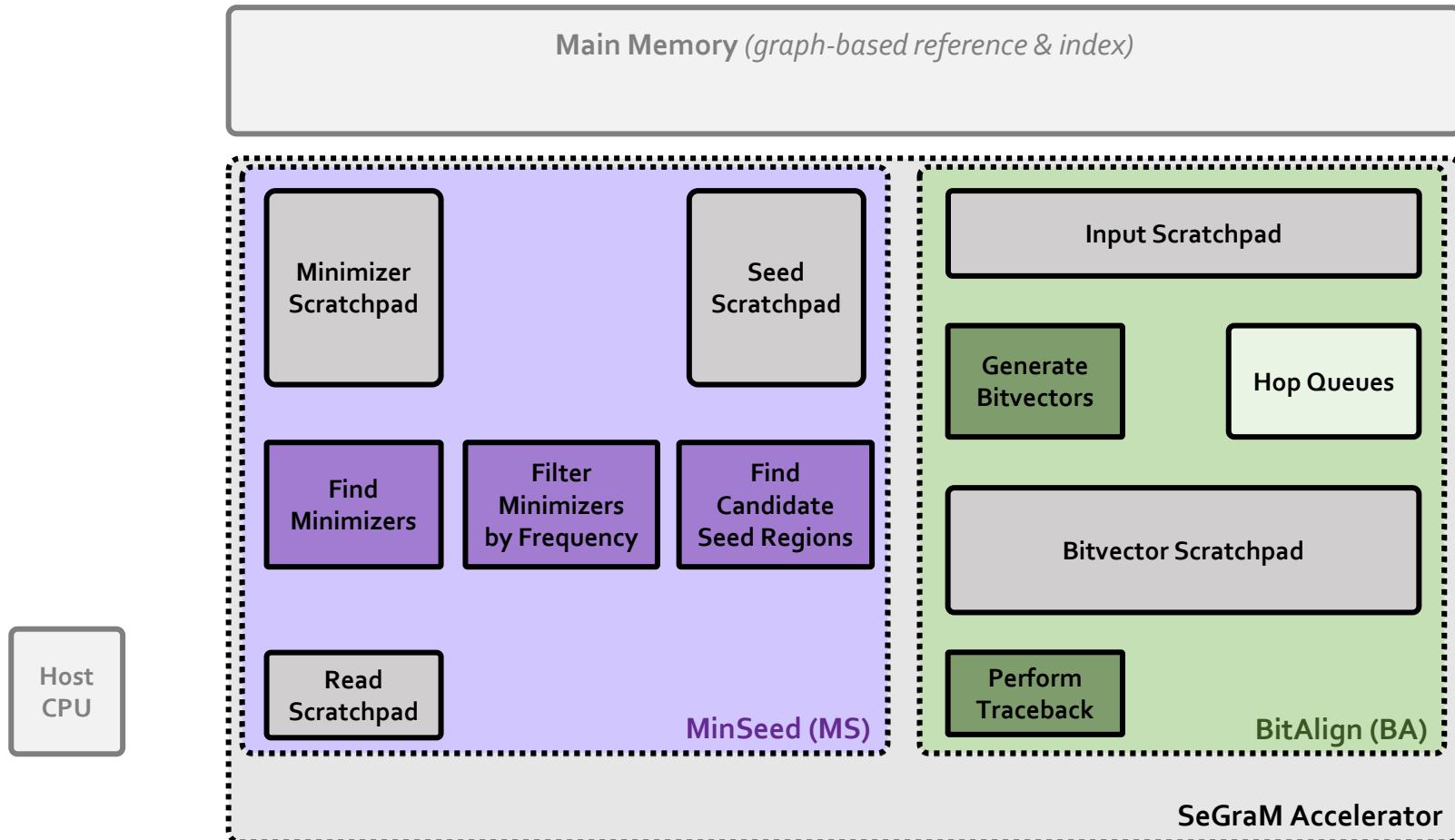
- ❑ We base SeGraM upon a **minimizer-based seeding algorithm**
- ❑ We propose a **novel bitvector-based alignment algorithm** to perform approximate string matching between a read and a graph-based reference genome

SW

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

HW

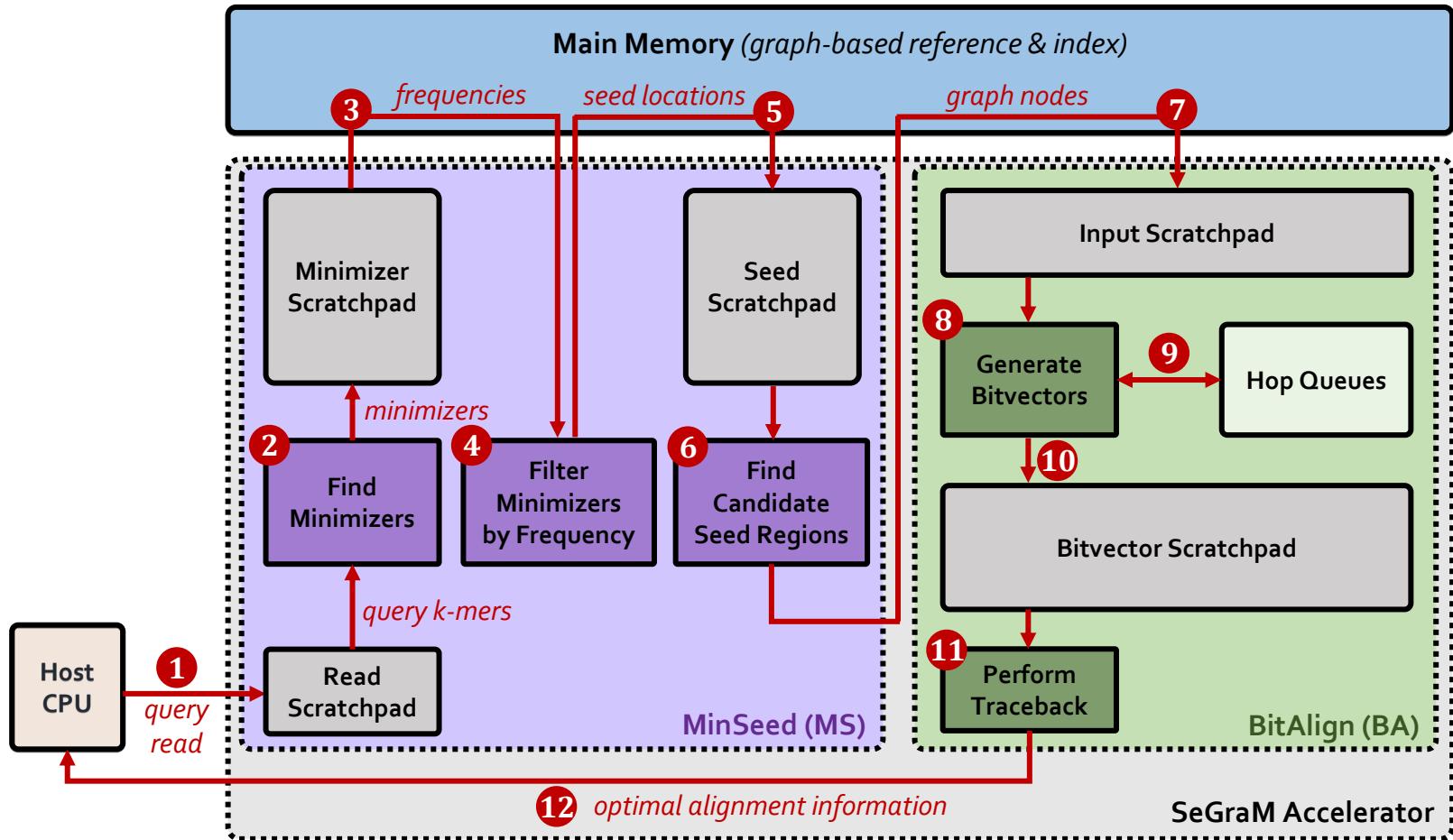
SeGraM Hardware Design



MinSeed: first hardware accelerator for **Minimizer-based Seeding**

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

SeGram Hardware Design

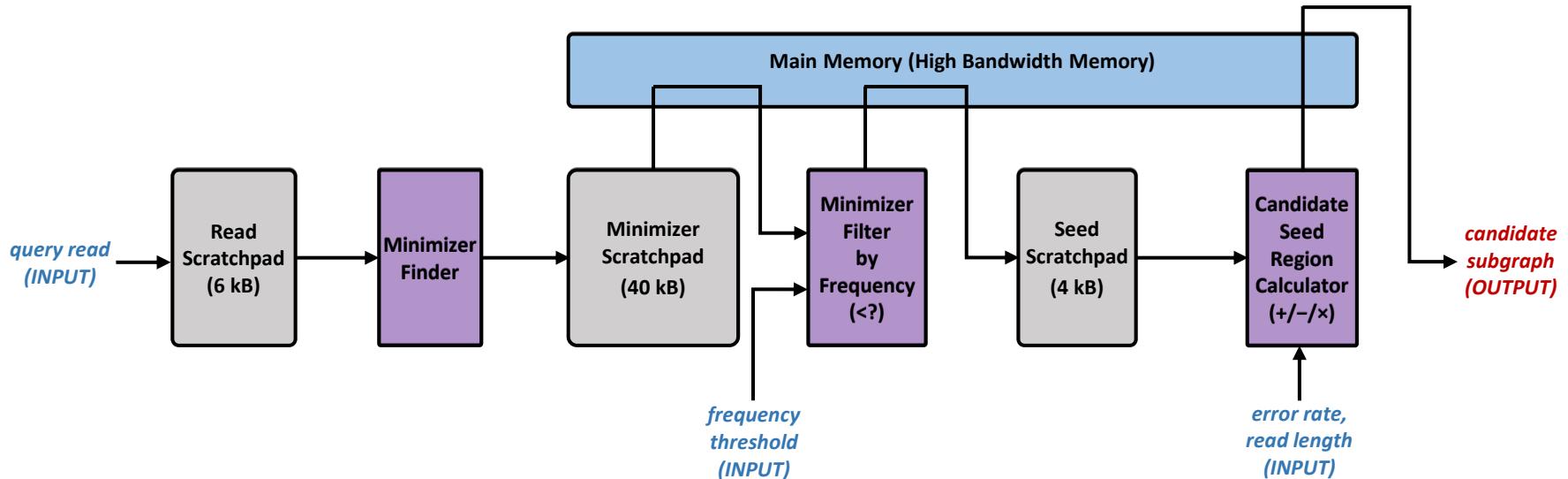


MinSeed: first hardware accelerator for **Minimizer-based Seeding**

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

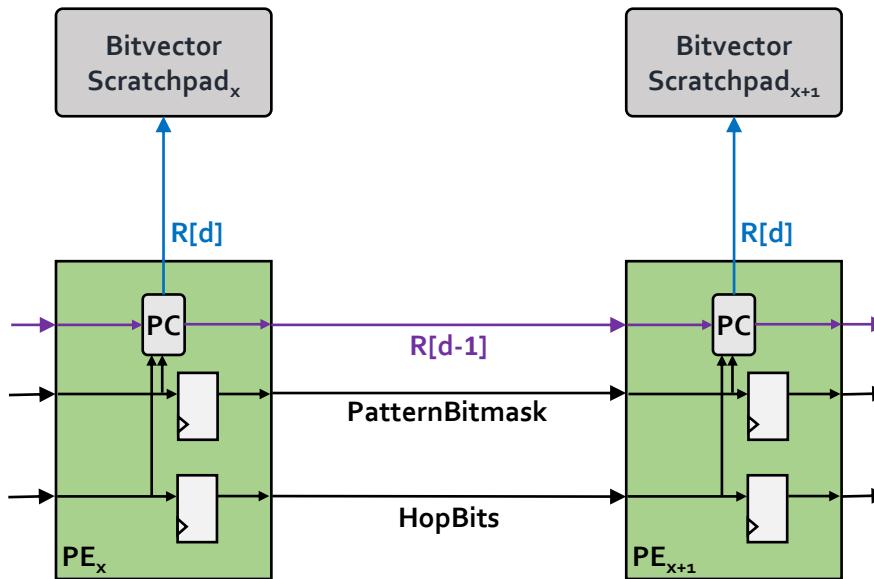
MinSeed HW

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



BitAlign HW

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



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

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

Damla Senol Cali^{†✉} Gurpreet S. Kalsi[✉] Zülfü 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*

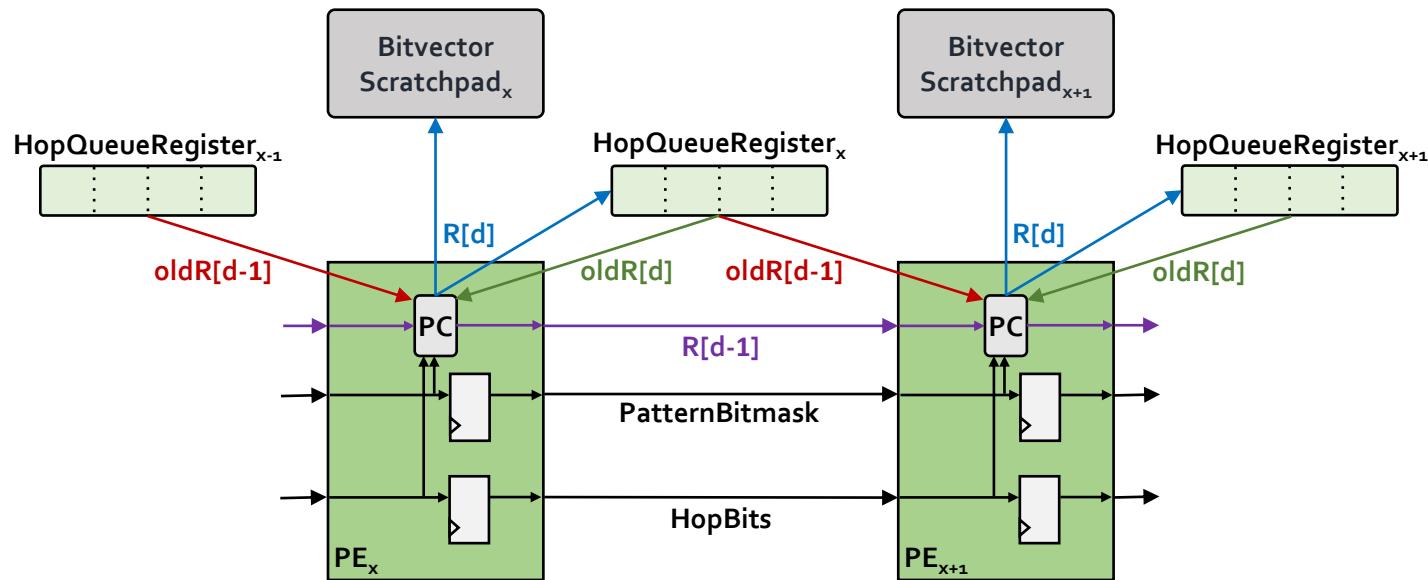
[‡]*Facebook* [○]*King Mongkut's University of Technology North Bangkok* ^{*}*University of Illinois at Urbana-Champaign*

<https://arxiv.org/abs/2009.07692>

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

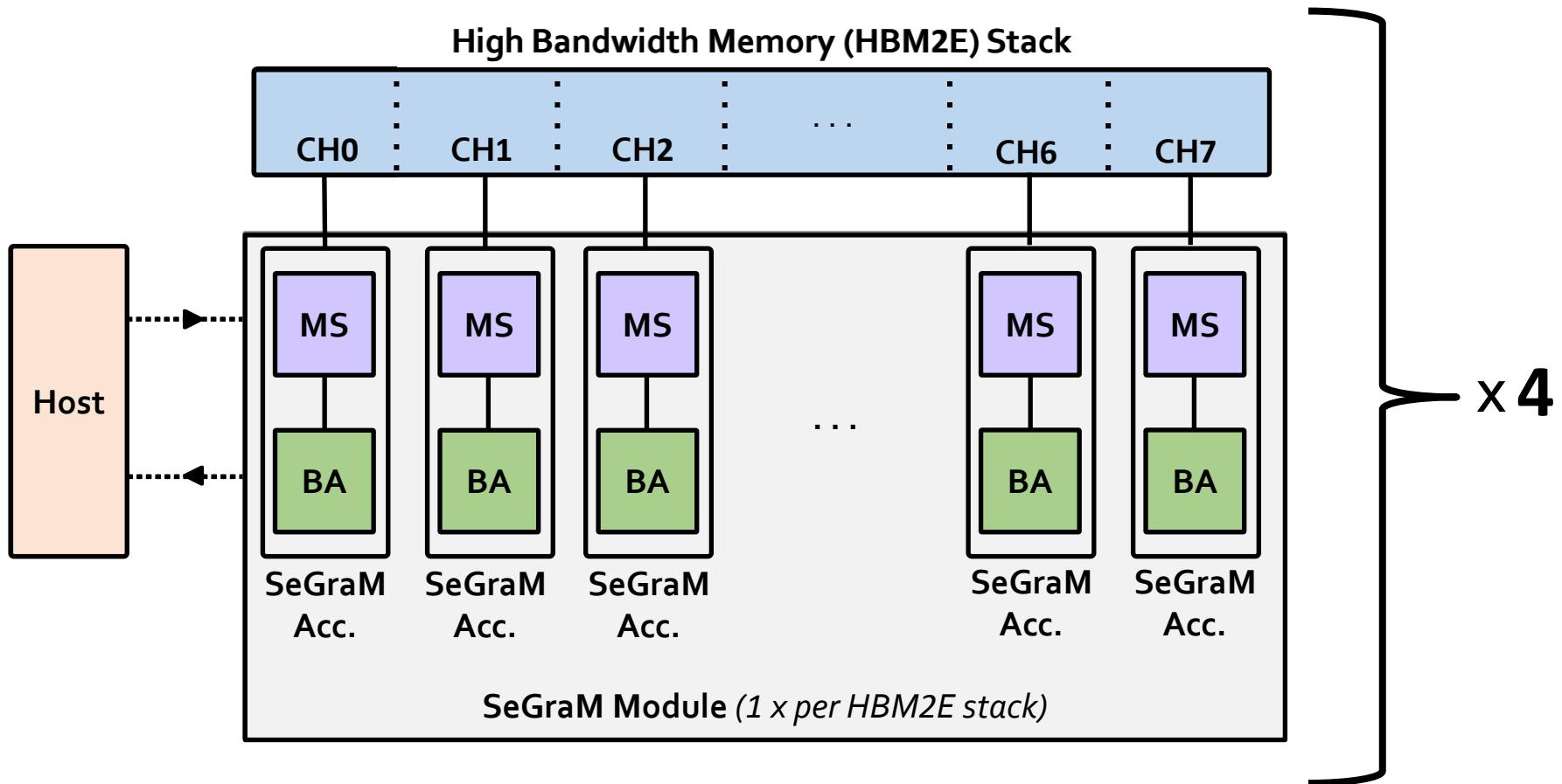
BitAlign HW

- 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*)



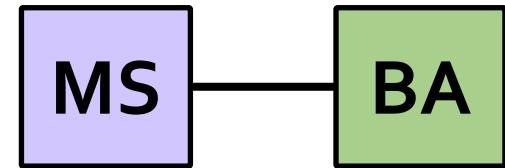
[*] D. Senol Cali *et al.* "[GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis](#)" (MICRO'20)

Overall System Design of SeGram

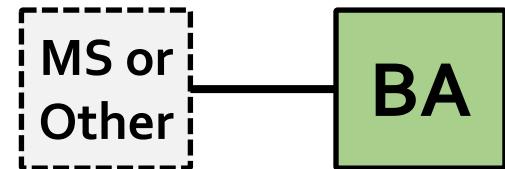


Use Cases of SeGram

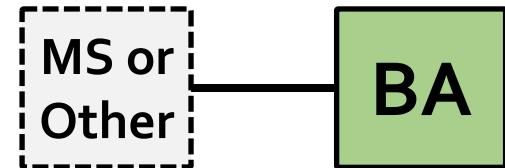
(1) Sequence-to-Graph
Mapping



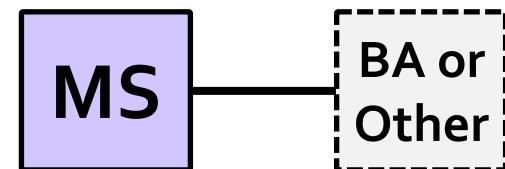
(2) Sequence-to-Graph
Alignment



(3) Sequence-to-Sequence
Alignment



(4) Seeding



Outline

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

Evaluation Methodology

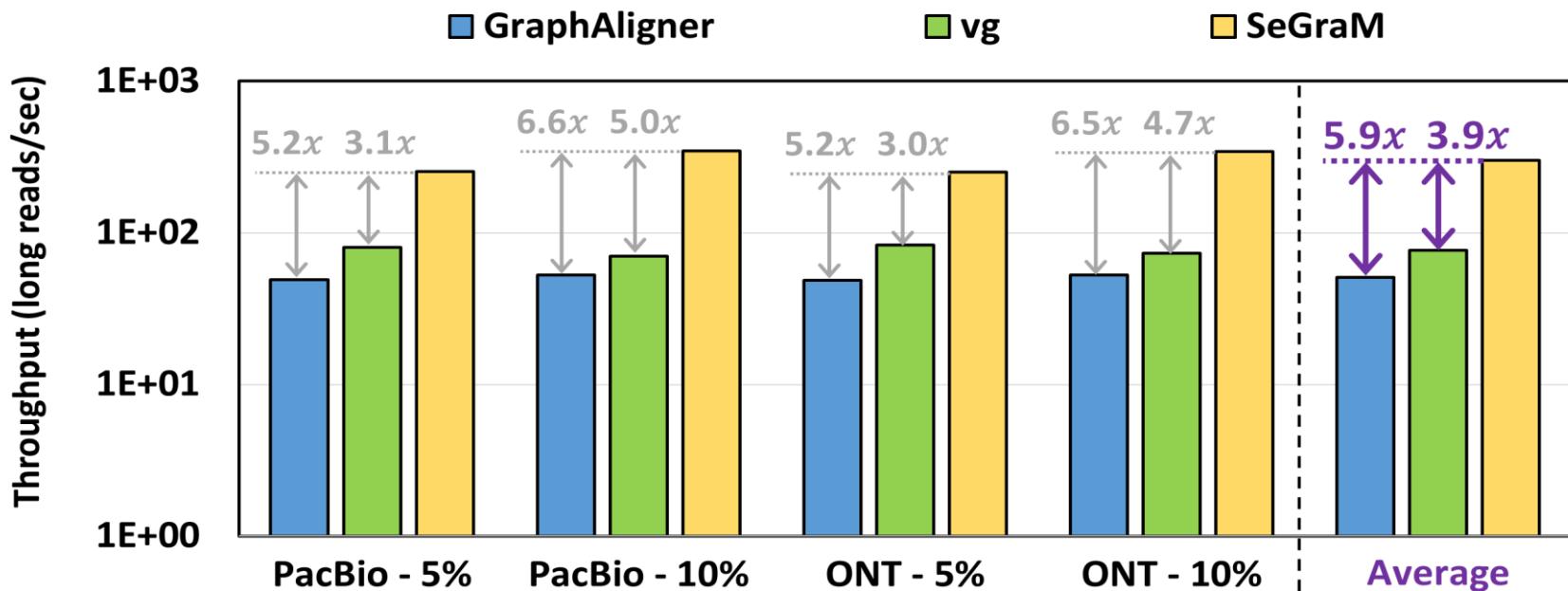
- **Performance, Area and Power Analysis:**
 - **Synthesized SystemVerilog models** of the MinSeed and BitAlign accelerator datapaths
 - **Simulation- and spreadsheet-based** performance modeling
- **Baseline Comparison Points:**
 - **GraphAligner, vg, and HGA** for sequence-to-graph mapping
 - **PaSGAL** for sequence-to-graph alignment
 - **Darwin, GenAx, and GenASM** for sequence-to-sequence alignment
- **Datasets:**
 - **Graph-based reference:** GRCh38 + 7 VCF files for HG001-007
 - **Simulated datasets** for both short and long reads

Key Results – Area & Power

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

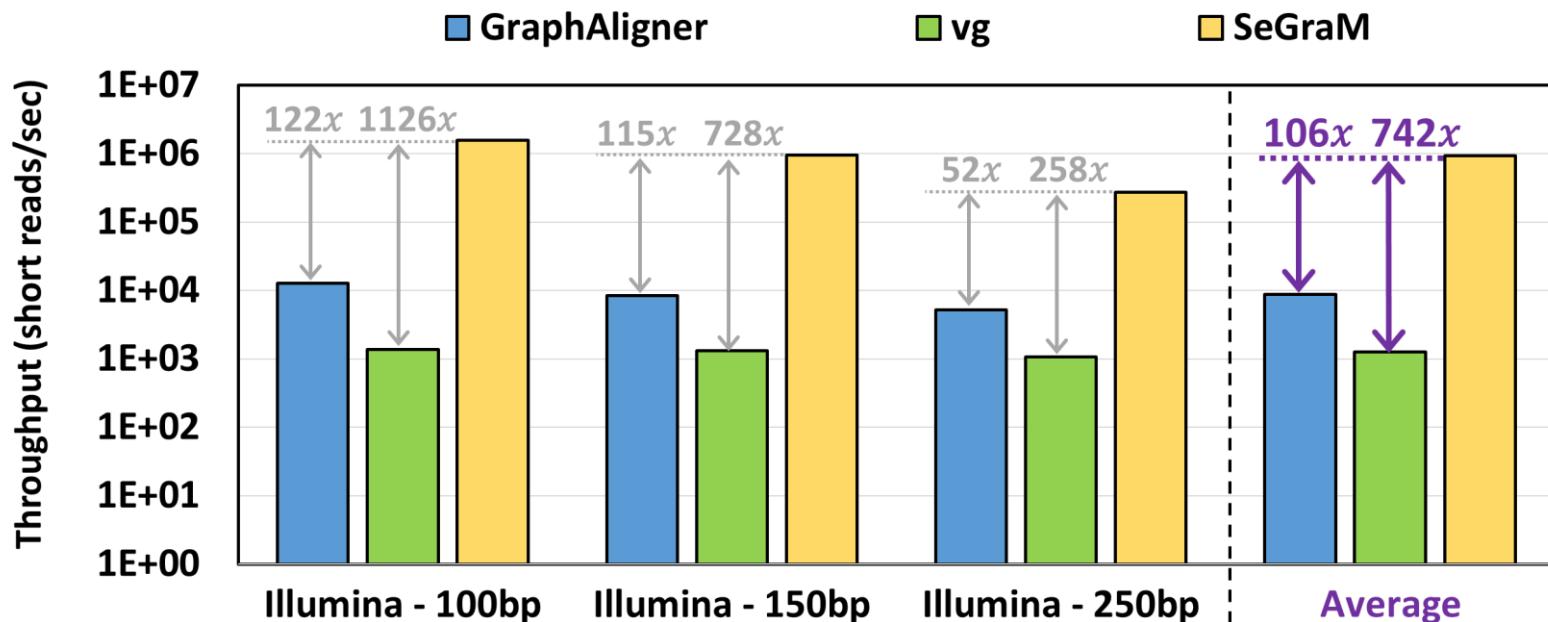
Component	Area (mm ²)	Power (mW)
MinSeed – Logic	0.017	10.8
Read Scratchpad (6 kB)	0.012	7.9
Minimizer Scratchpad (40 kB)	0.055	22.7
Seed Scratchpad (4 kB)	0.008	6.4
BitAlign – Edit Distance Calculation Logic with Hop Queue Registers (64 PEs)	0.393	378.0
BitAlign – Traceback Logic	0.020	2.7
Input Scratchpad (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



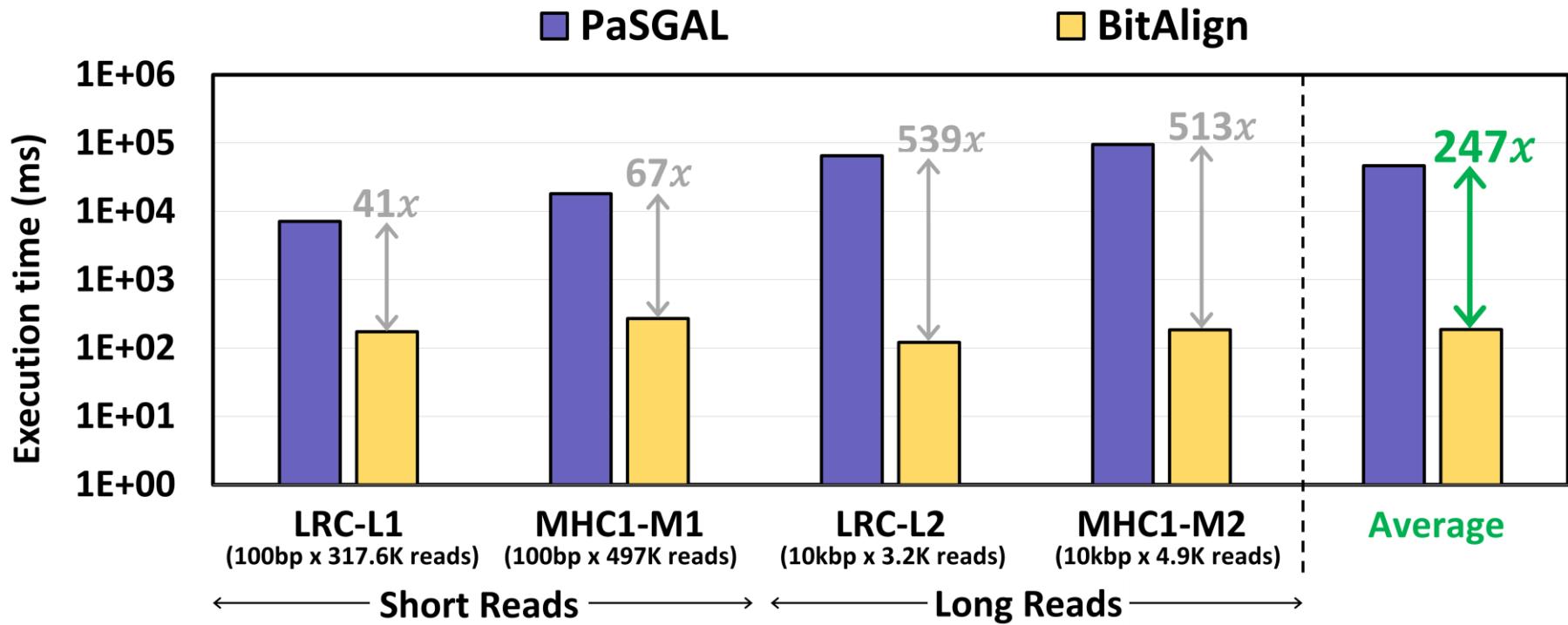
SeGraM provides **5.9x and 3.9x throughput improvement**
over GraphAligner and vg,
while **reducing the power consumption by 4.1x and 4.4x**

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 **41x-539x speedup** over PaSGAL

Key Results – BitAlign (S2S Alignment)

- BitAlign can also be used for sequence-to-sequence alignment
 - The cost of more functionality: **extra hop queue registers**
 - We do *not* sacrifice any performance
- **For long reads (over GACT of Darwin and GenASM):**
 - **4.8× and 1.2×** throughput improvement,
 - **2.7× and 7.5×** higher power consumption, and
 - **1.5× and 2.6×** higher area overhead
- **For short reads (over SillaX of GenAx and GenASM):**
 - **2.4× and 1.3×** throughput improvement

Outline

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

Additional Details in the Paper

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

Conclusion

- **SeGraM:** *First universal algorithm/hardware co-designed genomic mapping accelerator that supports:*
 - Sequence-to-graph (S₂G) & sequence-to-sequence (S₂S) mapping
 - Short & long reads
- **MinSeed:** *First minimizer-based seeding accelerator*
- **BitAlign:** *First (bitvector-based) S₂G alignment accelerator*
- **SeGraM supports multiple use cases:**
 - End-to-end S₂G mapping
 - S₂G alignment
 - S₂S alignment
 - Seeding
- **SeGraM outperforms state-of-the-art software & hardware solutions**

SeGram [ISCA 2022]

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

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

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

¹Bionano Genomics ²ETH Zürich ³Bilkent University ⁴Intel Labs

⁵Carnegie Mellon University ⁶University of Illinois Urbana-Champaign

SeGram – GitHub Page

<https://github.com/CMU-SAFARI/SeGram>

CMU-SAFARI / SeGram Public

Watch 6 Fork 0 Star 5

Code Issues Pull requests Actions Projects Wiki Security Insights

main 1 branch 0 tags Go to file Add file Code

damlasenolcali Update README.md 45d20dc 8 days ago 26 commits

src BitAlign source files 8 days ago

LICENSE Initial commit 7 months ago

README.md Update README.md 8 days ago

README.md

SeGram: A Universal Genomic Mapping Accelerator for both Sequence-to-Graph Mapping and Sequence-to-Sequence Mapping

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

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

Readme MIT license 5 stars 6 watching 0 forks

Releases

SeGraM

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

Damla Senol Cali, Konstantinos Kanellopoulos, Joël 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

intel®

ETH zürich

UNIVERSITY OF
ILLINOIS
URBANA-CHAMPAIGN



Bilkent University

SAFARI

December 12 2023

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)

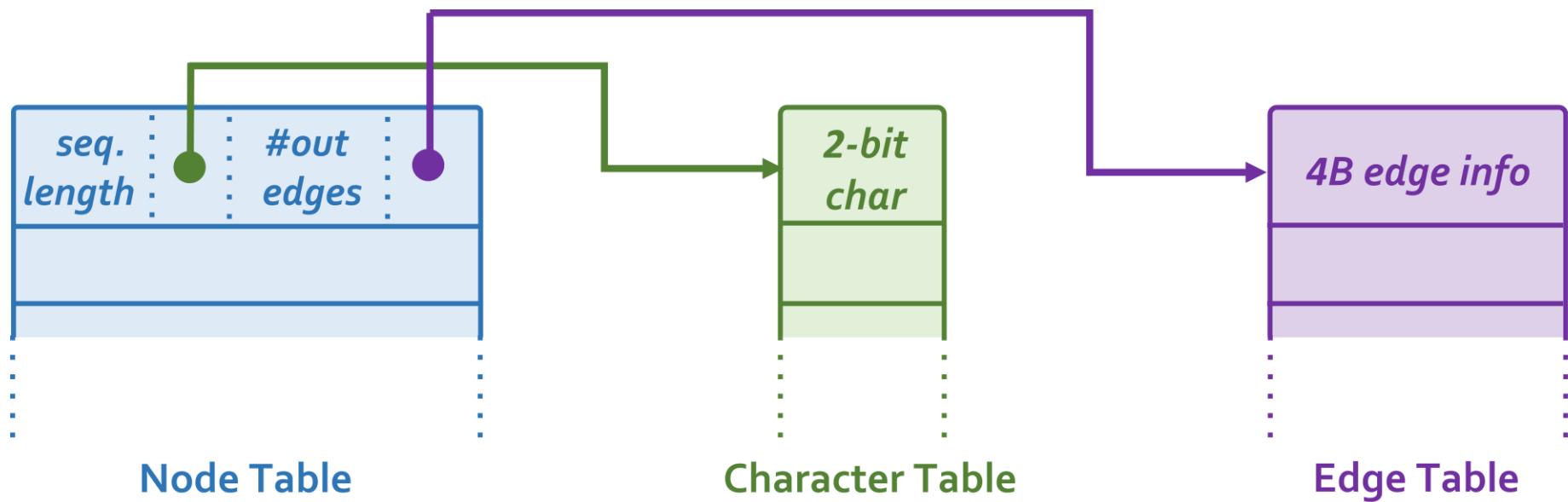
Sequence-to-Sequence (S2S) Mapping

- 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

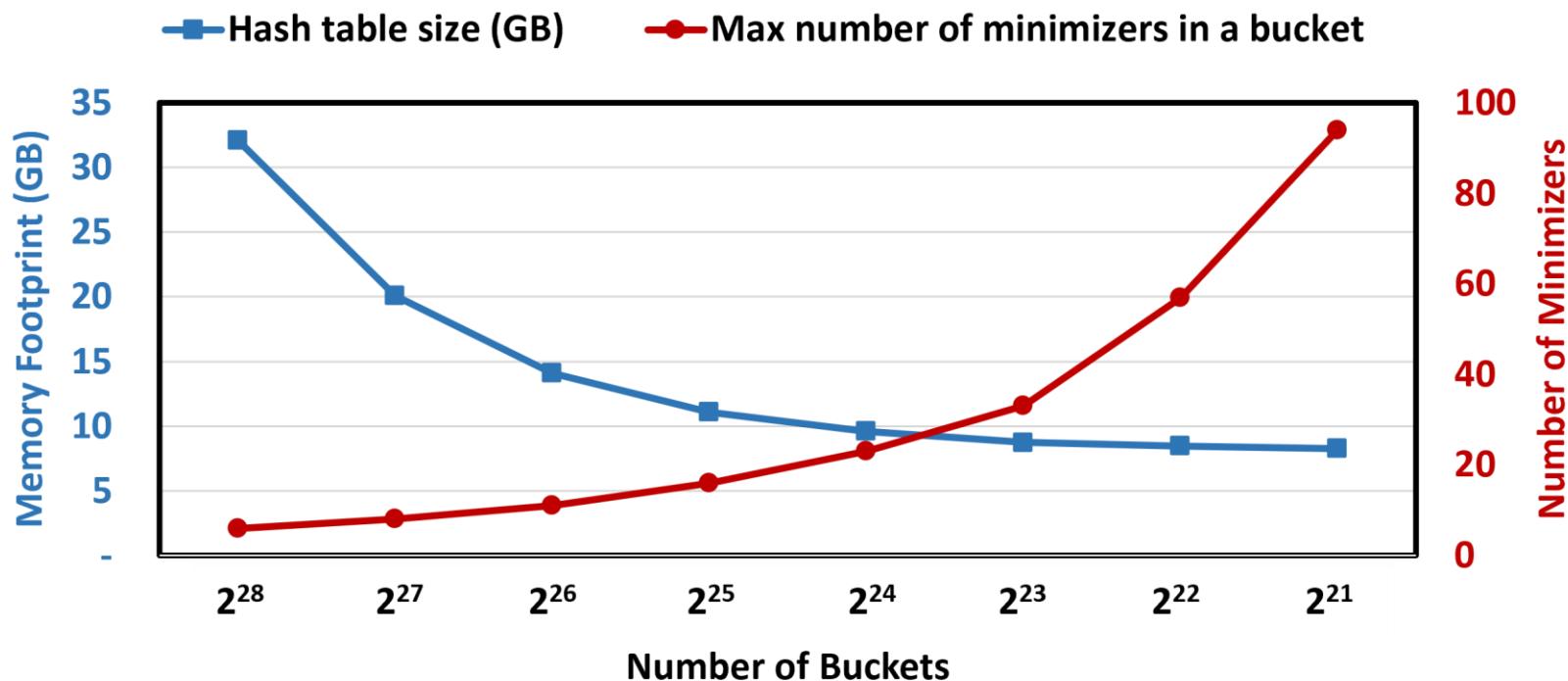
SeGraM – Graph Structure



SeGraM – Index Structure



SeGraM – Selection of #Buckets

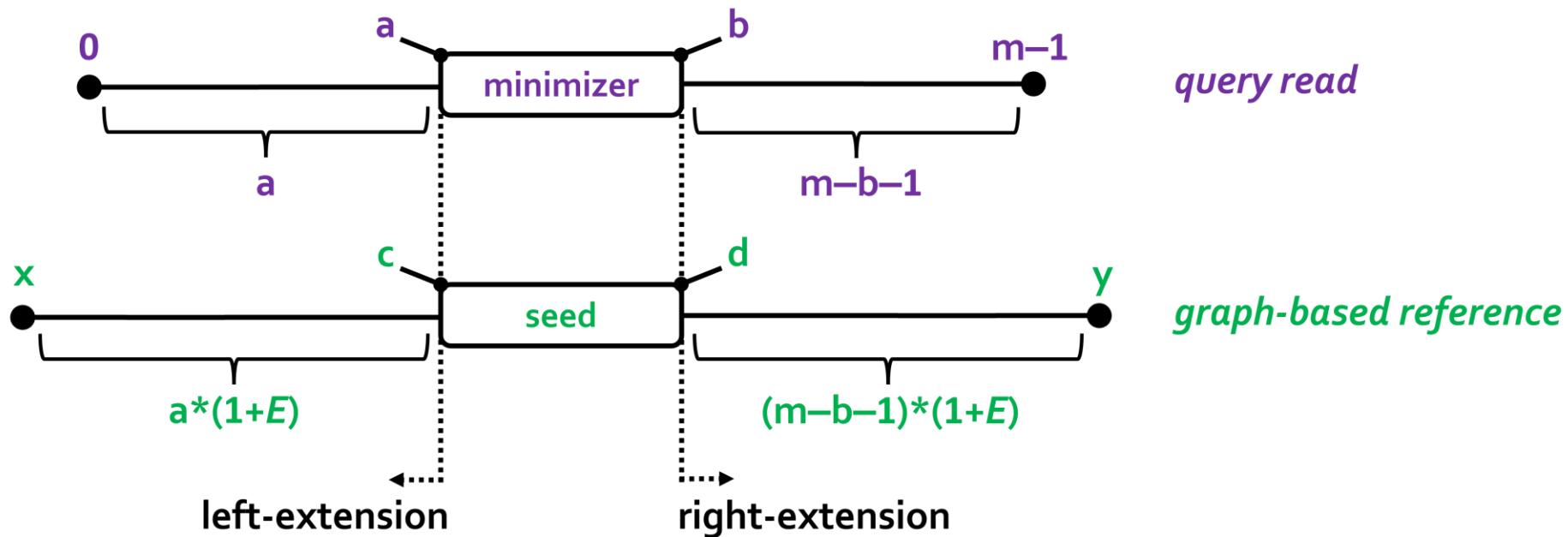


Minimizers

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

lexicographically
smallest k -mer

MinSeed – Region Calculation



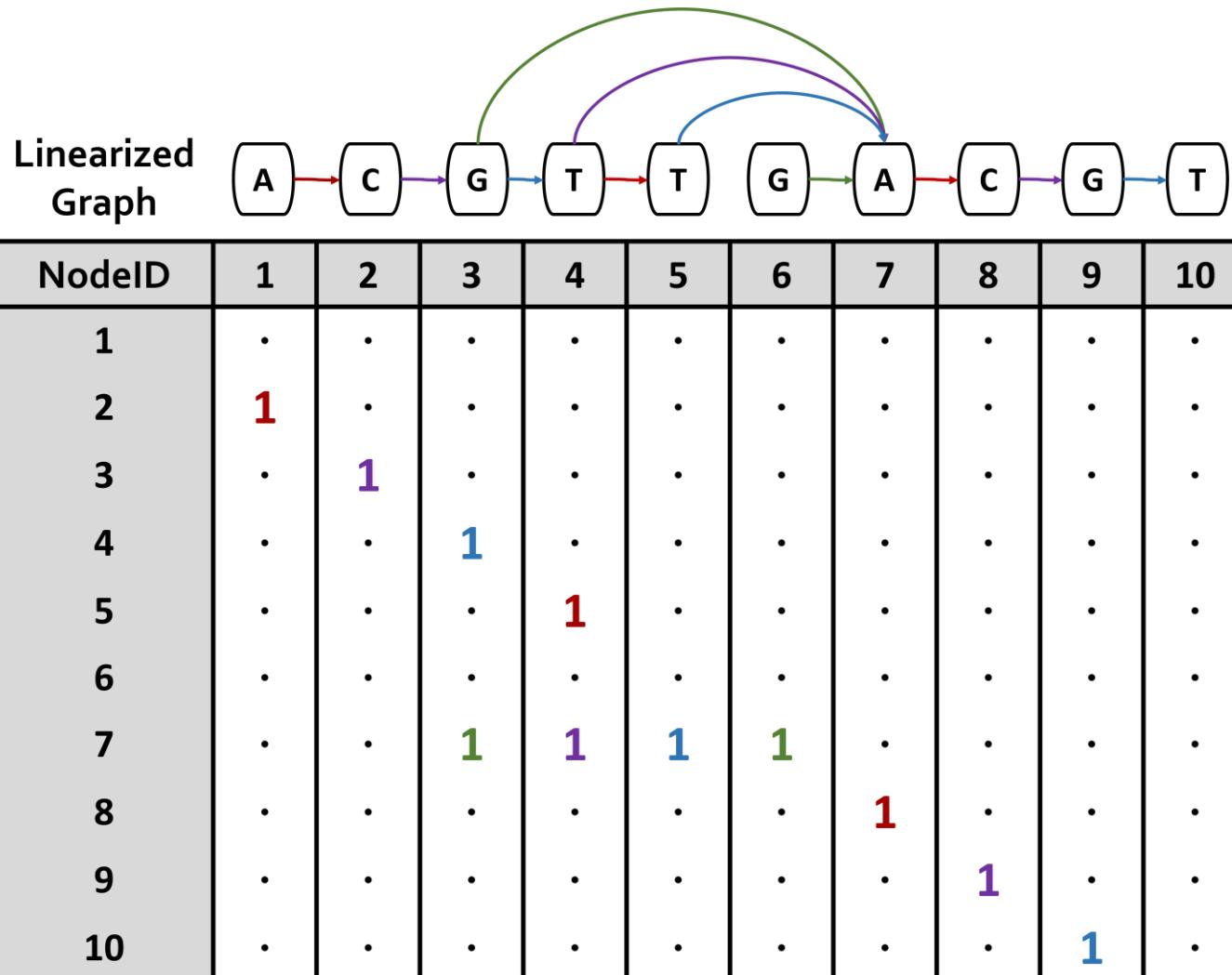
BitAlign Algorithm

Algorithm 1 BitAlign Algorithm

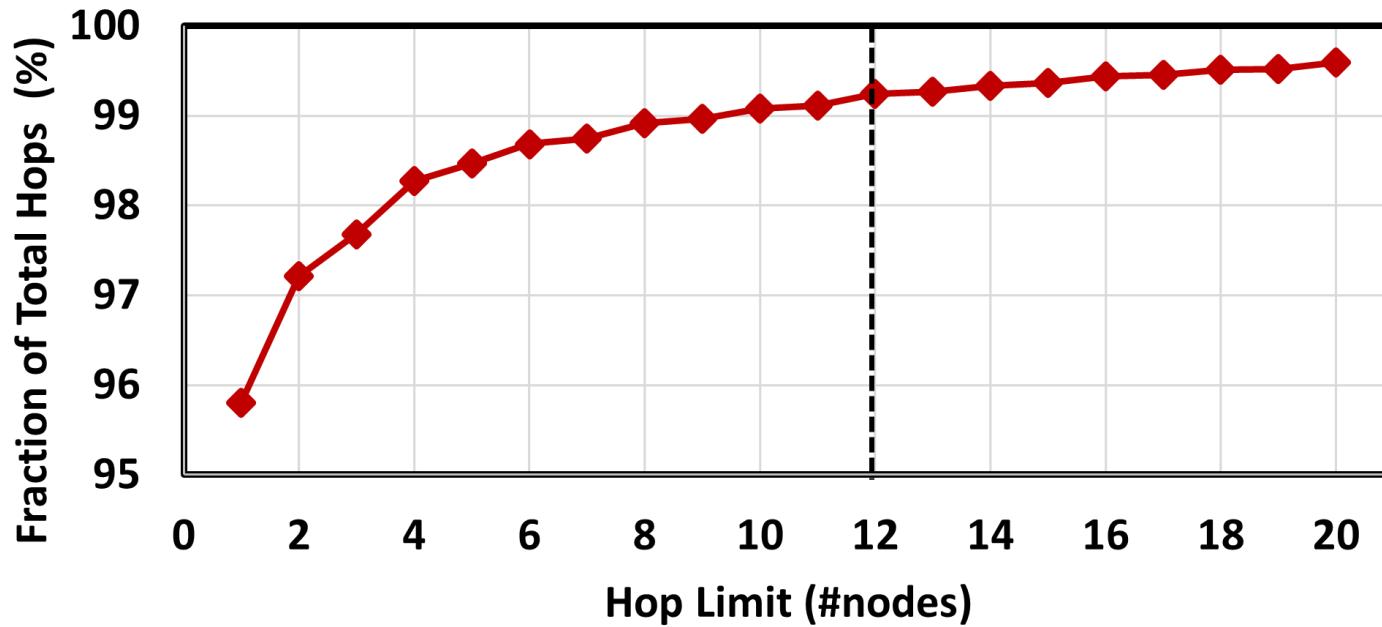
```
Inputs: linearized and topologically sorted subgraph (reference),
       query-read (pattern), k (edit distance threshold)
Outputs: editDist (minimum edit distance), CIGARstr (traceback output)

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

BitAlign – HopBits



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: AAA T G T T T A T G C T A C T G
Read: AAA T G T T T A T G C T A C 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

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

For $d = 1 \dots k$:

deletion = $\text{oldR}[d-1]$

substitution = $\text{oldR}[d-1] \ll 1$

insertion = $R[d-1] \ll 1$

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

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

Check MSB of $R[d]$:

If 1, no match.

If 0, match with d many errors.

Large number of iterations

Bitap Algorithm (cont'd.)

□ Step 2: Edit Distance Calculation

For each character of the text (char):

Copy previous R bitvectors as oldR

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

For $d = 1 \dots k$:

deletion $= \text{oldR}[d-1]$

substitution $= \text{oldR}[d-1] \ll 1$

insertion $= R[d-1] \ll 1$

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

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

Check MSB of $R[d]$:

If 1, no match.

If 0, match with d many errors.

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

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 $d = 1 \dots k$:

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

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

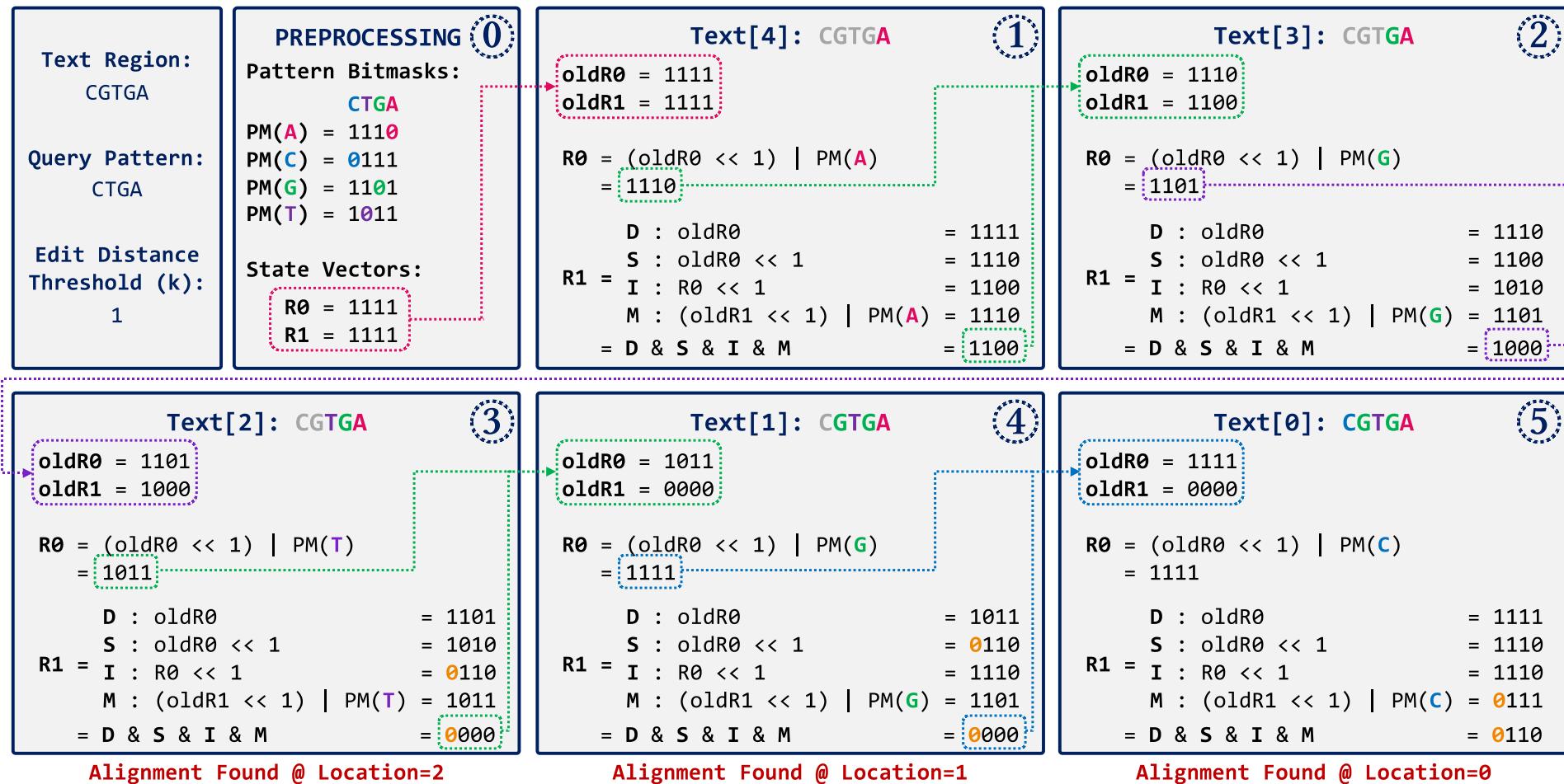
$R[d] = \text{deletion} \& \text{mismatch} \& \text{insertion} \& \text{match}$

Check MSB of $R[d]$:

If 1, no match.

If 0, match with d many errors.

Example for the Bitap Algorithm



Limitations of Bitap

1) Data Dependency Between Iterations: Algorithm

- 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: Hardware

- Text-level parallelism
- Limited by the number of compute units in existing systems

5) Limited Memory Bandwidth:

- High memory bandwidth required to read and write the computed bitvectors to memory

GenASM: ASM Framework for GSA

Our Goal:

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

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

GenASM Algorithm

□ GenASM-DC Algorithm:

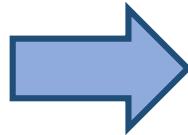
- Modified Bitap for Distance Calculation
- Extended for efficient long read support
- Besides bit-parallelism that Bitap has, extended for parallelism:
 - Loop unrolling
 - Text-level parallelism

□ GenASM-TB Algorithm:

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

Loop Unrolling in GenASM-DC

Cycle#	Thread ₁ Ro/1/2/..
#1	To-Ro
...	...
#8	To-R ₇
#9	T ₁ -Ro
...	...
#16	T ₁ -R ₇
#17	T ₂ -Ro
...	...
#24	T ₂ -R ₇
#25	T ₃ -Ro
...	...
#32	T ₃ -R ₇



Cycle#	Thread ₁ Ro/4	Thread ₂ R ₁ /5	Thread ₃ R ₂ /6	Thread ₄ R ₃ /7
#1	To-Ro	-	-	-
#2	T ₁ -Ro	To-R ₁	-	-
#3	T ₂ -Ro	T ₁ -R ₁	To-R ₂	-
#4	T ₃ -Ro	T ₂ -R ₁	T ₁ -R ₂	To-R ₃
#5	To-R ₄	T ₃ -R ₁	T ₂ -R ₂	T ₁ -R ₃
#6	T ₁ -R ₄	To-R ₅	T ₃ -R ₂	T ₂ -R ₃
#7	T ₂ -R ₄	T ₁ -R ₅	To-R ₆	T ₃ -R ₃
#8	T ₃ -R ₄	T ₂ -R ₅	T ₁ -R ₆	To-R ₇
#9	-	T ₃ -R ₅	T ₂ -R ₆	T ₁ -R ₇
#10	-	-	T ₃ -R ₆	T ₂ -R ₇
#11	-	-	-	T ₃ -R ₇



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	Text[1]: G	Text[2]: T	Text[3]: G	Text[4]: A
$\begin{cases} R0- : \dots \\ R1-M : 0111 \end{cases}$	$\begin{cases} R0- : \dots \\ R1-D : 1011 \end{cases}$	$\begin{cases} R0-M : 1011 \\ R1- : \dots \end{cases}$	$\begin{cases} R0-M : 1101 \\ R1- : \dots \end{cases}$	$\begin{cases} R0-M : 1110 \\ R1- : \dots \end{cases}$
Match(C) <3,0,1>	Del(-) <2,1,1>	Match(T) <2,2,0>	Match(G) <1,3,0>	Match(A) <0,4,0>

Substitution Example (Text Location=1)

(b)

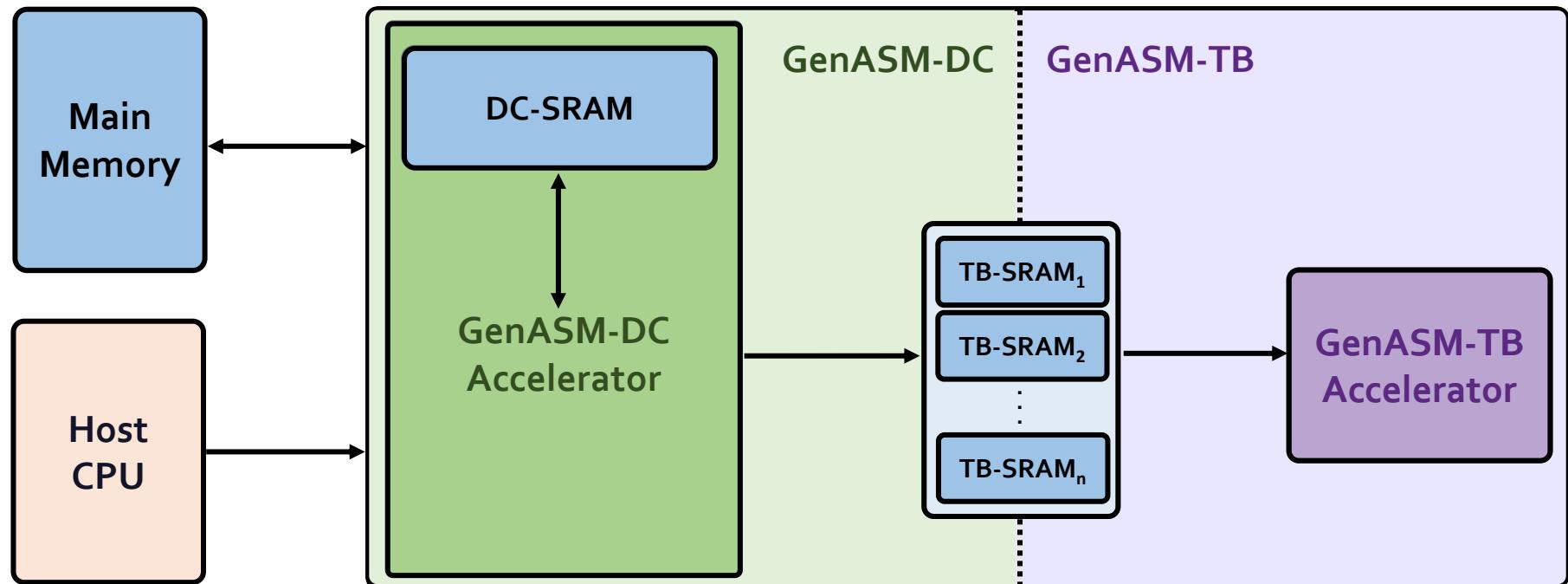
Text[1]: G	Text[2]: T	Text[3]: G	Text[4]: A
$\begin{cases} R0- : \dots \\ R1-S : 0110 \end{cases}$	$\begin{cases} R0-M : 1011 \\ R1- : \dots \end{cases}$	$\begin{cases} R0-M : 1101 \\ R1- : \dots \end{cases}$	$\begin{cases} R0-M : 1110 \\ R1- : \dots \end{cases}$
Subs(C) <3,1,1>	Match(T) <2,2,0>	Match(G) <1,3,0>	Match(A) <0,4,0>

Insertion Example (Text Location=2)

(c)

Text[-]	Text[2]: T	Text[3]: G	Text[4]: A
$\begin{cases} R0- : \dots \\ R1-I : 0110 \end{cases}$	$\begin{cases} R0-M : 1011 \\ R1- : \dots \end{cases}$	$\begin{cases} R0-M : 1101 \\ R1- : \dots \end{cases}$	$\begin{cases} R0-M : 1110 \\ R1- : \dots \end{cases}$
Ins(C) <3,2,1>	Match(T) <2,2,0>	Match(G) <1,3,0>	Match(A) <0,4,0>

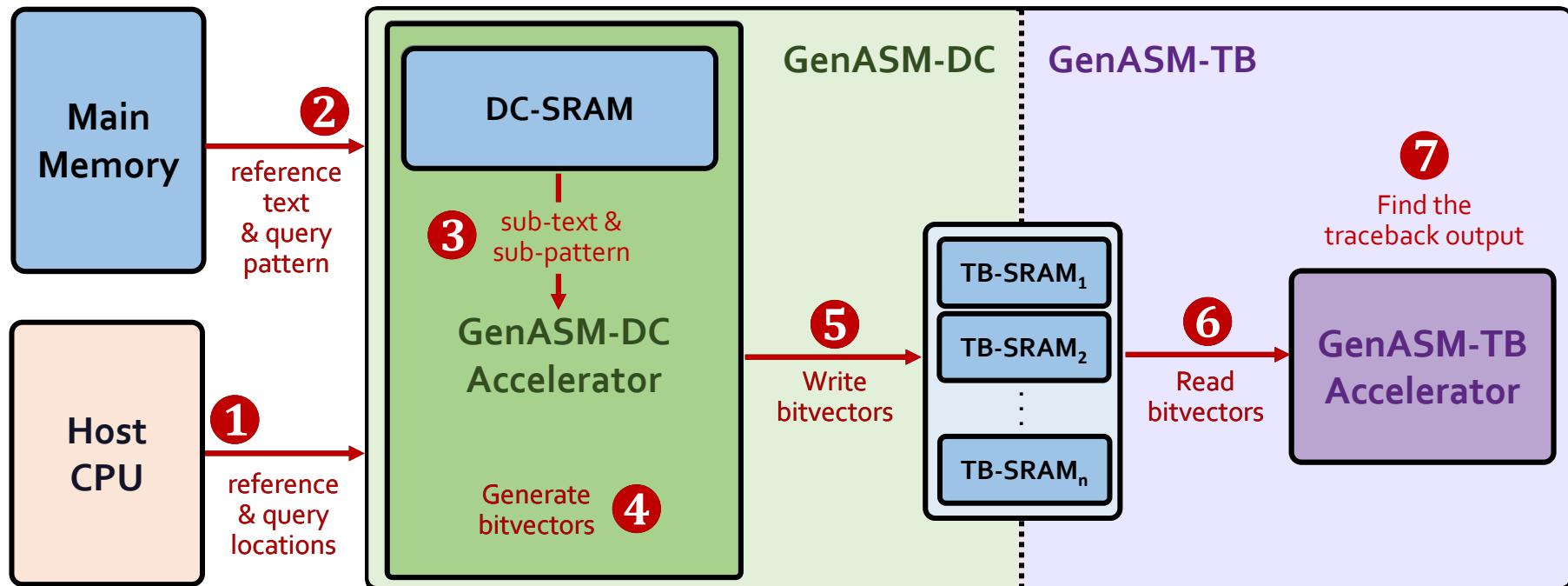
GenASM Hardware Design



GenASM-DC:
generates bitvectors
and performs edit
Distance Calculation

GenASM-TB:
performs TraceBack
and assembles the
optimal alignment

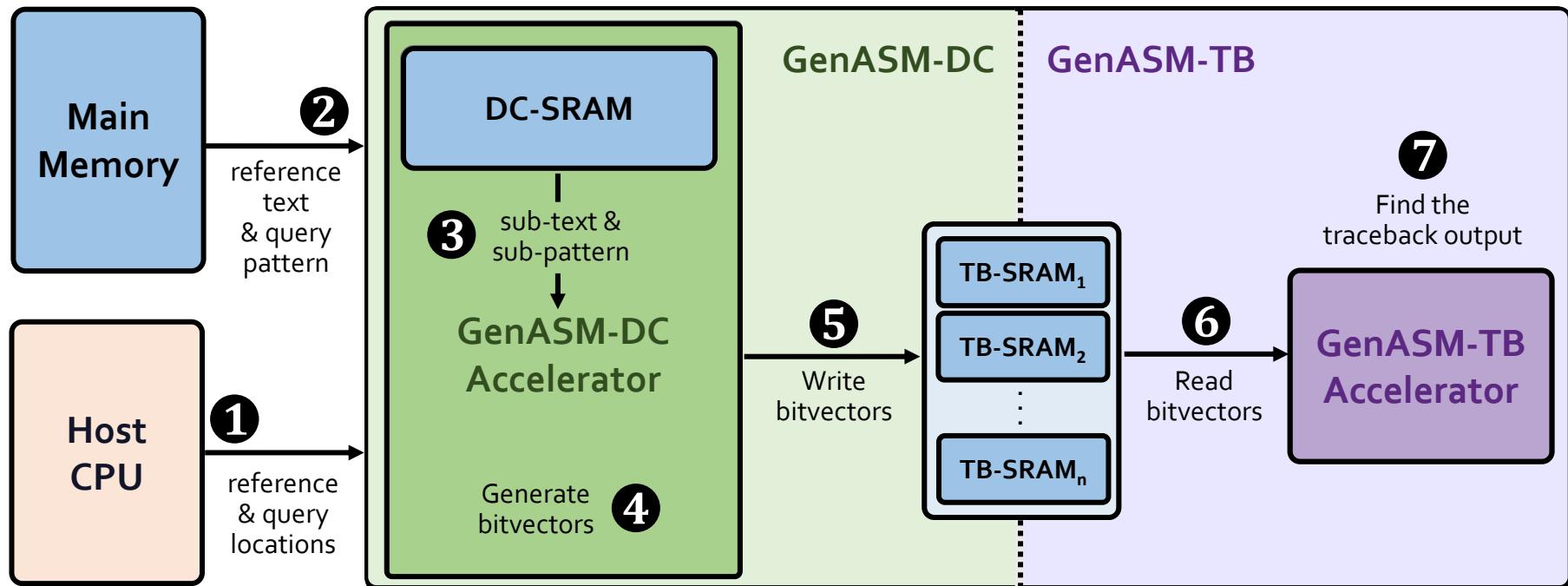
GenASM Hardware Design



GenASM-DC:
generates bitvectors
and performs edit
Distance Calculation

GenASM-TB:
performs TraceBack
and assembles the
optimal alignment

GenASM Hardware Design

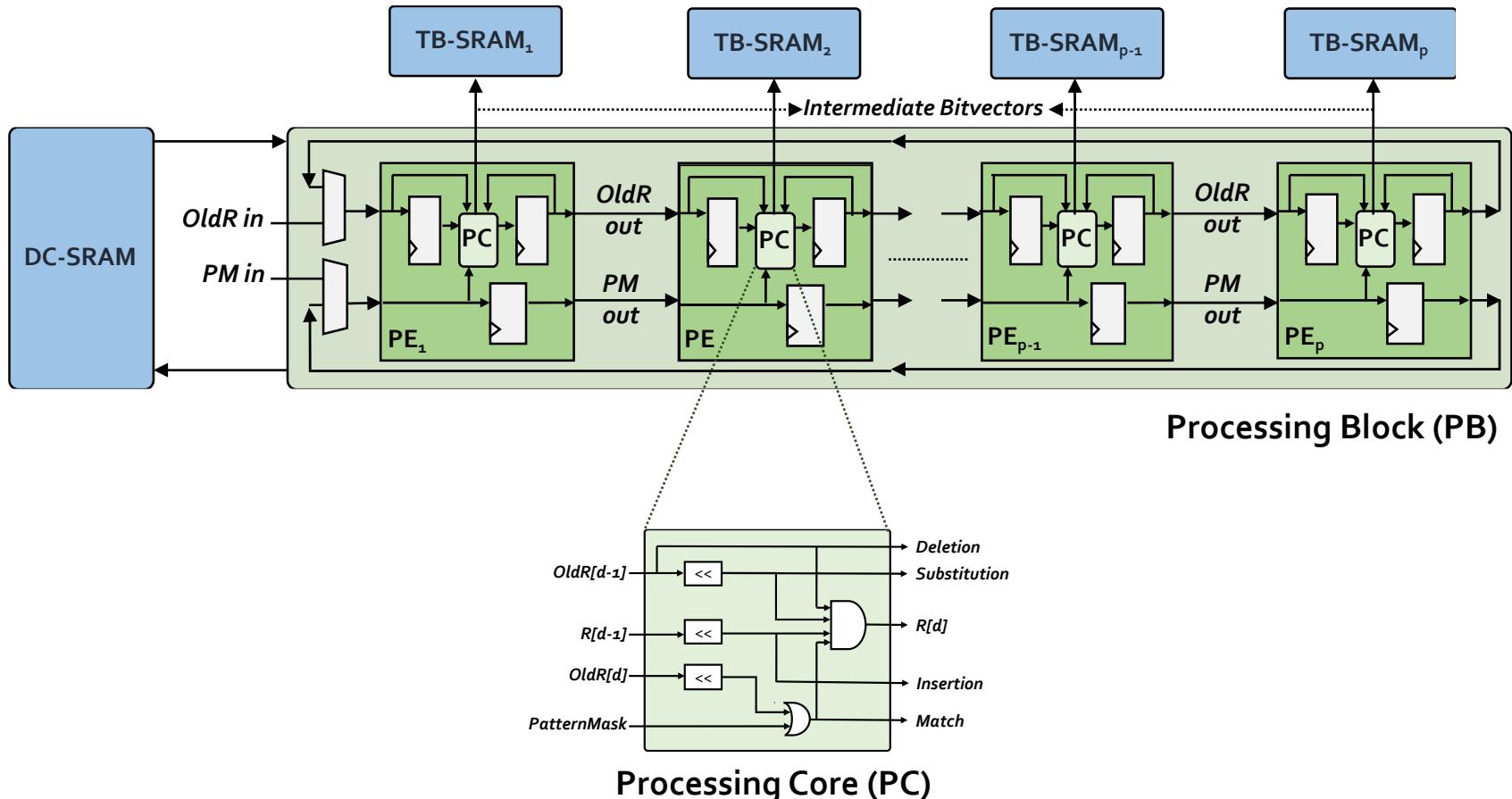


Our *specialized compute units* and *on-chip SRAMs* help us to:

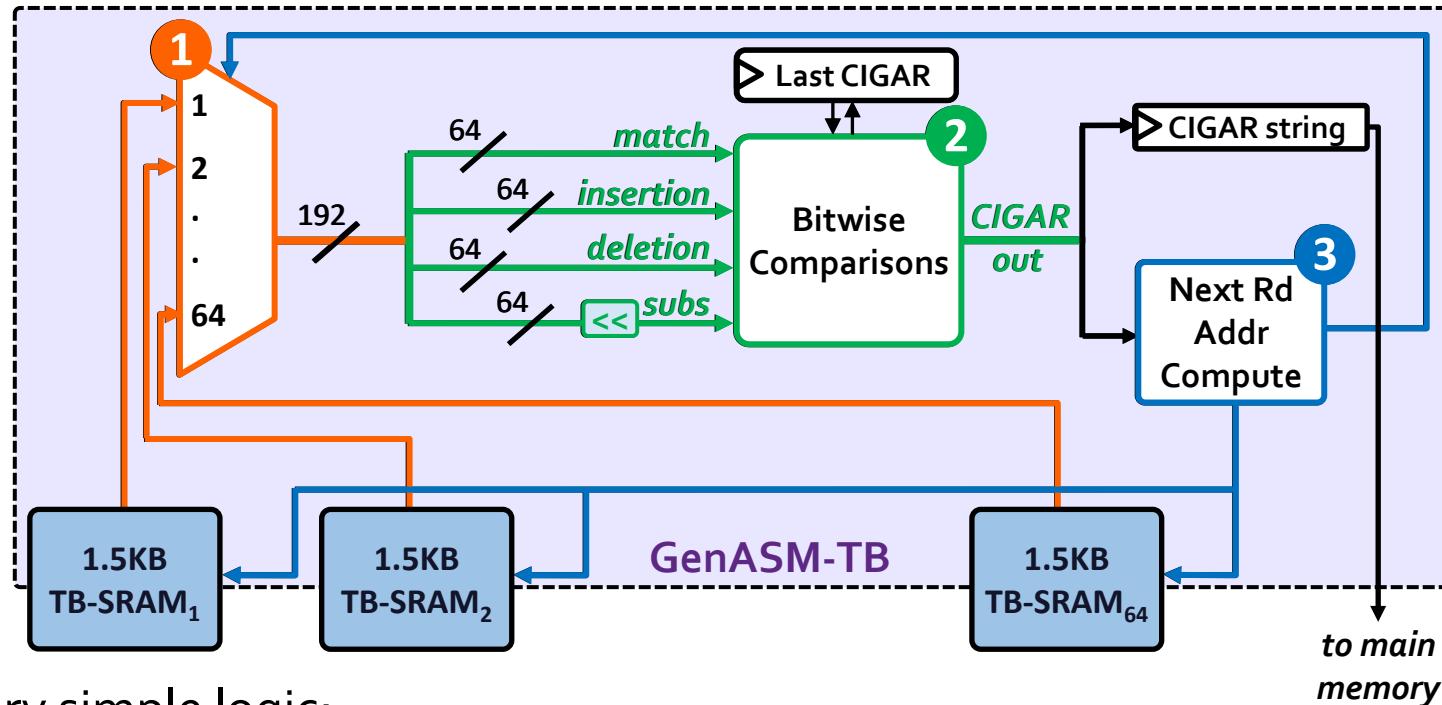
- Match **the rate of computation** with **memory capacity and bandwidth**
 - Achieve high performance and power efficiency
 - Scale linearly in performance with the number of parallel compute units that we add to the system

GenASM-DC: Hardware Design

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



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

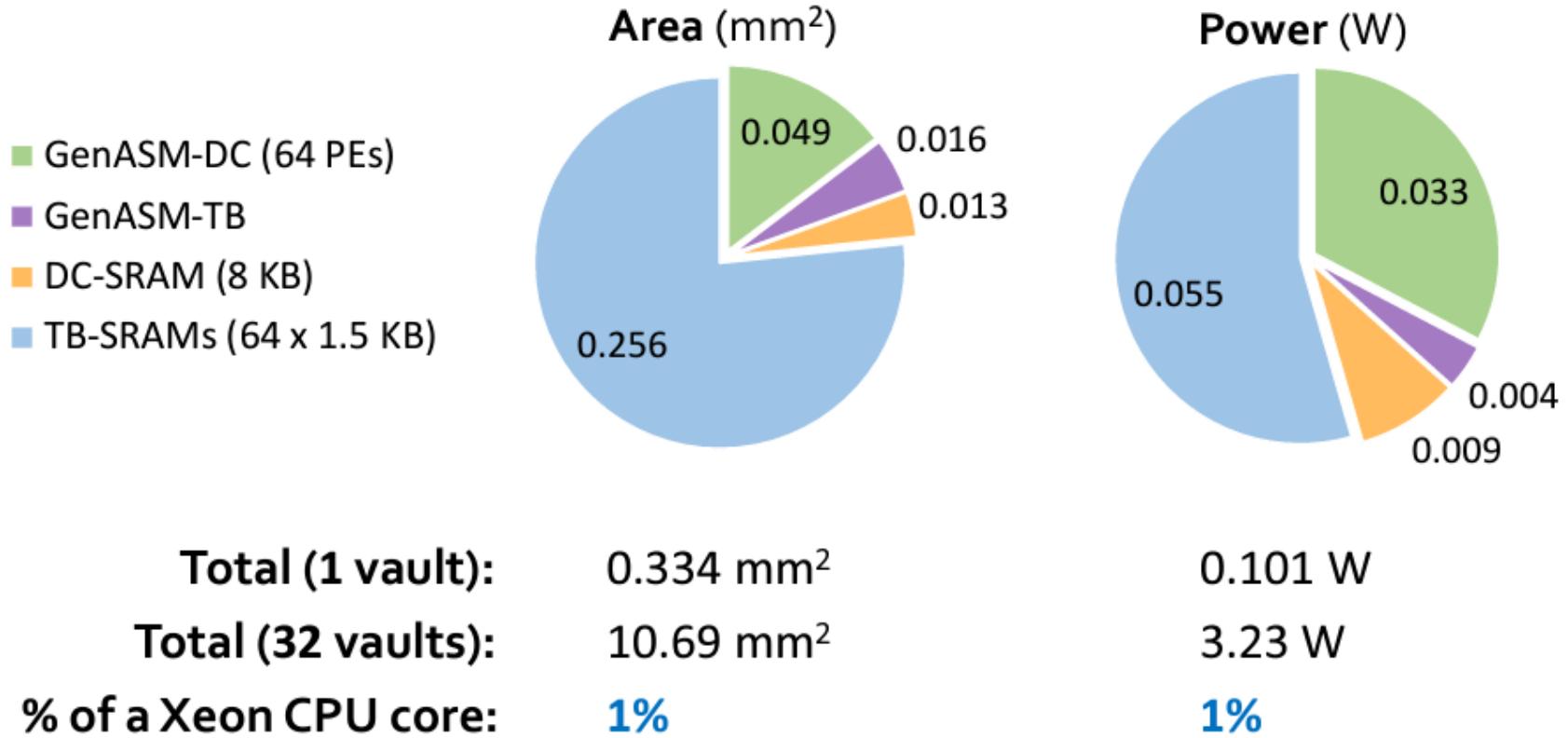
Evaluation Methodology (cont'd.)

- ❑ 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

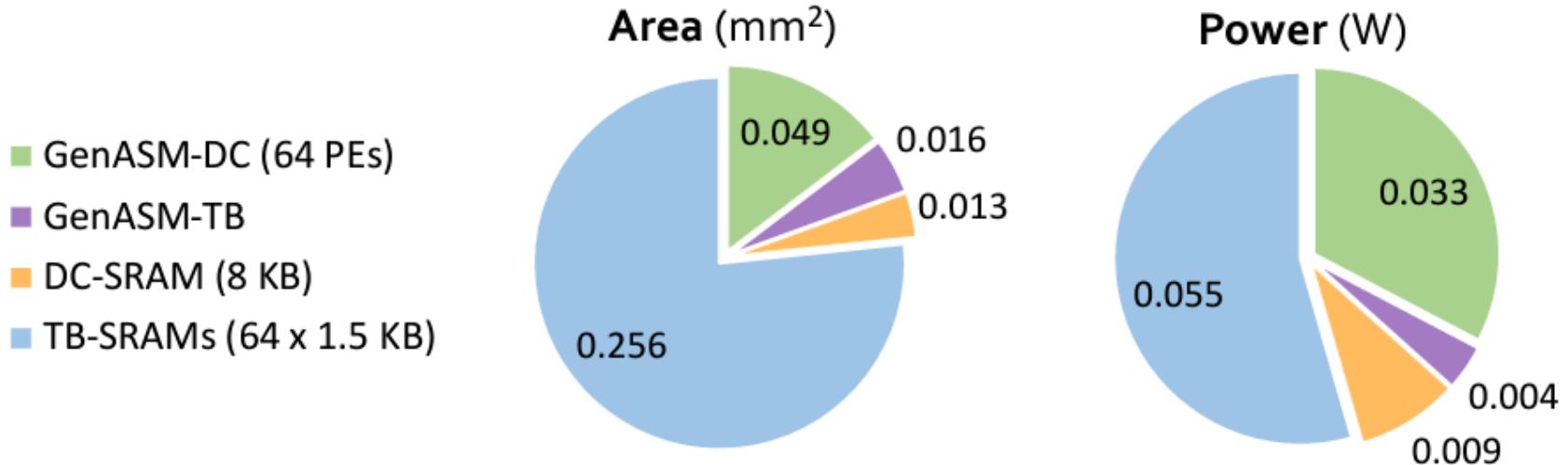
Key Results – Area and Power

- 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**



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

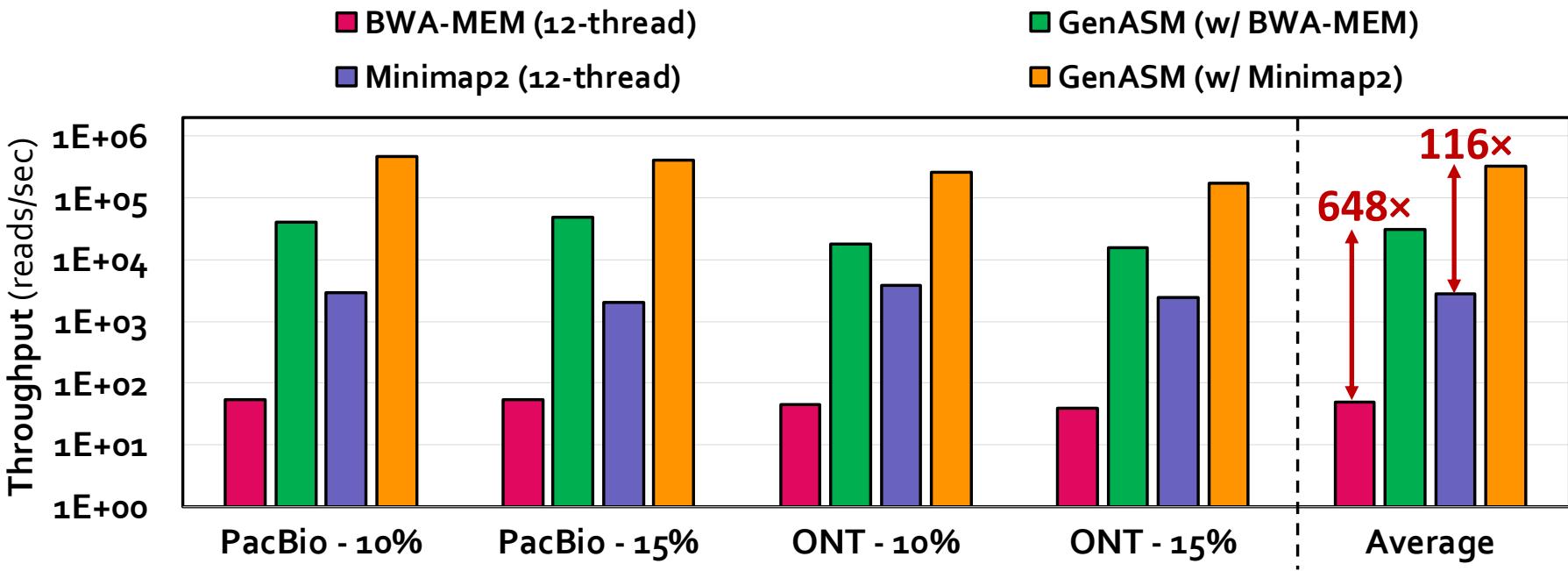
(2) Pre-Alignment Filtering for Short Reads

- 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

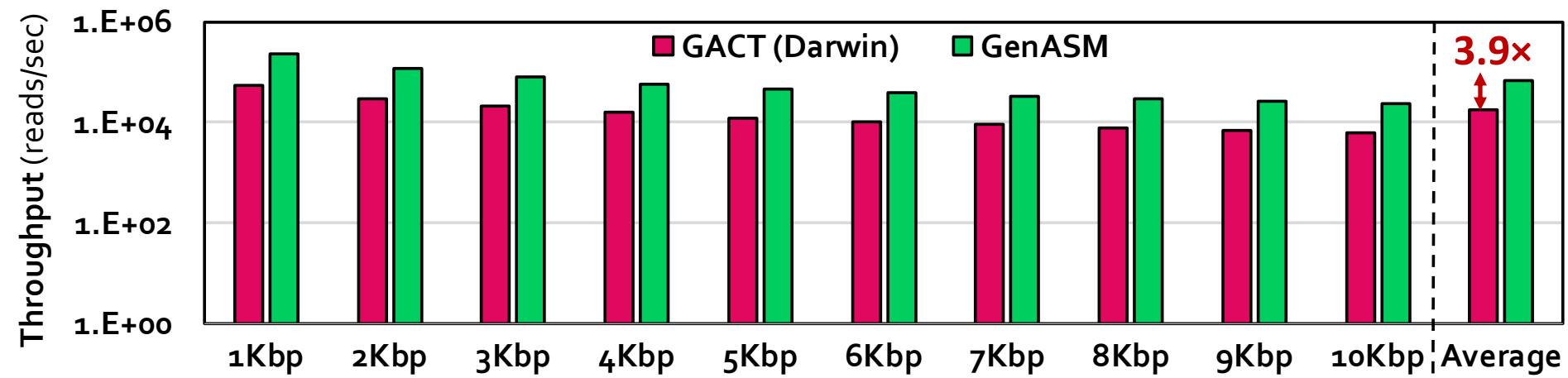
Key Results – Use Case 1 (Long Reads)



SW

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

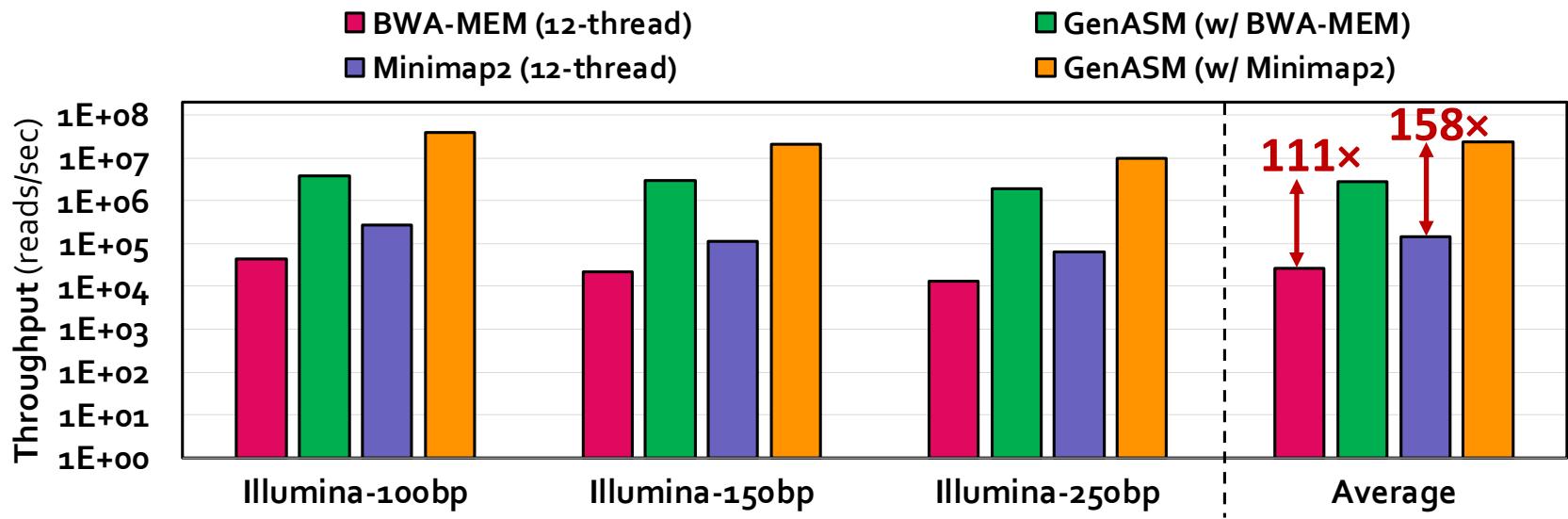
Key Results – Use Case 1 (Long Reads)



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:

- **3.7x** speedup
- **1.7x** 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

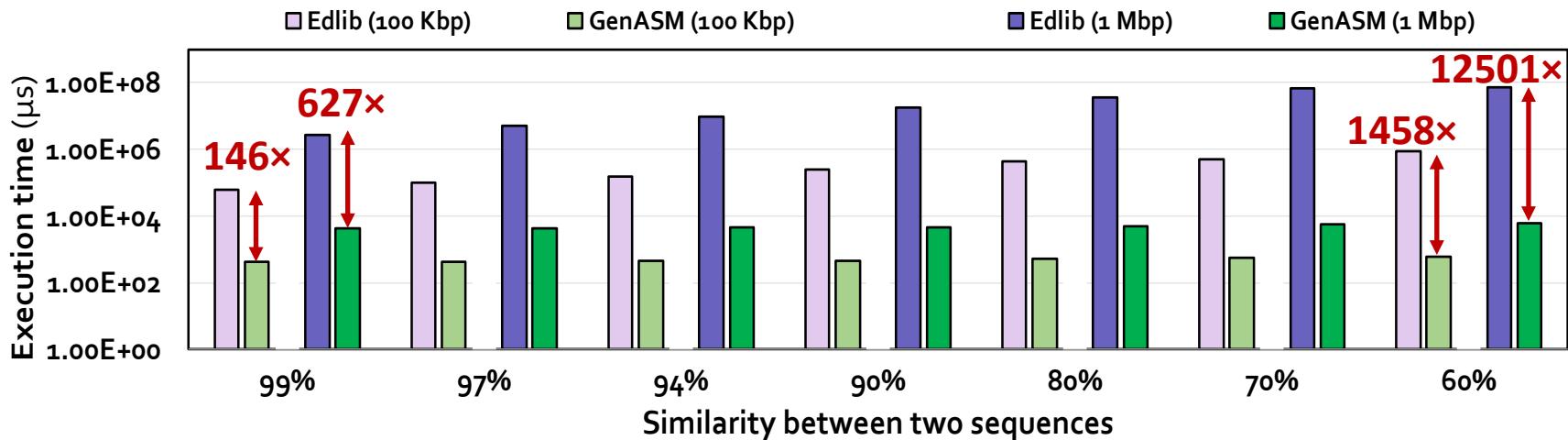
(2) Pre-Alignment Filtering for Short Reads

- 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× 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>

The screenshot shows the GitHub repository page for 'CMU-SAFARI/GenASM'. The page includes a header with repository details, a code navigation bar, a commit history table, a main content area with a title and description, and a sidebar with repository statistics.

Repository Header: CMU-SAFARI / GenASM (Public) | Edit Pins | Unwatch 5 | Fork 6 | Star 26

Code Navigation: Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, Settings

Commit History:

Author	Commit Message	Date	Commits
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

Main Content Area:

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.

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

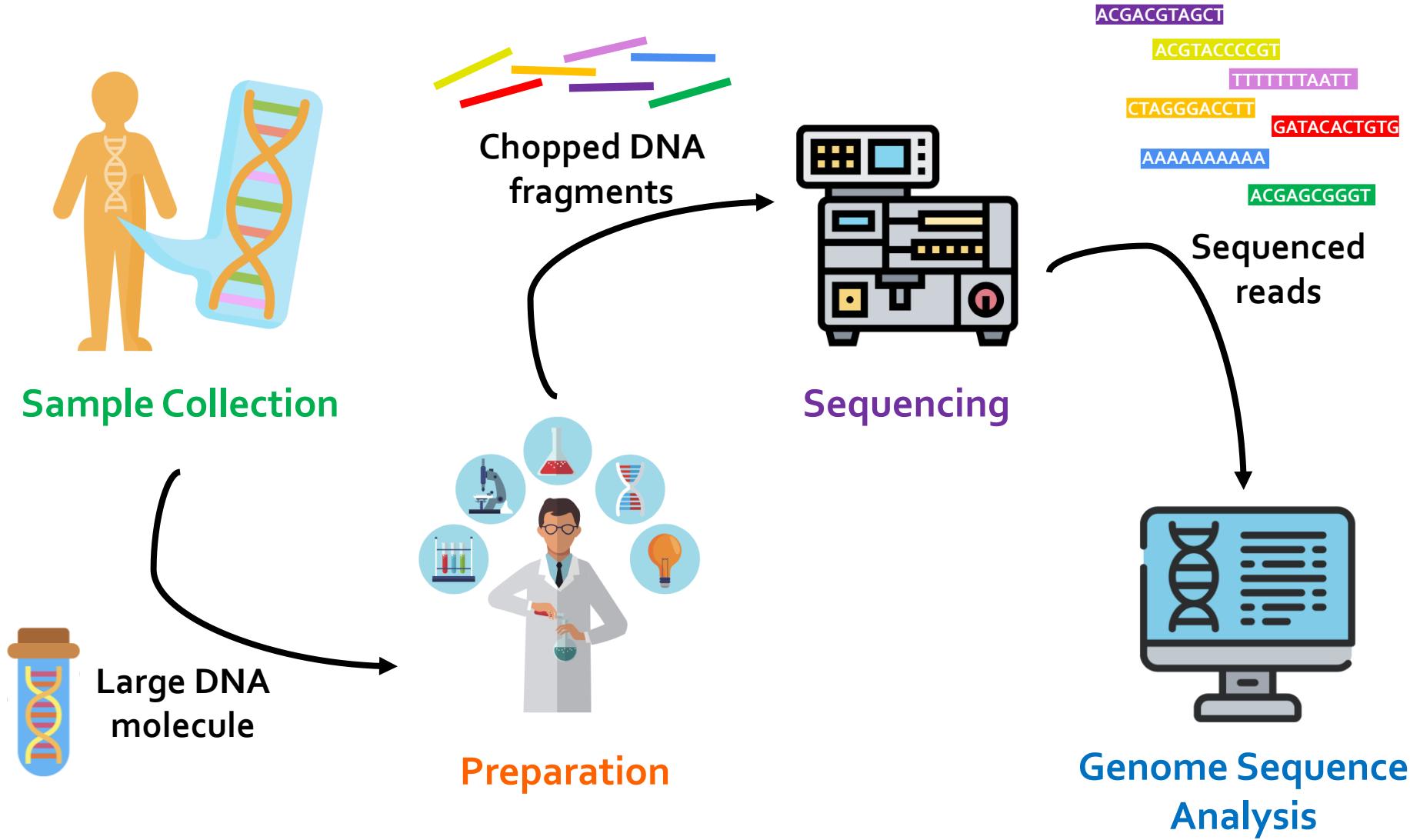
Statistics:

- Readme
- GPL-3.0 license
- 26 stars
- 5 watching
- 6 forks

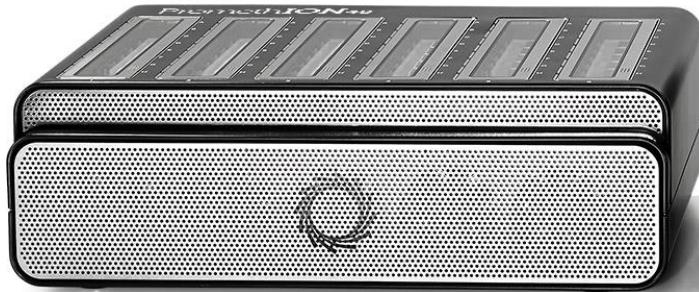
Backup Slides

(Sequencing)

Genome Sequencing



Sequencing Technologies



Oxford Nanopore
(ONT)



PacBio

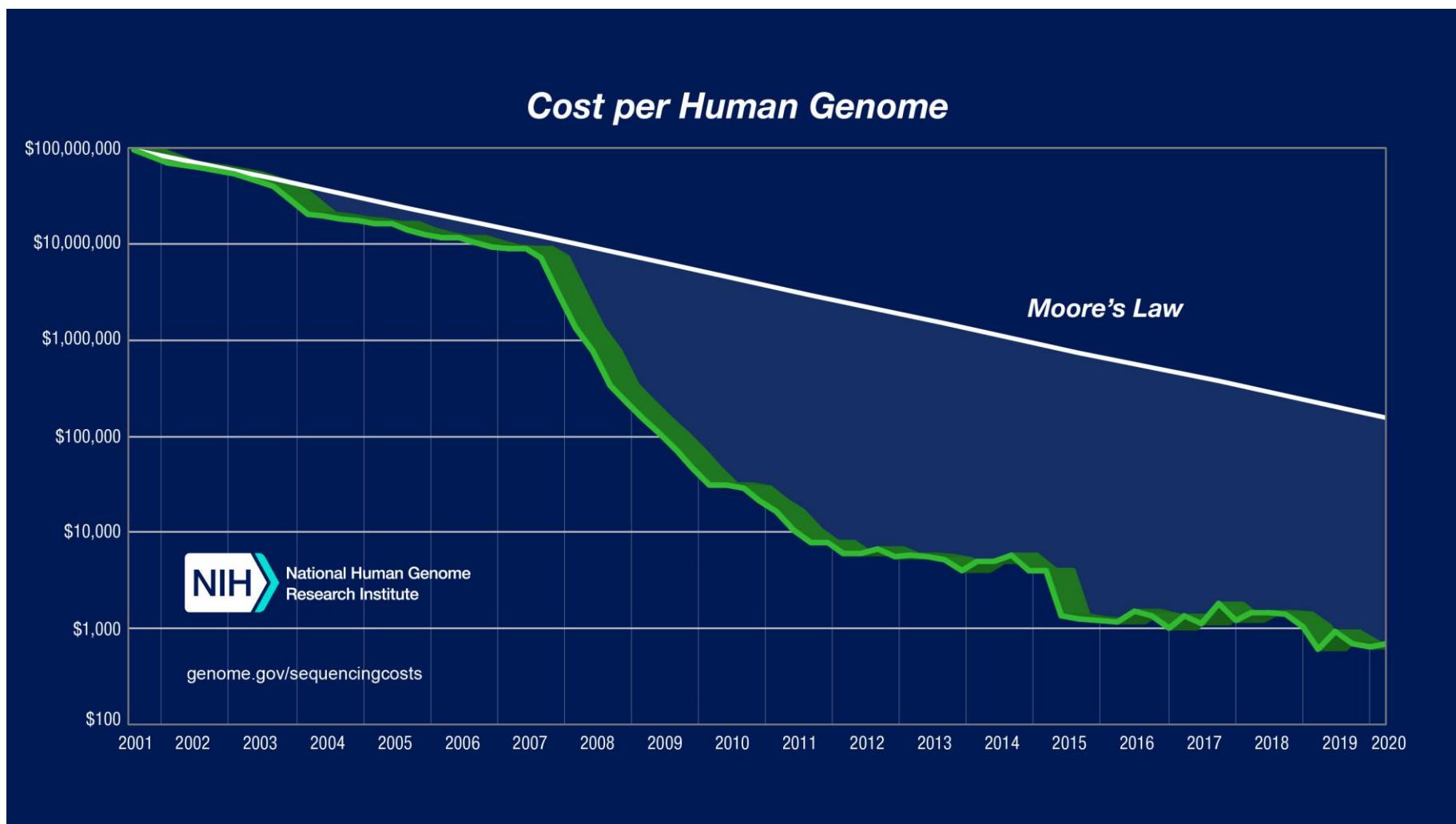
Illumina



***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.)

Cost per Human Genome

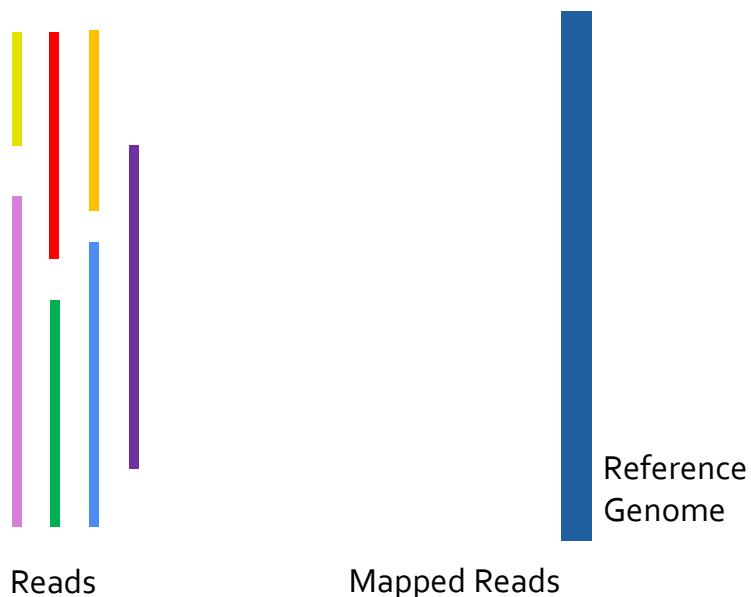


*From NIH (<https://www.genome.gov/about-genomics/fact-sheets/DNA-Sequencing-Costs-Data>)

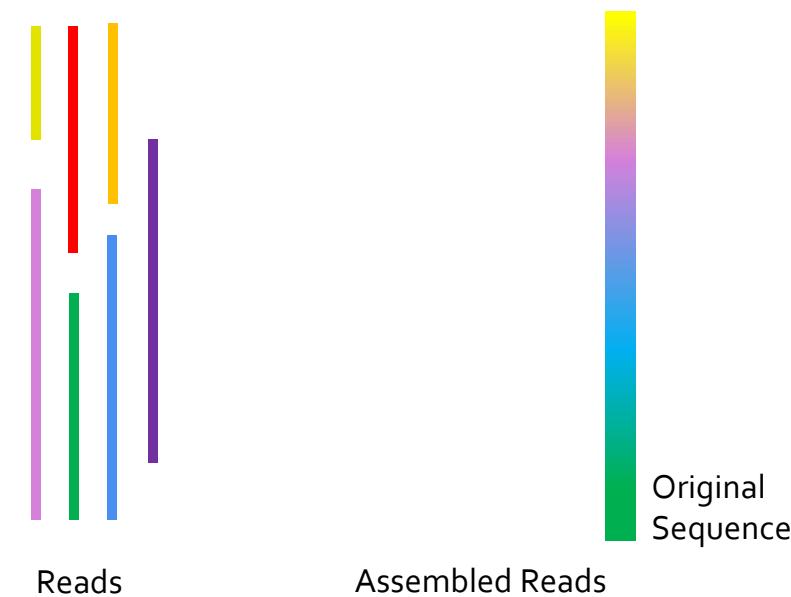
Genome Sequence Analysis



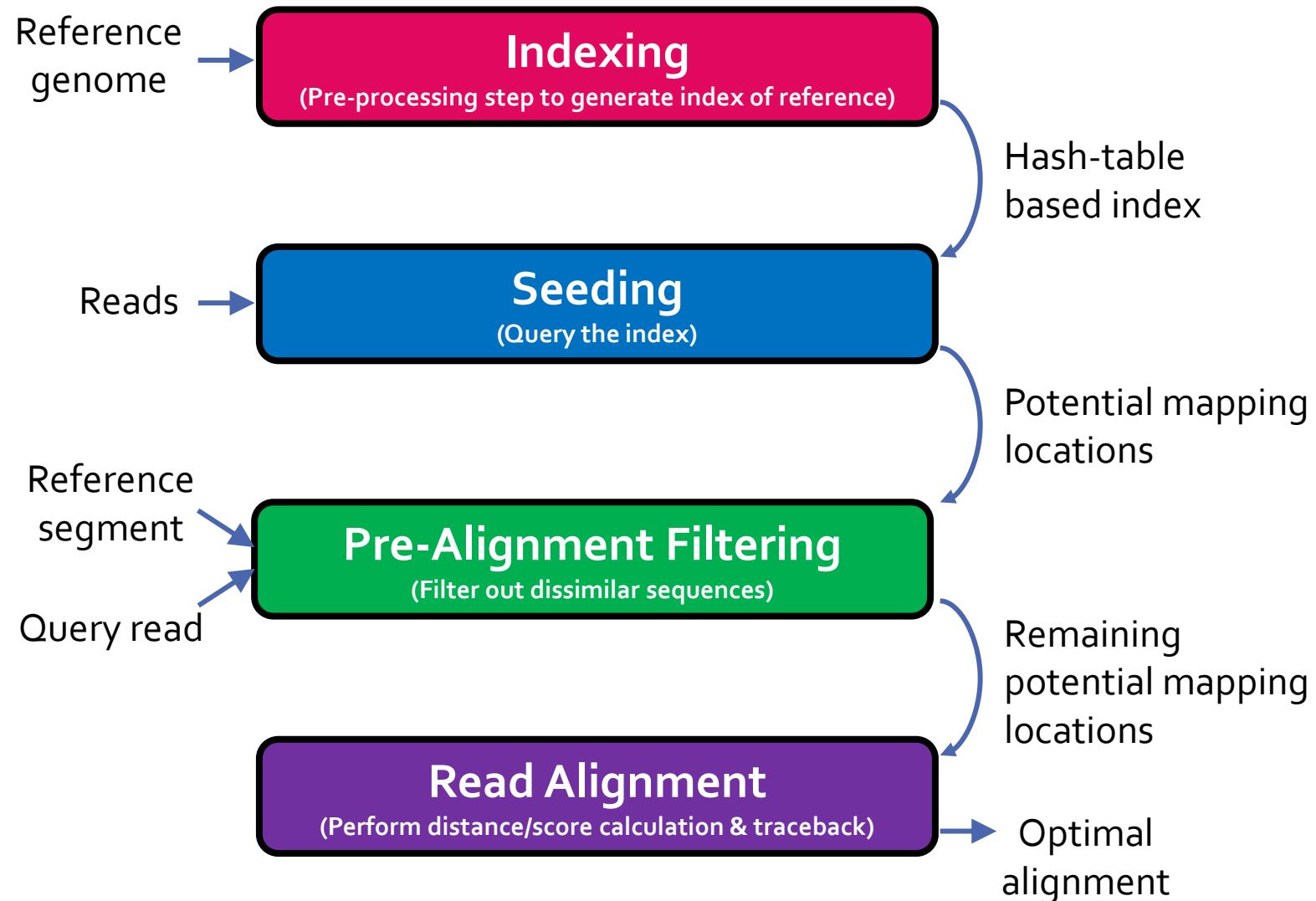
Read Mapping, method of aligning the reads against the reference genome in order to **detect matches and variations**.



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

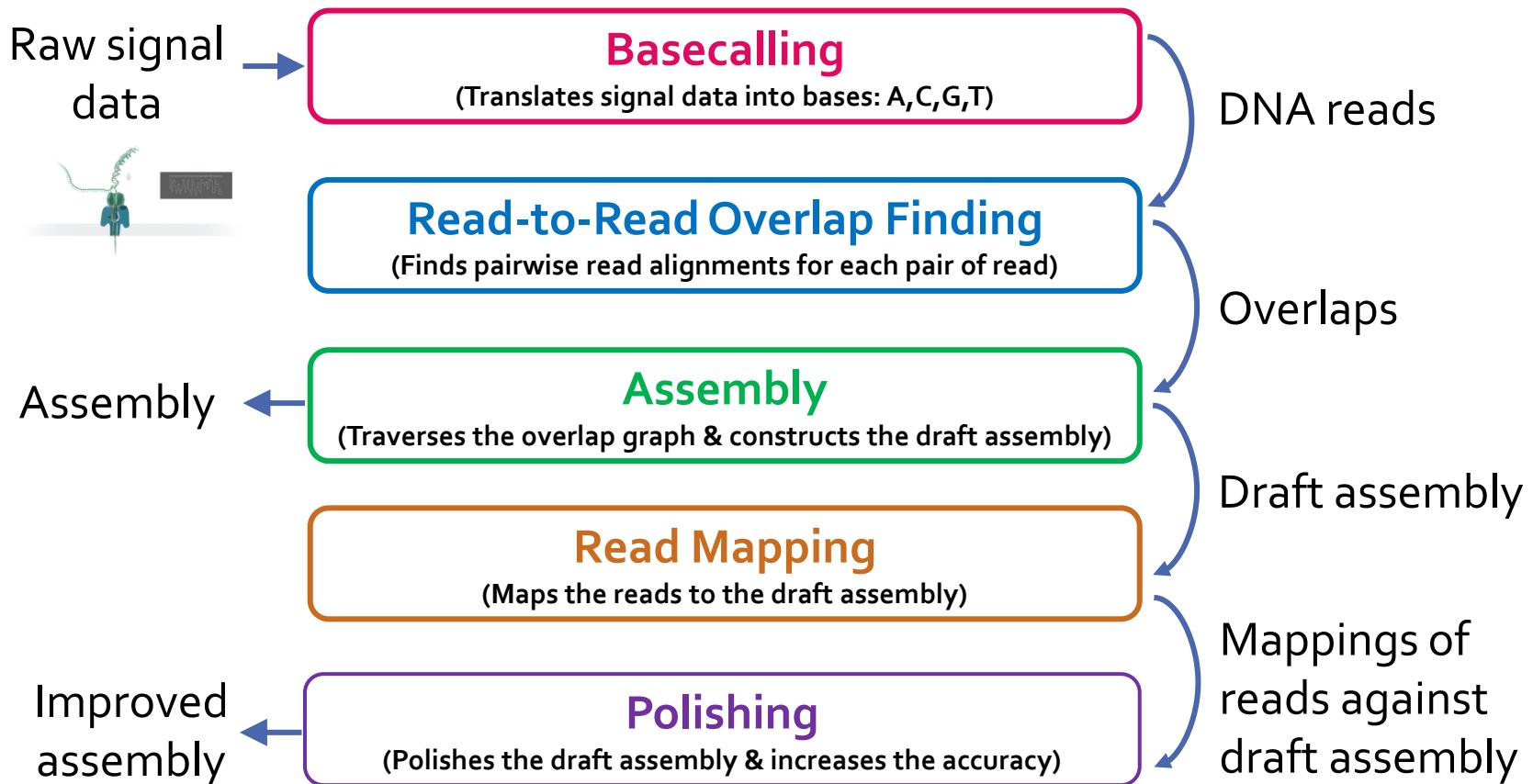


Read Mapping Pipeline



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.



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 Ghose¹, Can Alkan^{2*} and Onur Mutlu^{3,1*}

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

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