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

Damla Senol Cali, Ph.D.

https://damlasenolcali.github.io/

damlasenolcali@gmail.com

Staff Software Engineer, Hardware Acceleration

Bionano Genomics

AACBB Workshop @ ISCA 2022

June 18, 2022

Carnegie Mellon SAFARI ETH zürich

Genome Sequencing

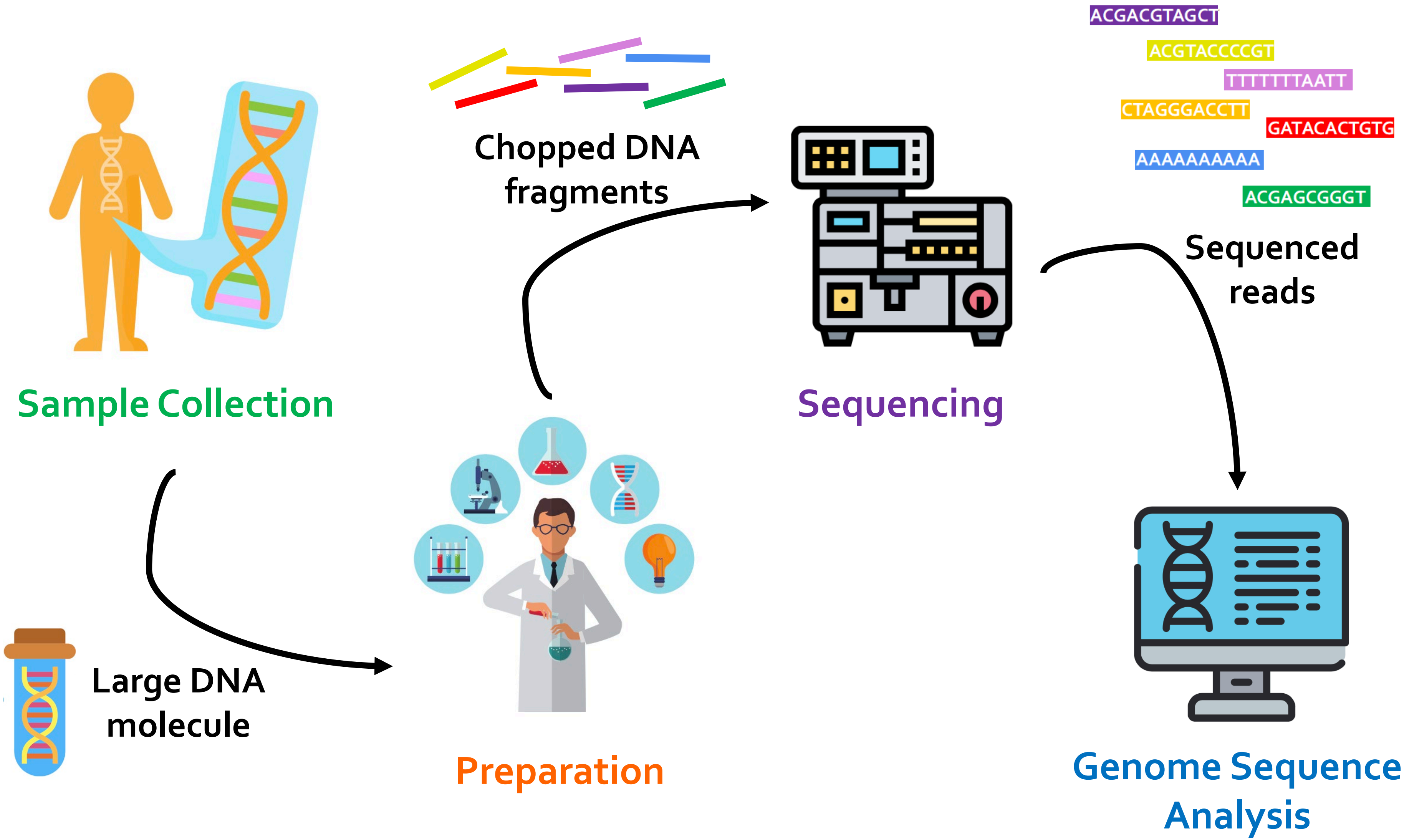
- □ **Genome sequencing:** Enables us to determine the order of the DNA sequence in an organism’s genome
  - ○ Plays a pivotal role in:
    - Personalized medicine
    - Outbreak tracing
    - Understanding of evolution

Genome

DNA

- □ **Challenges:**
  - ○ There is no sequencing machine that takes long DNA as an input, and gives the complete sequence as output
  - ○ Sequencing machines extract small randomized fragments of the original DNA sequence

Genome Sequencing (cont'd.)



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)

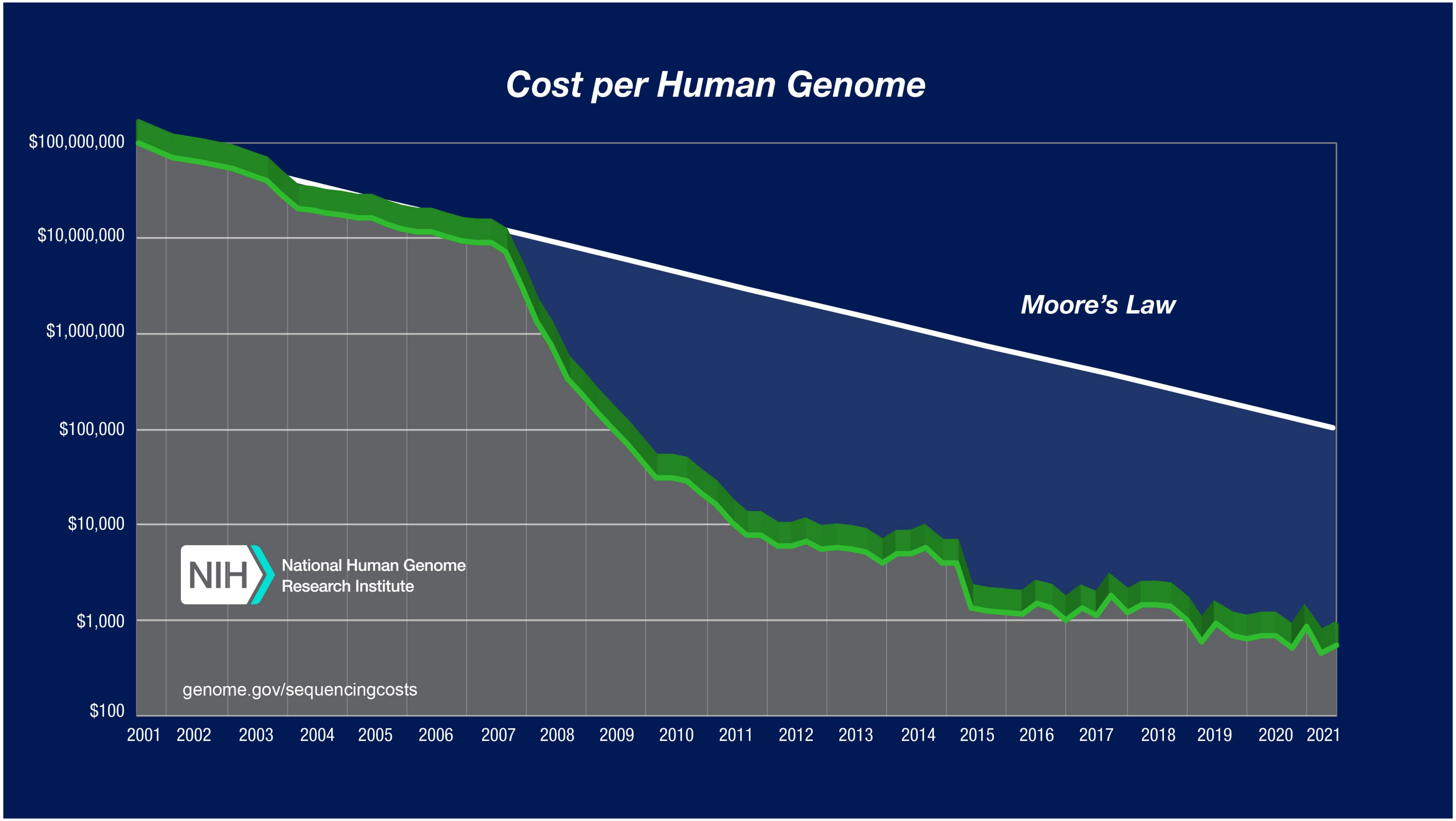
PacBio

Illumina

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

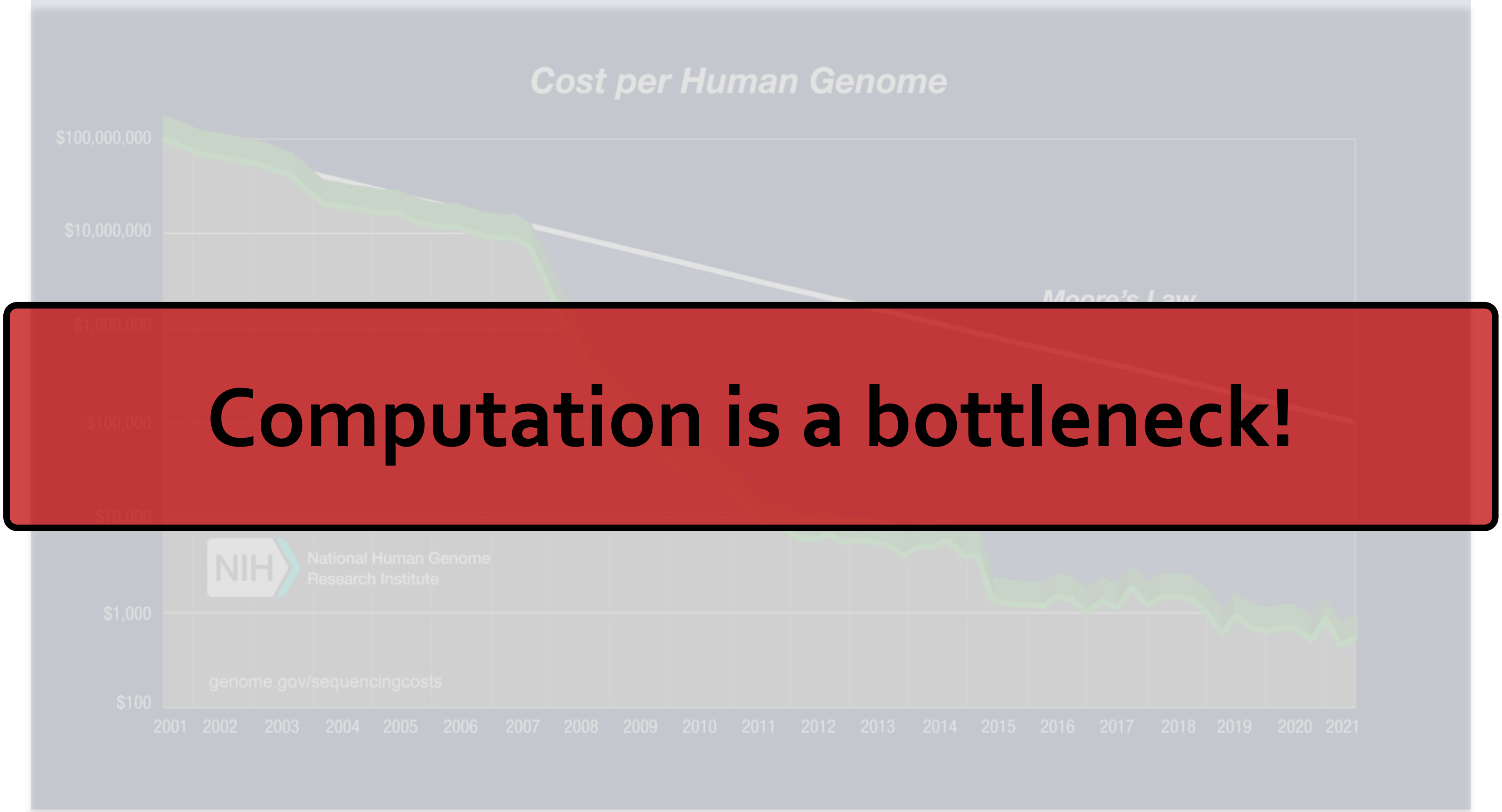
Long reads: **thousands to millions of base pairs** and **error rate of 5–10%**

Current State of Sequencing



*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)

Our Goal & Approach

- ❑ **Our Goal:**

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

- ❑ **Our Approach:**

  (1) Analyze the multiple steps and the associated tools in the genome sequence analysis pipeline,

  (2) Expose the tradeoffs between accuracy, performance, memory usage and scalability, and

  (3) Co-design fast and efficient algorithms along with scalable and energy-efficient customized hardware accelerators for the key bottleneck steps of the pipeline

Research Contributions

Bottleneck analysis of genome assembly pipeline for long reads

[Briefings in Bioinformatics, 2018]

GenASM: Approximate string matching framework for genome sequence analysis

[MICRO 2020]

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

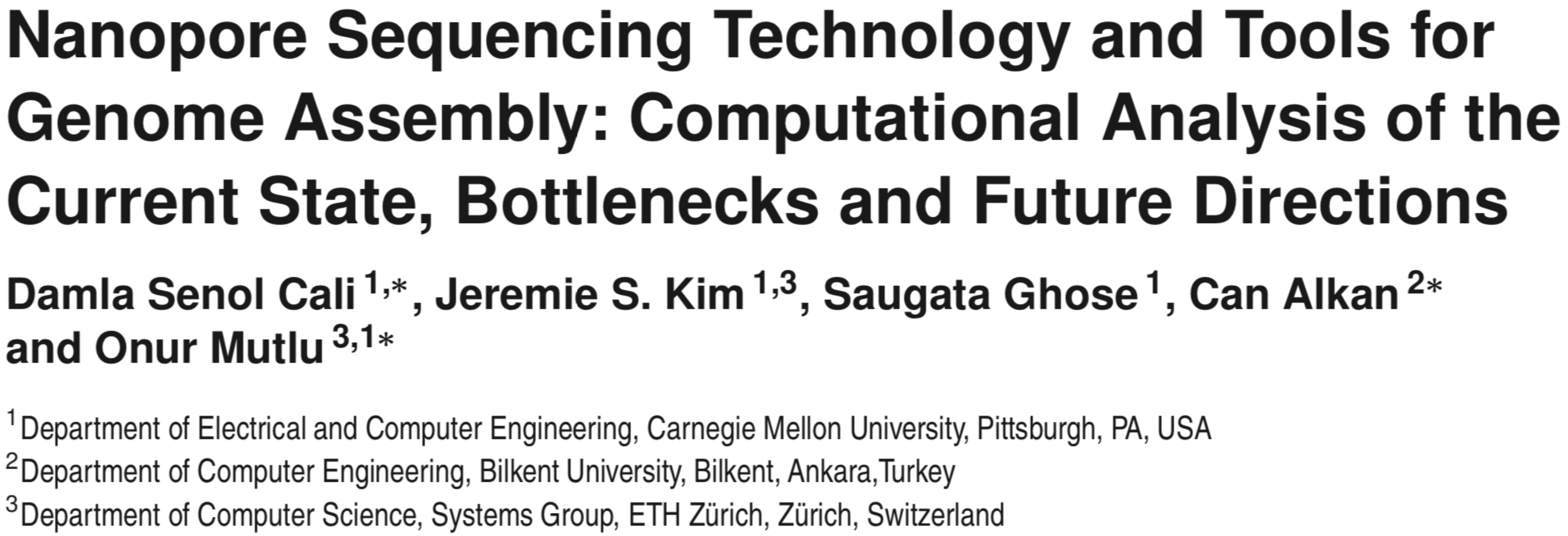
[Ongoing]

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

[ISCA 2022]

Nanopore Sequencing & Tools [BiB 2018]

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,* April 2018.



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

Key Findings

Goal 1:
High-performance and low-power

Goal 2:
Memory-efficient

Goal 3:
Scalable/highly-parallel

Research Contributions

Bottleneck analysis of genome assembly pipeline for long reads

[Briefings in Bioinformatics, 2018]

GenASM: Approximate string matching framework for genome sequence analysis

[MICRO 2020]

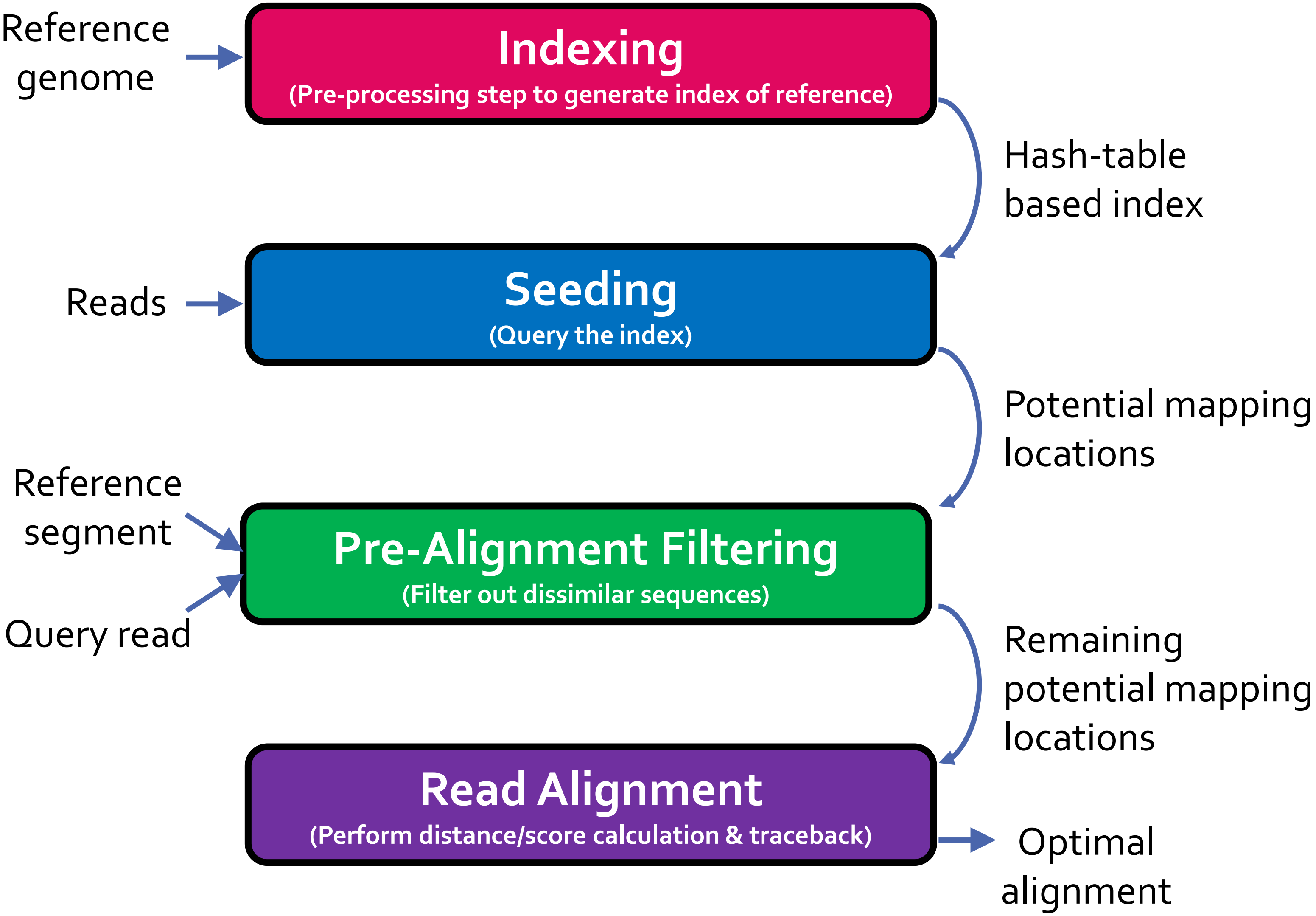
BitMAc: FPGA-based near-memory acceleration of bitvector-based sequence alignment

[Ongoing]

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

[ISCA 2022]

Read Mapping Pipeline



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

GSA with Read Mapping

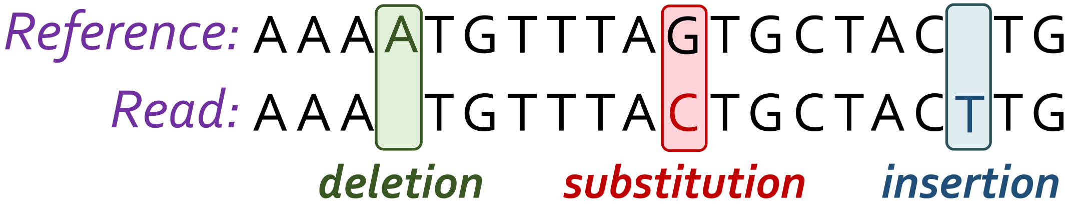
- **Read mapping:** *First key step* in genome sequence analysis (GSA)
  - Aligns reads to one or more possible locations within the reference genome, and
  - Finds the matches and differences between the read and the reference genome segment at that location

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

- Bottlenecked by the computational power and memory bandwidth limitations of existing systems

Approximate String Matching

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



Reference: AAAATGTTTAGTGCTAC TG

Read: AAA TGTTTACTGCTACTTG

deletion ***substitution*** ***insertion***

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

Bitap Algorithm

- 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.

Limitations of Bitap

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

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)

Limitations of Bitap

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

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

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)

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

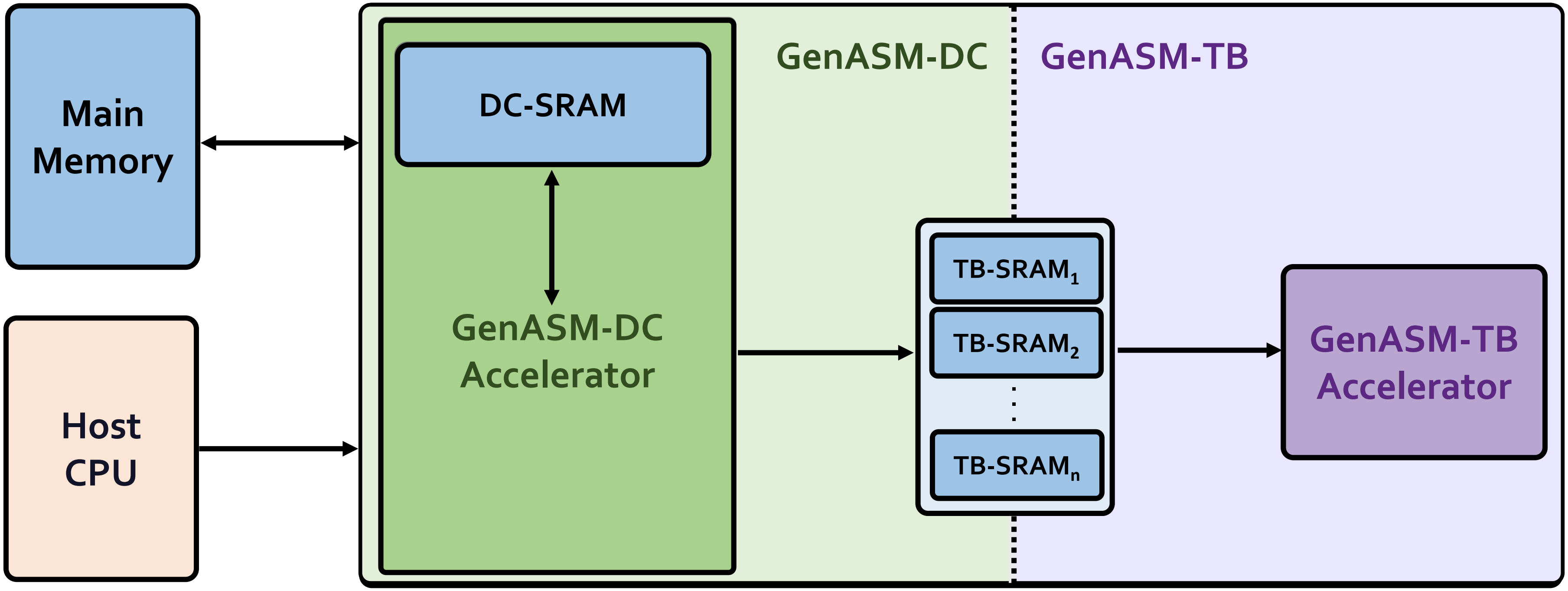
GenASM: ASM Framework for GSA

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**

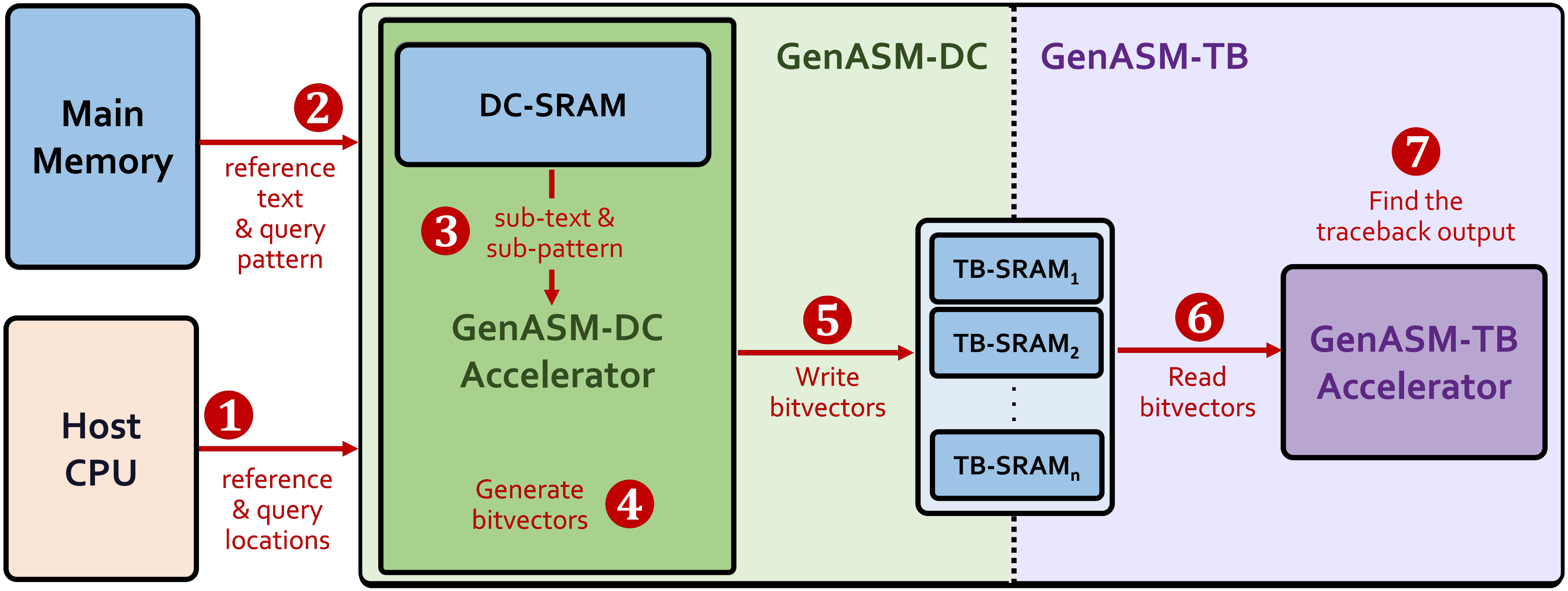
GenASM Hardware Design



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

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

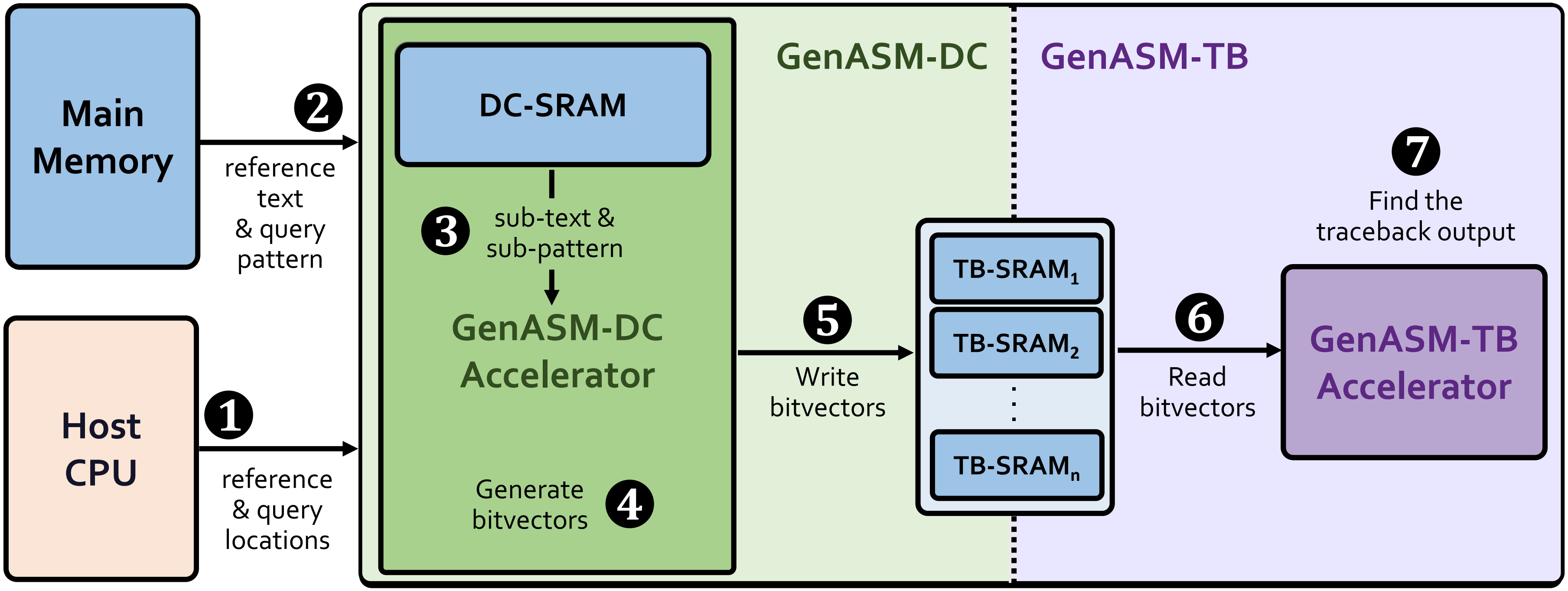
GenASM Hardware Design



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:

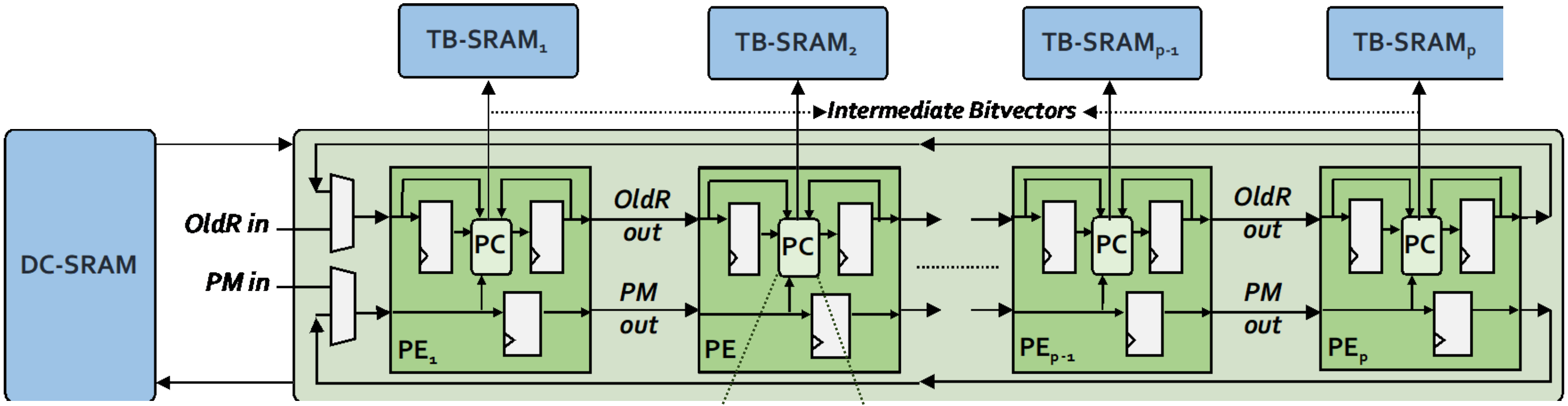
→ Match the rate of computation with memory capacity and bandwidth

→ Achieve high performance and power efficiency

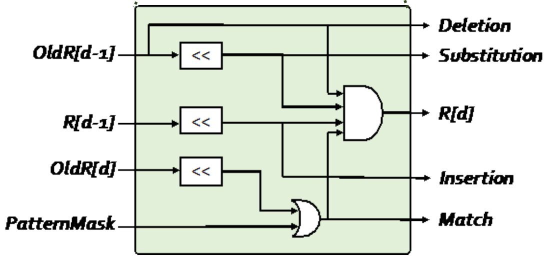
→ Scale linearly in performance with the number of parallel compute units that we add to the system

GenASM-DC: Hardware Design

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

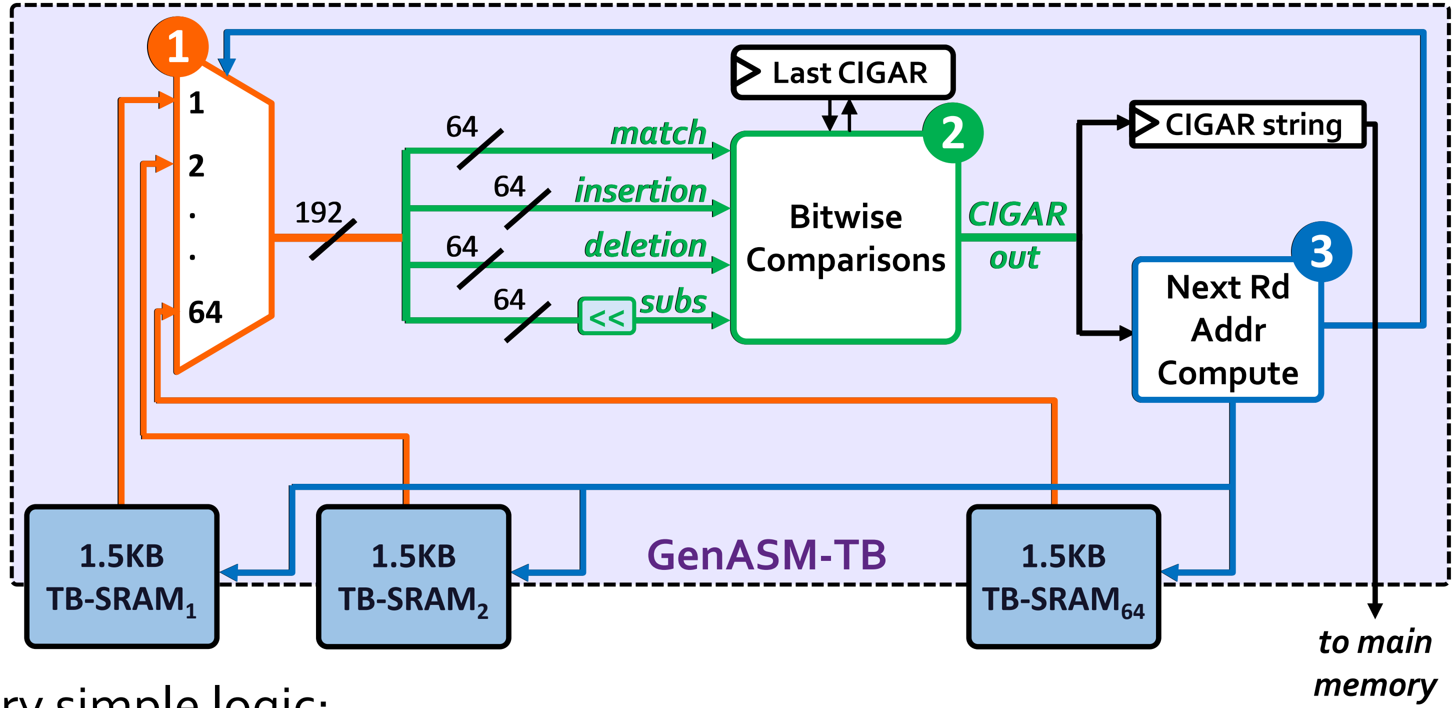


Processing Block (PB)



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

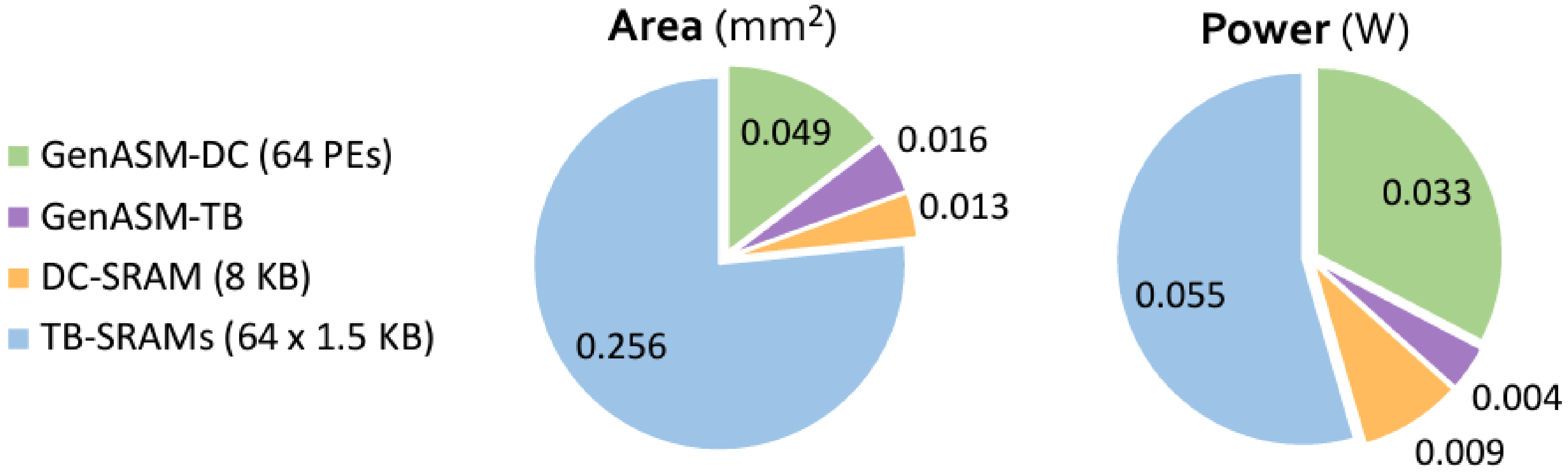
Evaluation Methodology (cont'd.)

- **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**



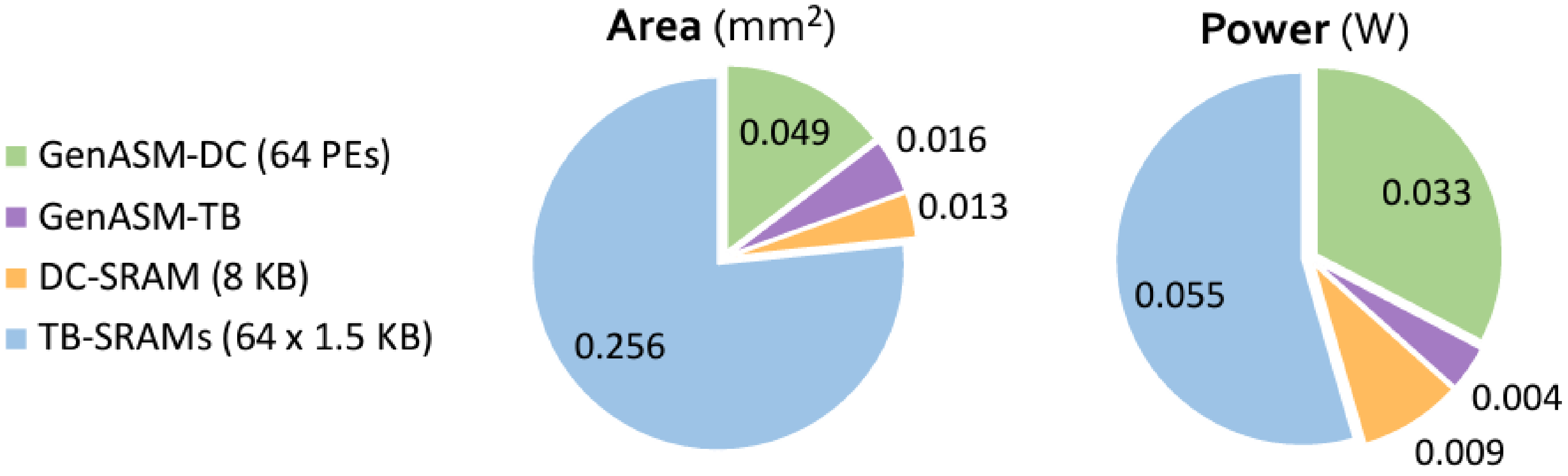
Area (mm^2)

Power (W)

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

(1) Read Alignment

❑ **116×** speedup, **37×** less power than **Minimap2** (state-of-the-art **SW**)

❑ **111×** speedup, **33×** less power than **BWA-MEM** (state-of-the-art **SW**)

❑ **3.9×** better throughput, **2.7×** less power than **Darwin** (state-of-the-art **HW**)

❑ **1.9×** better throughput, **82%** less logic power than **GenAx** (state-of-the-art **HW**)

(2) Pre-Alignment Filtering

❑ **3.7×** speedup, **1.7×** less power than **Shouji** (state-of-the-art **HW**), while significantly improving the accuracy of pre-alignment filtering

(3) Edit Distance Calculation

❑ **22–12501×** speedup, **548–582×** less power than **Edlib** (state-of-the-art **SW**)

❑ **9.3–400×** speedup, **67×** less power than **ASAP** (state-of-the-art **HW**)

Additional Details in the Paper

- ❑ Details of the **GenASM-DC and GenASM-TB algorithms**
- ❑ **Big-O analysis** of the algorithms
- ❑ Detailed explanation of **evaluated use cases**
- ❑ **Evaluation methodology details**

  (datasets, baselines, performance model)
- ❑ **Additional results** for the three evaluated use cases
- ❑ **Sources of improvements in GenASM**

  (algorithm-level, hardware-level, technology-level)
- ❑ Discussion of **four other potential use cases** of GenASM

Summary of GenASM

- **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

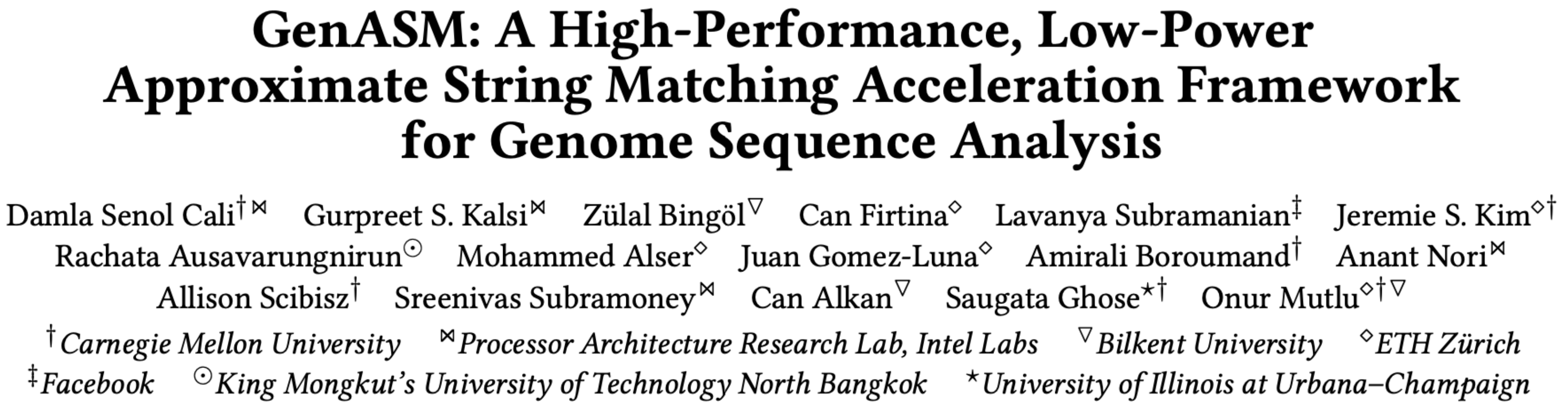
- **Key Results:** GenASM is significantly more efficient for all the three use cases (in terms of throughput and throughput per unit power) than state-of-the-art software and hardware baselines

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.



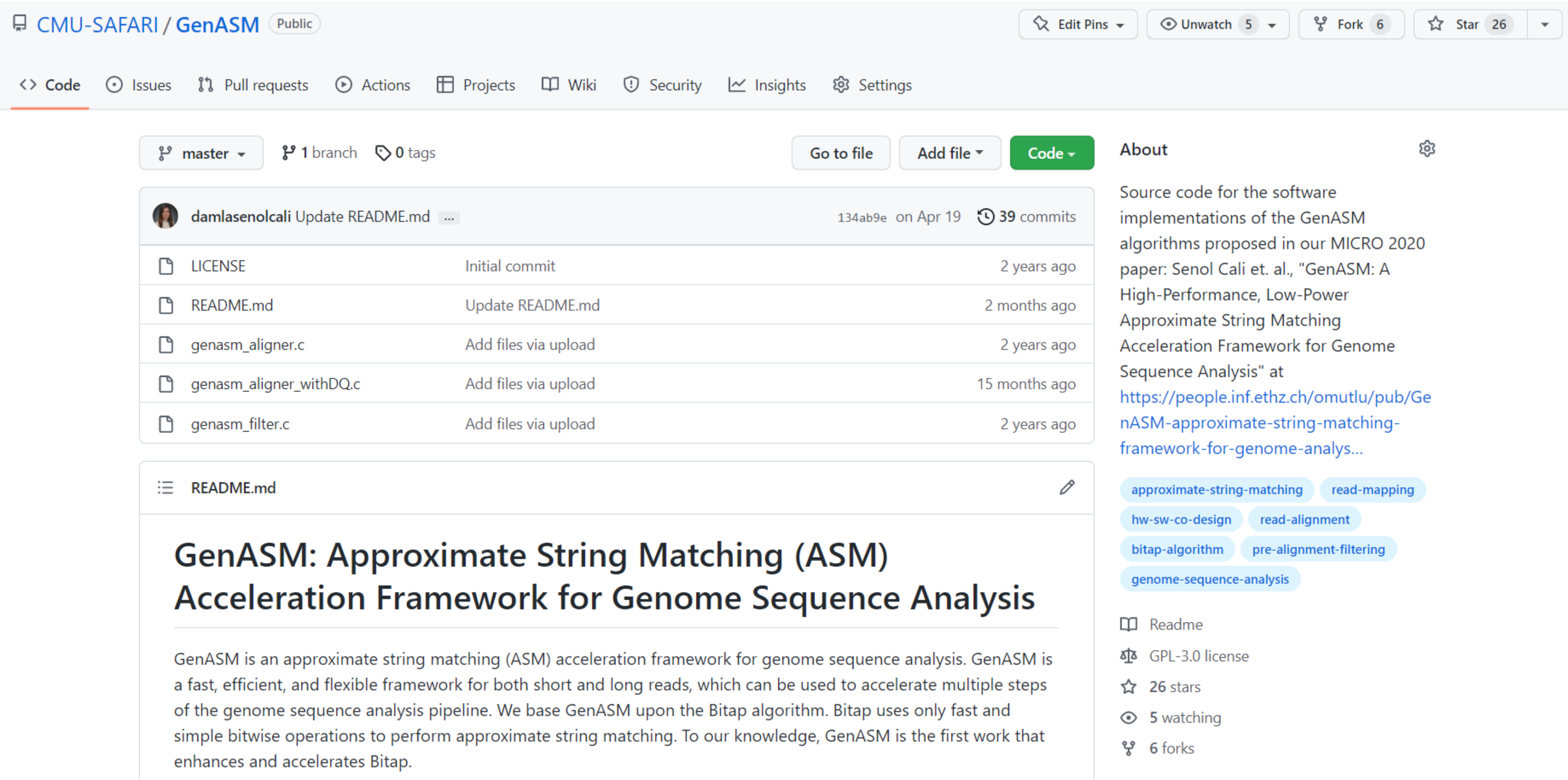
GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis

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

README.md

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.

About

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...

Research Contributions

Bottleneck analysis of genome assembly pipeline for long reads

[Briefings in Bioinformatics, 2018]

GenASM: Approximate string matching framework for genome sequence analysis

[MICRO 2020]

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

[Ongoing]

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

[ISCA 2022]

Genome Sequence Analysis

- ***Sequence-to-sequence mapping (traditional read mapping)*:**
  - *Critical step* in genome sequence analysis (GSA)
  - Maps *reads* collected from an individual to a known *linear reference genome sequence*
  - Well studied with many available tools and accelerators
- Recent works replace 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
- ***Sequence-to-graph mapping*** results in notable quality improvements in GSA
  - More difficult computational problem
  - Much smaller number of practical software tools currently available
  - No prior hardware design for graph-based GSA

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

ACGTACGT

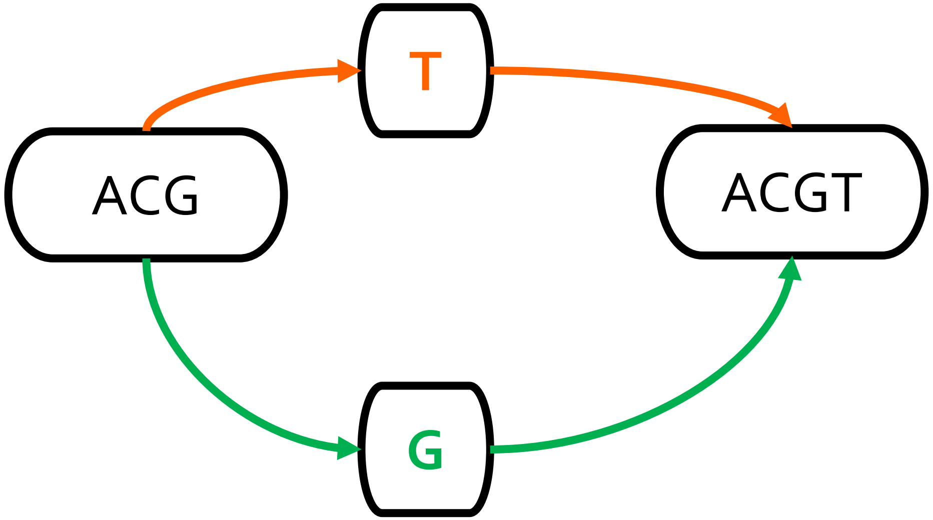
Genome Graphs

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

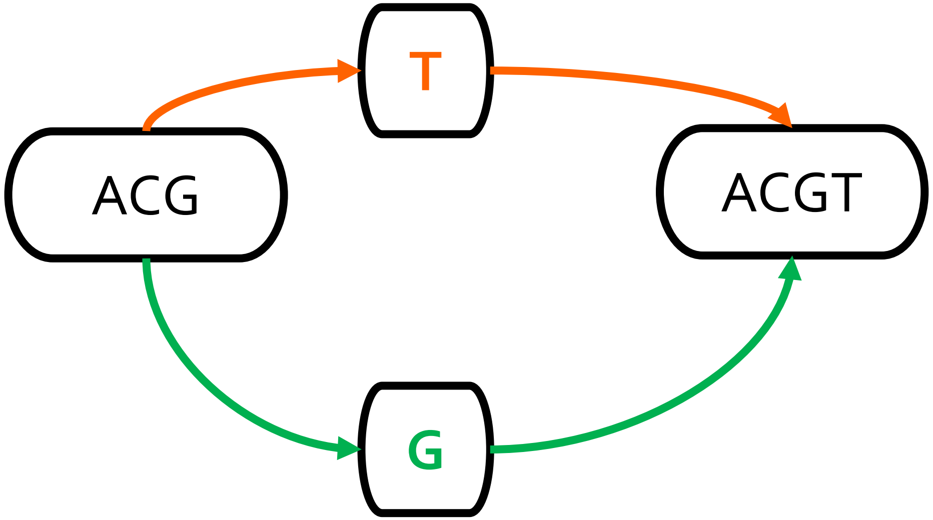
Genome graphs:

- ❑ 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

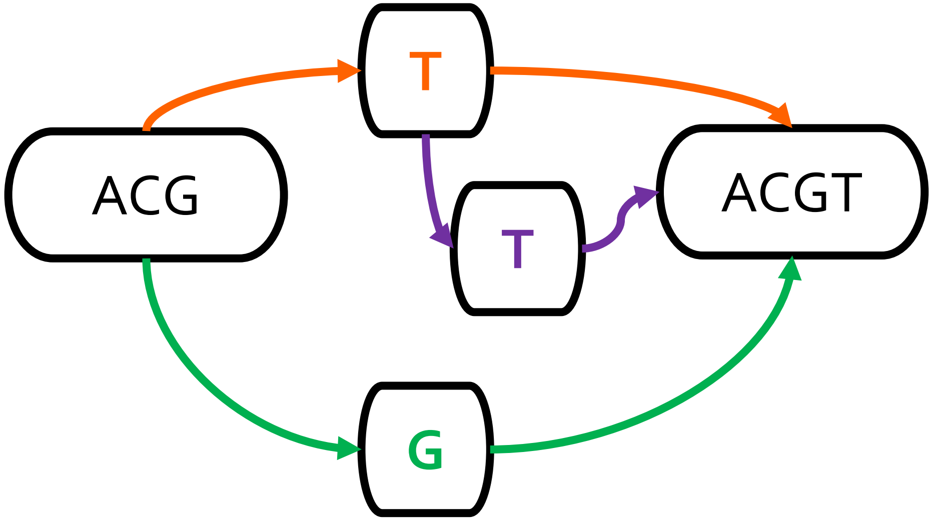
Genome graphs:

- ❑ 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:

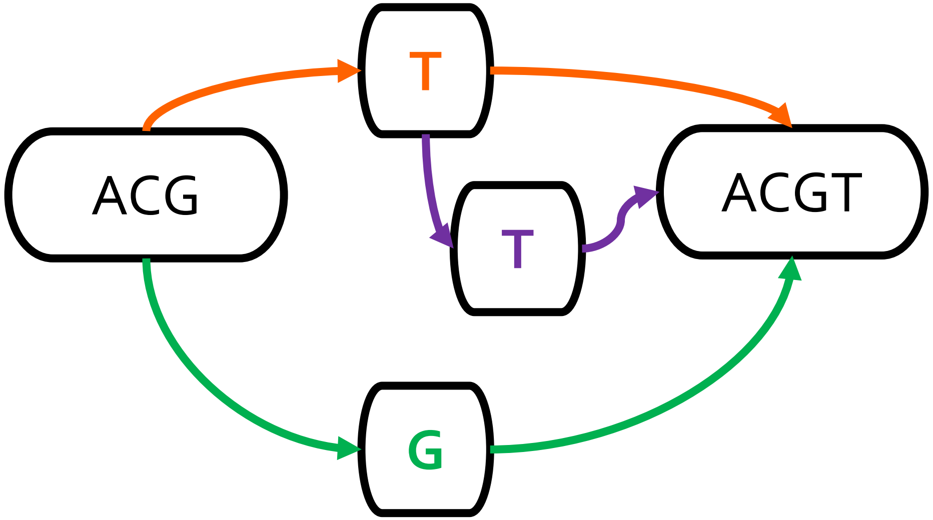
- ❑ Combine the linear reference genome with the known genetic variations in the entire population as a graph-based data structure
- ❑ Enable us to move away from aligning with a single linear reference genome (reference bias) and more accurately express the genetic diversity in a population

Sequence #1: ACGTACGT

Sequence #2: ACGGACGT

Sequence #3: ACGTTACGT

Sequence #4: ACGACGT



Genome Graphs

Genome graphs:

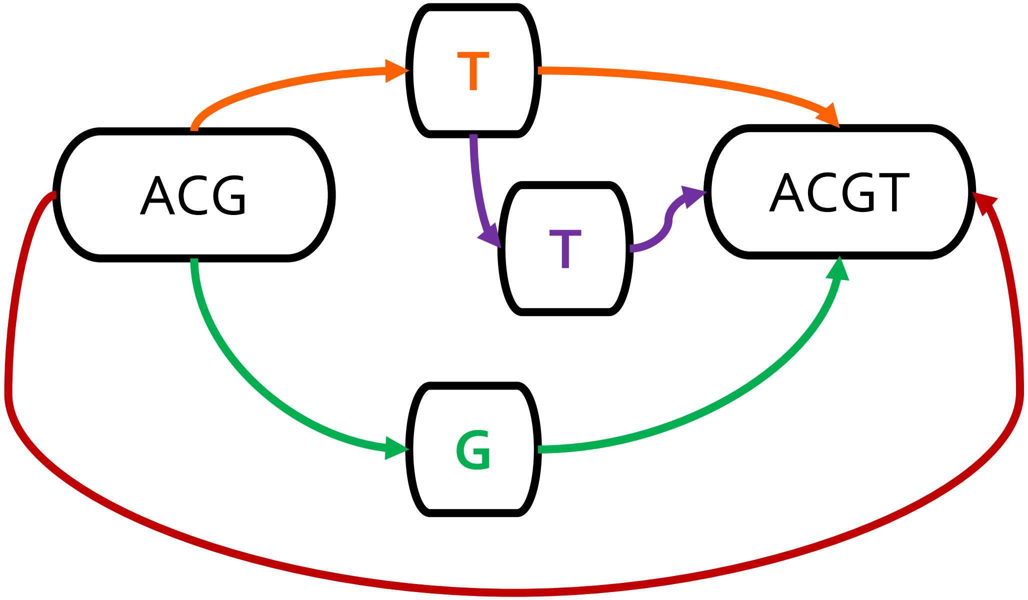
- ❑ Combine the linear reference genome with the known genetic variations in the entire population as a graph-based data structure
- ❑ Enable us to move away from aligning with a single linear reference genome (reference bias) and more accurately express the genetic diversity in a population

Sequence #1: ACGTACGT

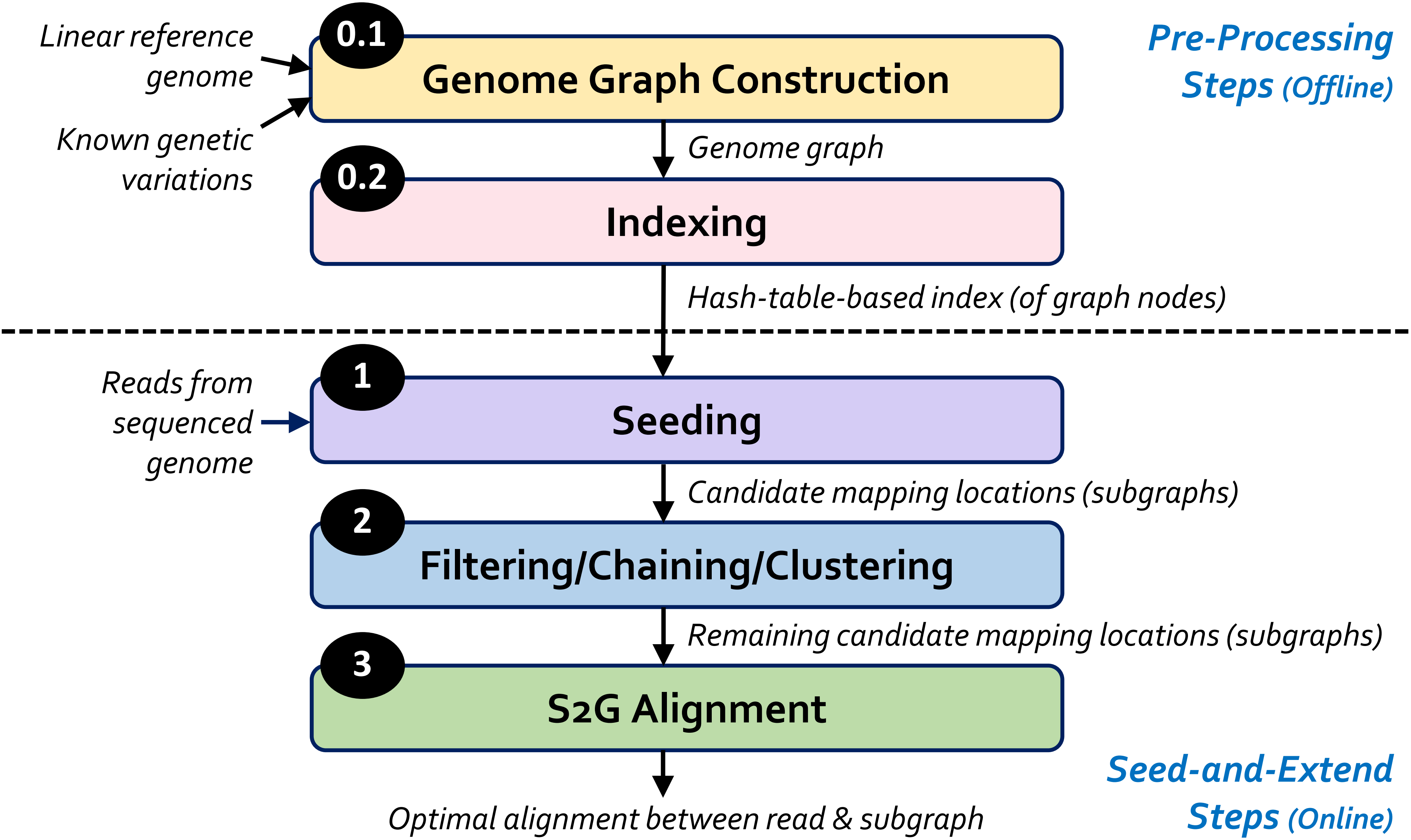
Sequence #2: ACGGACGT

Sequence #3: ACGTTACGT

Sequence #4: ACGACGT



Sequence-to-Graph Mapping Pipeline



Linear reference genome

Known genetic variations

0.1 **Genome Graph Construction**

Genome graph

0.2 **Indexing**

Hash-table-based index (of graph nodes)

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

Reads from sequenced genome

1 **Seeding**

Candidate mapping locations (subgraphs)

2 **Filtering/Chaining/Clustering**

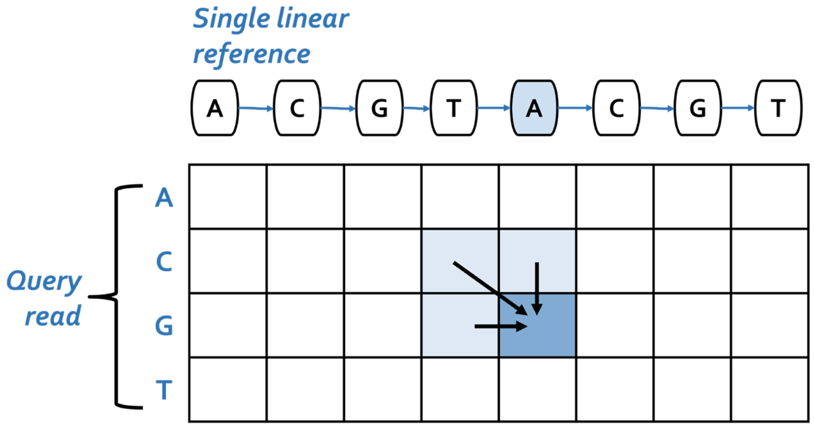
Remaining candidate mapping locations (subgraphs)

3 **S2G Alignment**

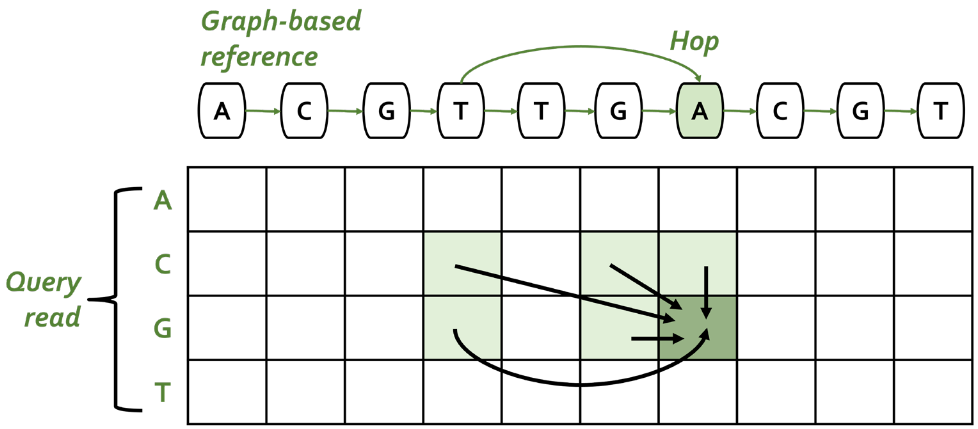
Optimal alignment between read & subgraph

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

S2S vs. S2G Alignment



(a) Sequence-to-Sequence Alignment



(b) Sequence-to-Graph 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

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

SeGraM: Universal Genomic Mapping Accelerator

Our Goal:

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

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

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

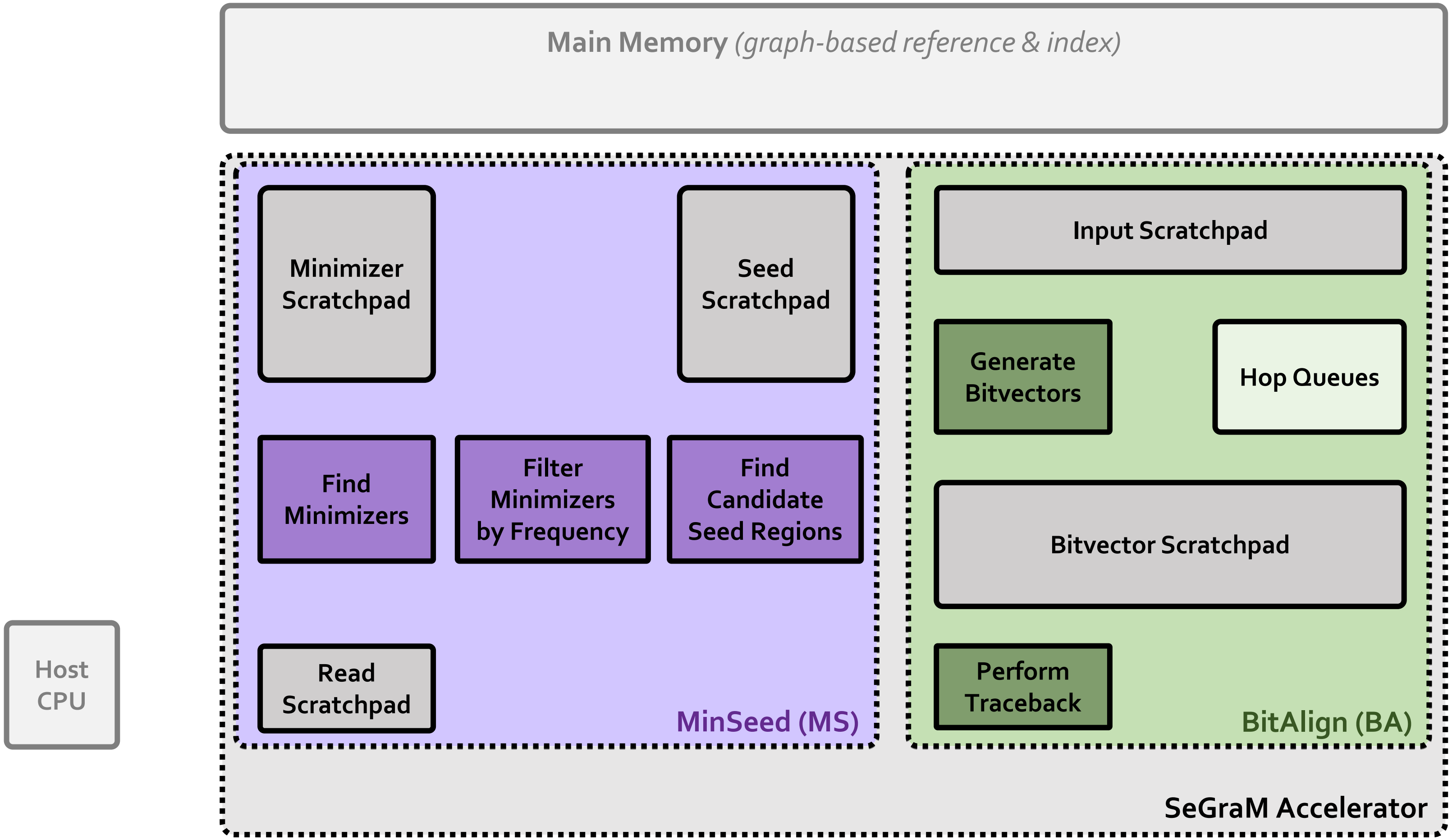
❑ We base SeGraM upon a minimizer-based seeding algorithm and a novel bitvector-based alignment algorithm

SW

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

HW

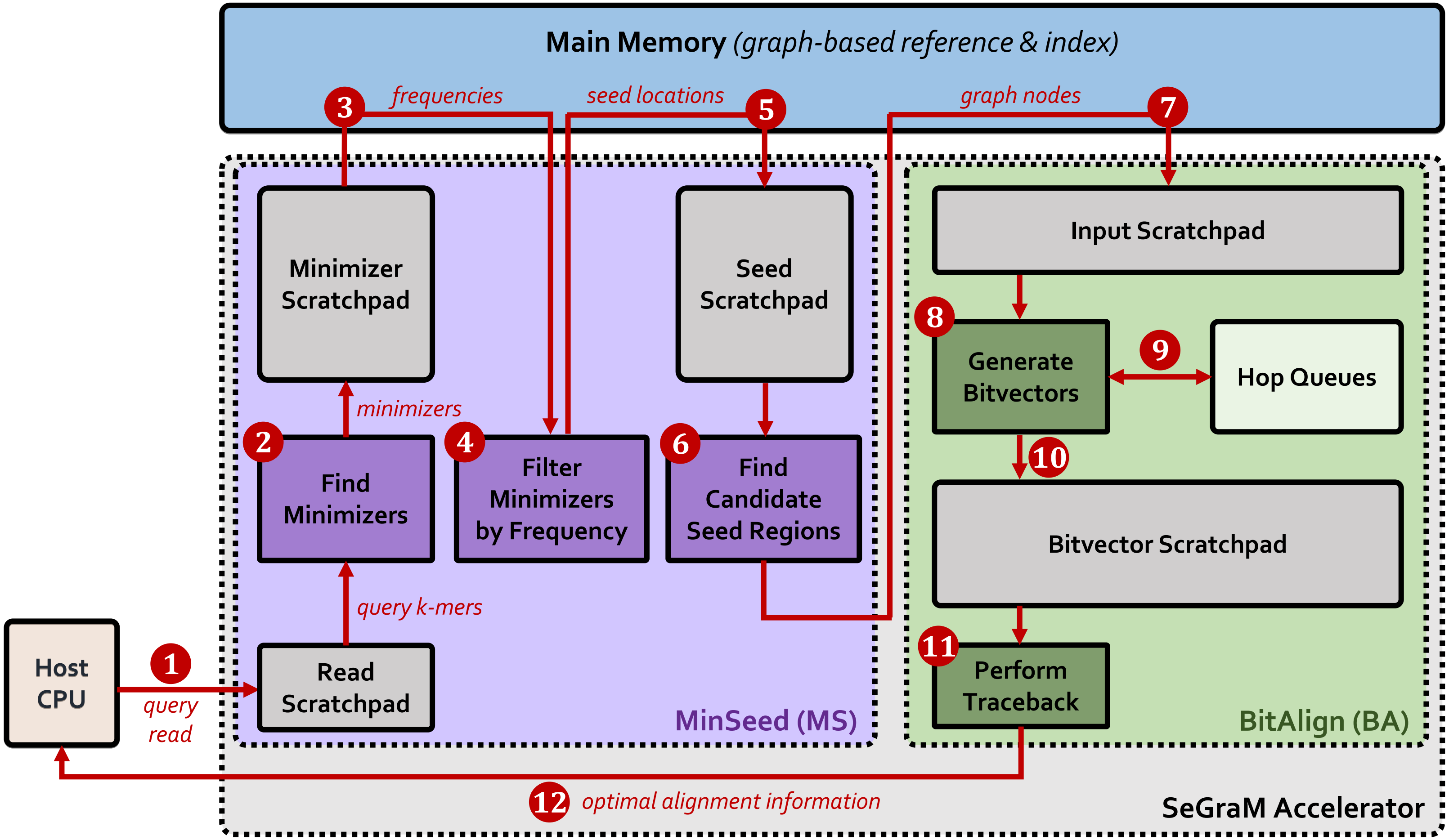
SeGraM Hardware Design



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

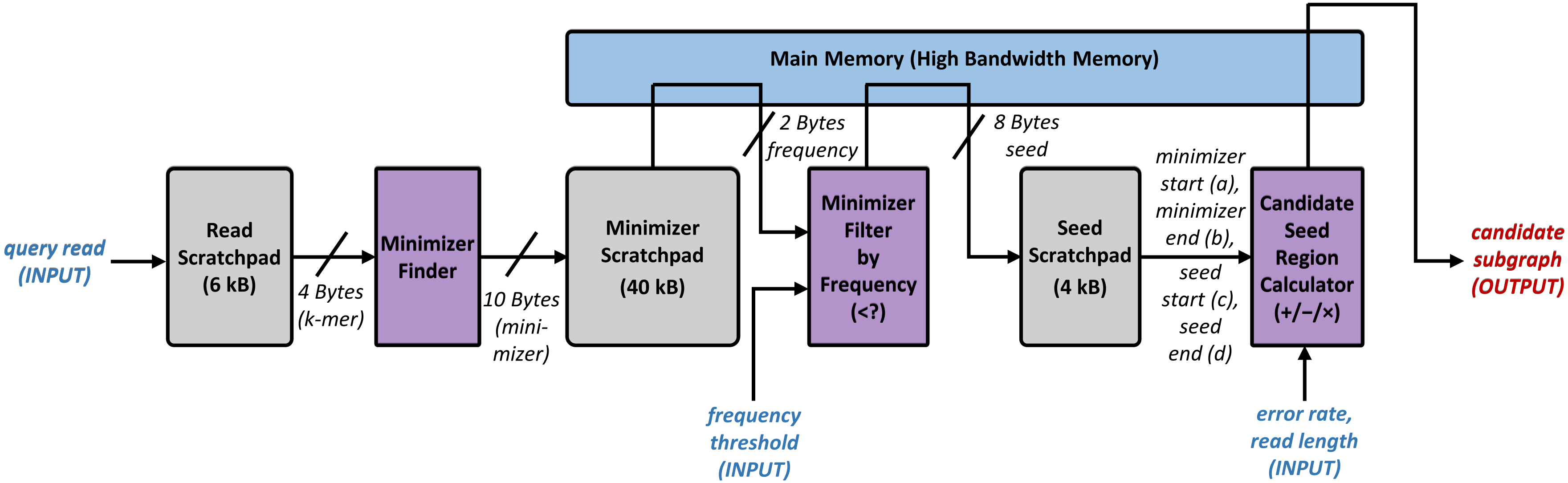


MinSeed: *first* hardware accelerator for **Min**imizer-based **Seed**ing

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

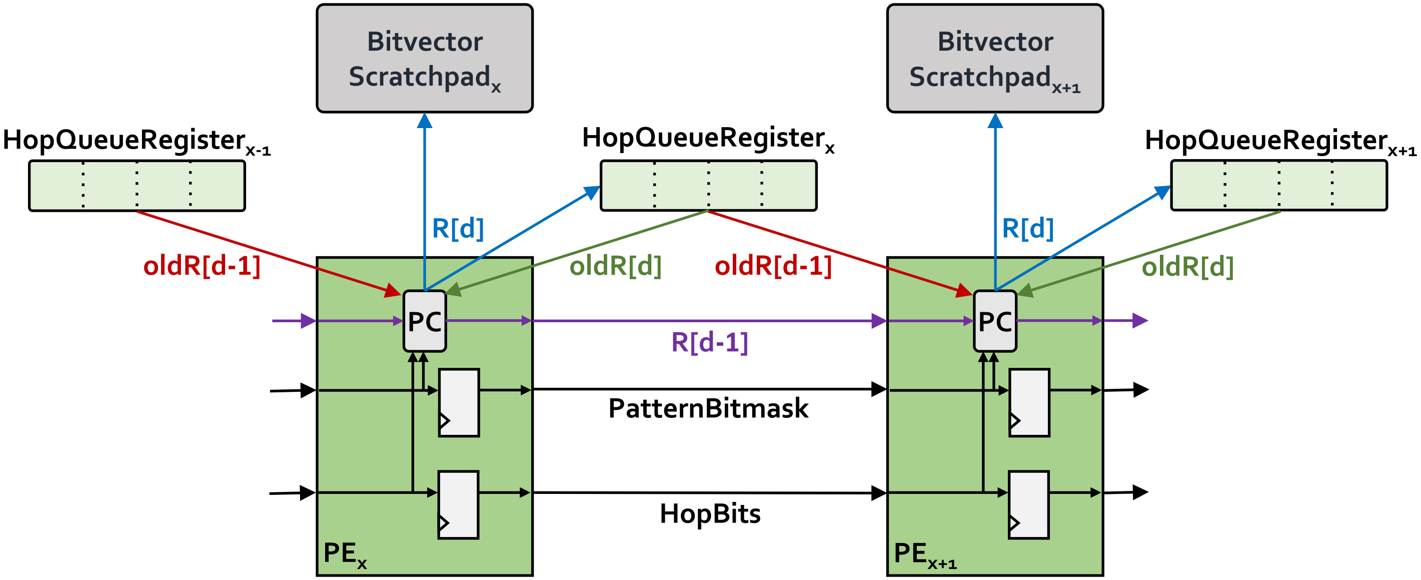
MinSeed HW

- MinSeed accelerator consists of three computation modules, three scratchpads, and the memory interface
  - Computation modules are implemented with simple logic
  - For all three scratchpads, we employ a double buffering technique to hide the latency of the MinSeed accelerator
  - We couple MinSeed with High-Bandwidth Memory (HBM) to enable low-latency and highly-parallel memory access

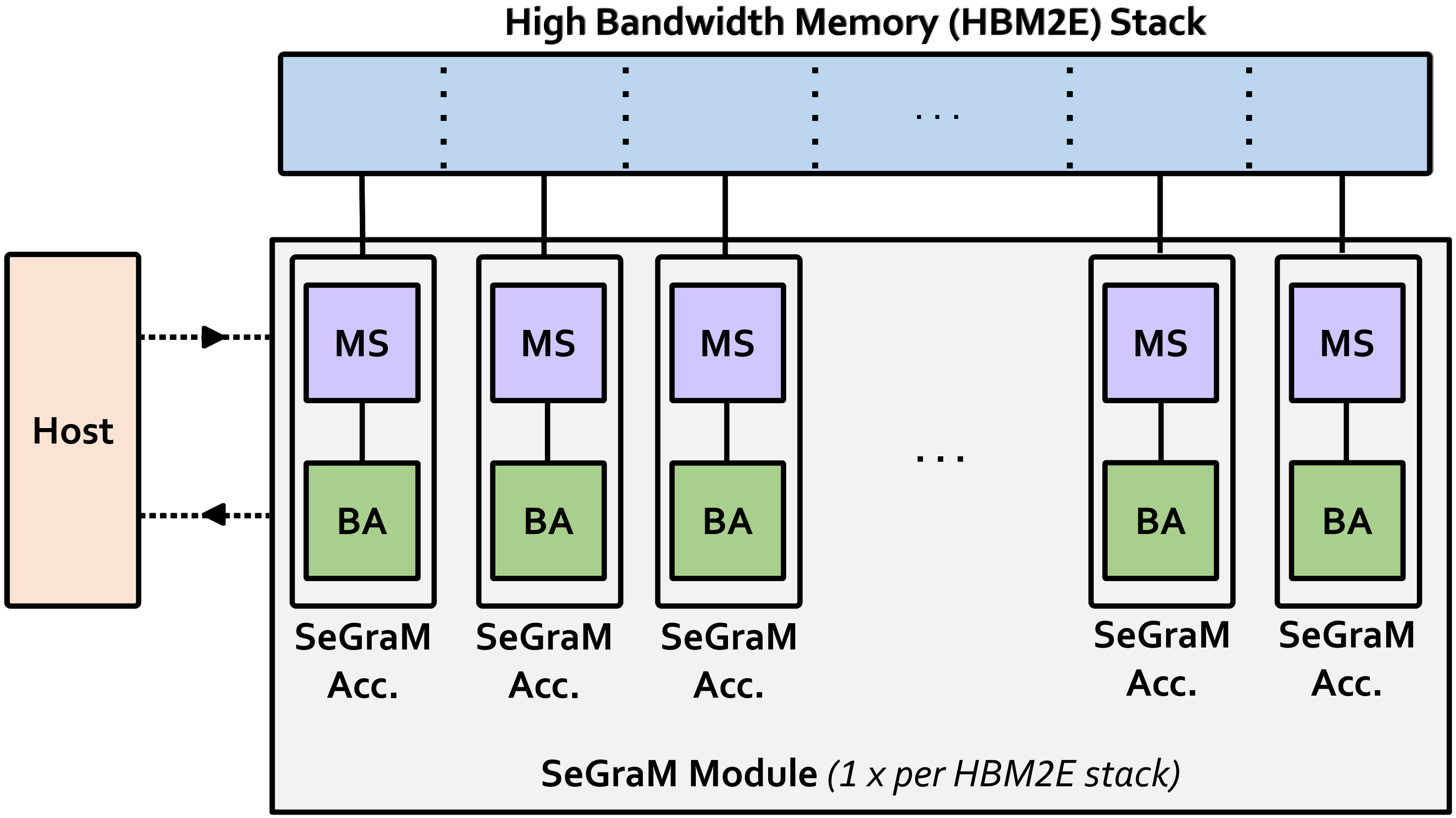

Main Memory (High Bandwidth Memory)
query read (INPUT)
Read Scratchpad (6 kB)
4 Bytes (k-mer)
Minimizer Finder
10 Bytes (mini-mizer)
Minimizer Scratchpad (40 kB)
2 Bytes frequency
Minimizer Filter by Frequency (<?)
frequency threshold (INPUT)
8 Bytes seed
Seed Scratchpad (4 kB)
minimizer start (a), minimizer end (b),
seed start (c), seed end (d)
Candidate Seed Region Calculator (+/−/×)
error rate, read length (INPUT)
candidate subgraph (OUTPUT)

BitAlign HW

- Linear cyclic systolic array-based accelerator
- Incorporates *hop queue registers* to feed the bitvectors of non-neighboring characters/nodes (i.e., *hops*)
- We implement the hop information between nodes of the graph as an adjacency matrix called **HopBits**
- Based on empirical analysis, we select **12** as the *hop limit*



Overall System Design of SeGraM



Channels
(8 × per HBM2E stack)

MinSeed (MS) HW
(1 × per channel;
8 × per module)
+
BitAlign (BA) HW
(1 × per MinSeed HW;
8 × per module)

=

SeGraM Accelerator
(8 × per module)

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

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 – 32 SeGraM Accelerators	**27.744**	**24256.0 (24.3 W)**
HBM2E (4 stacks)	**--**	**3.8 W**

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**)

Additional Details in the Paper

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

Summary of SeGraM

Problem:

- Sequence-to-sequence (S2S) mapping causes reference bias
- Recent works replace the linear reference sequence with a graph-based representation of the reference genome
- Sequence-to-graph (S2G) mapping is a more difficult computational problem, with a much smaller number of practical software tools

Key Contributions:

- SeGraM: *Universal algorithm/hardware co-designed genomic mapping accelerator that supports both S2G and S2S mapping*
  - MinSeed: *First* minimizer-based seeding accelerator
  - BitAlign: *First* sequence-to-graph alignment accelerator based upon our new bitvector-based, highly-parallel algorithm

Key Results:

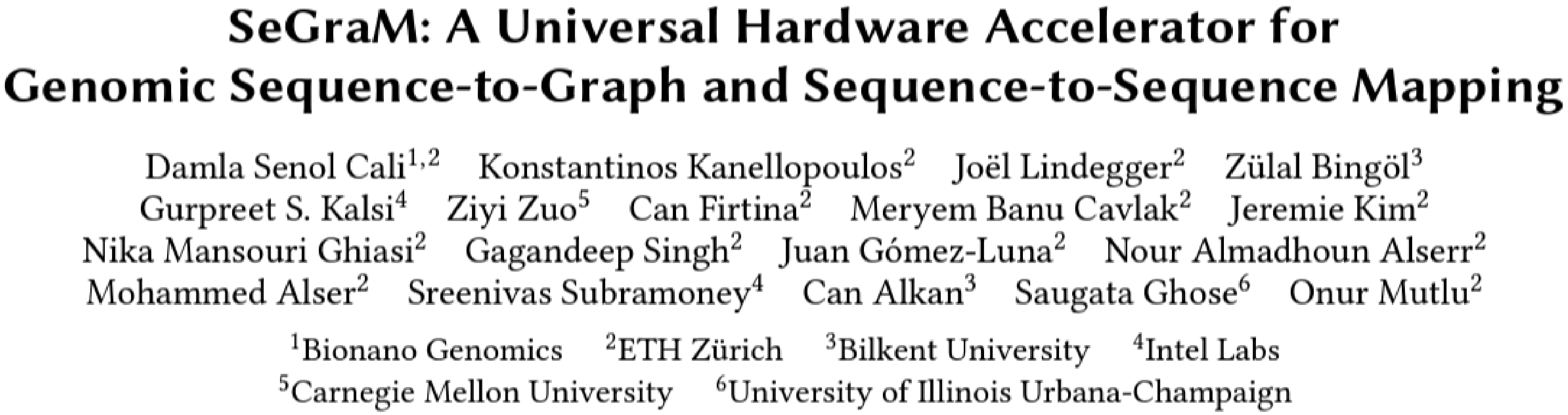
- SeGraM provides greatly higher throughput and lower power consumption compared to state-of-the-art SW tools for S2G mapping
- BitAlign significantly outperforms a state-of-the-art S2G alignment tool and three state-of-the-art HW solutions for S2S alignment

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.



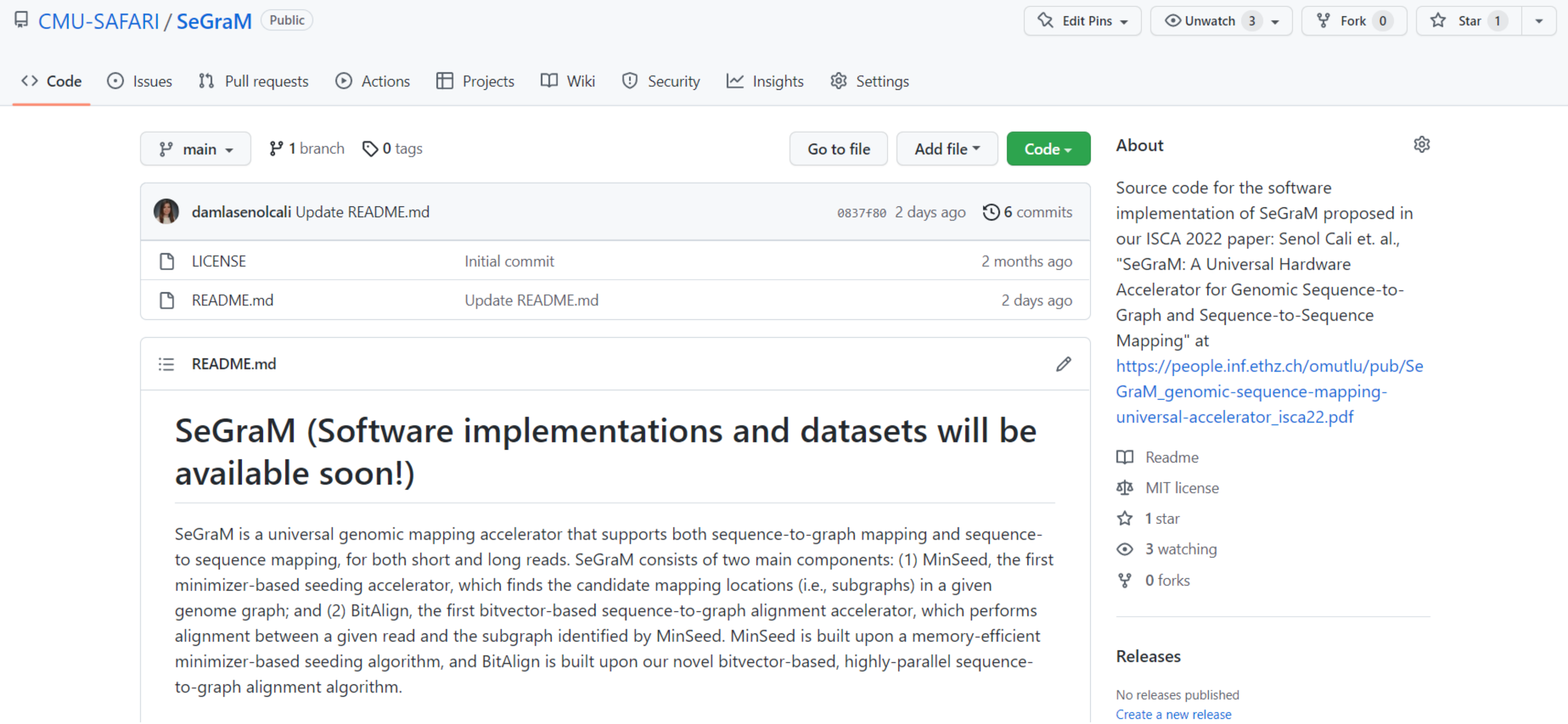
SeGraM: A Universal Hardware Accelerator for Genomic Sequence-to-Graph and Sequence-to-Sequence Mapping

Damla Senol Cali[1,2] 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

Readme
MIT license
1 star
3 watching
0 forks

Releases

No releases published
Create a new release

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

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

Staff Software Engineer, Hardware Acceleration
Bionano Genomics

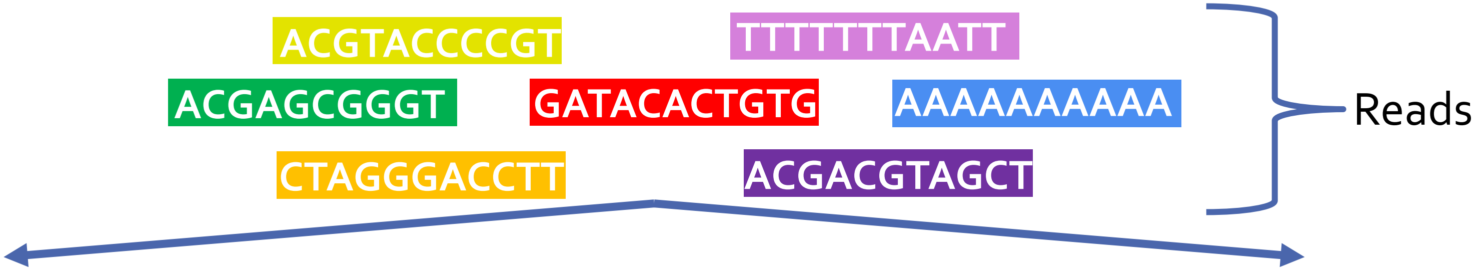
AACBB Workshop @ ISCA 2022
June 18, 2022

Carnegie Mellon SAFARI ETH zürich

Backup Slides

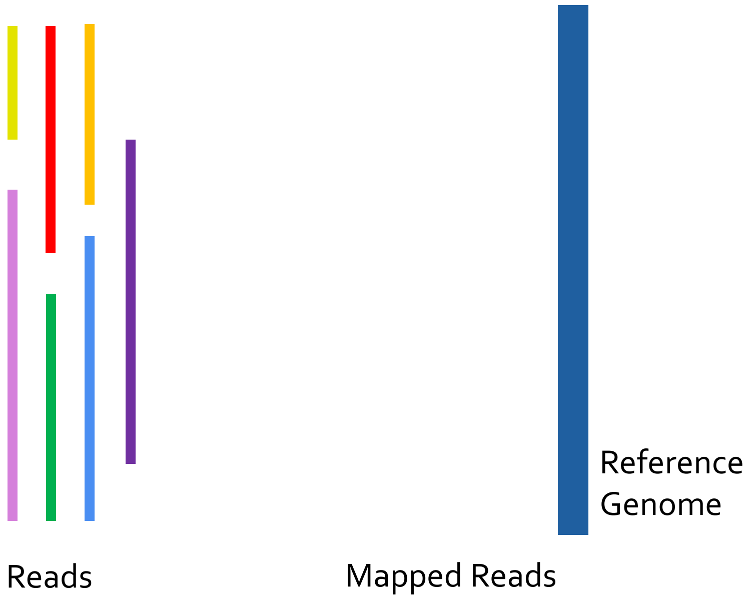
(BiB Paper)

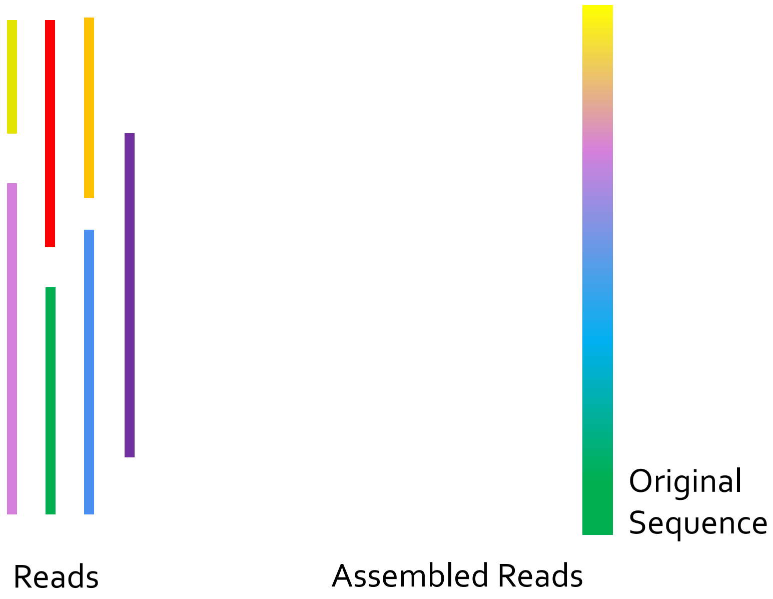
Genome Sequence Analysis



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

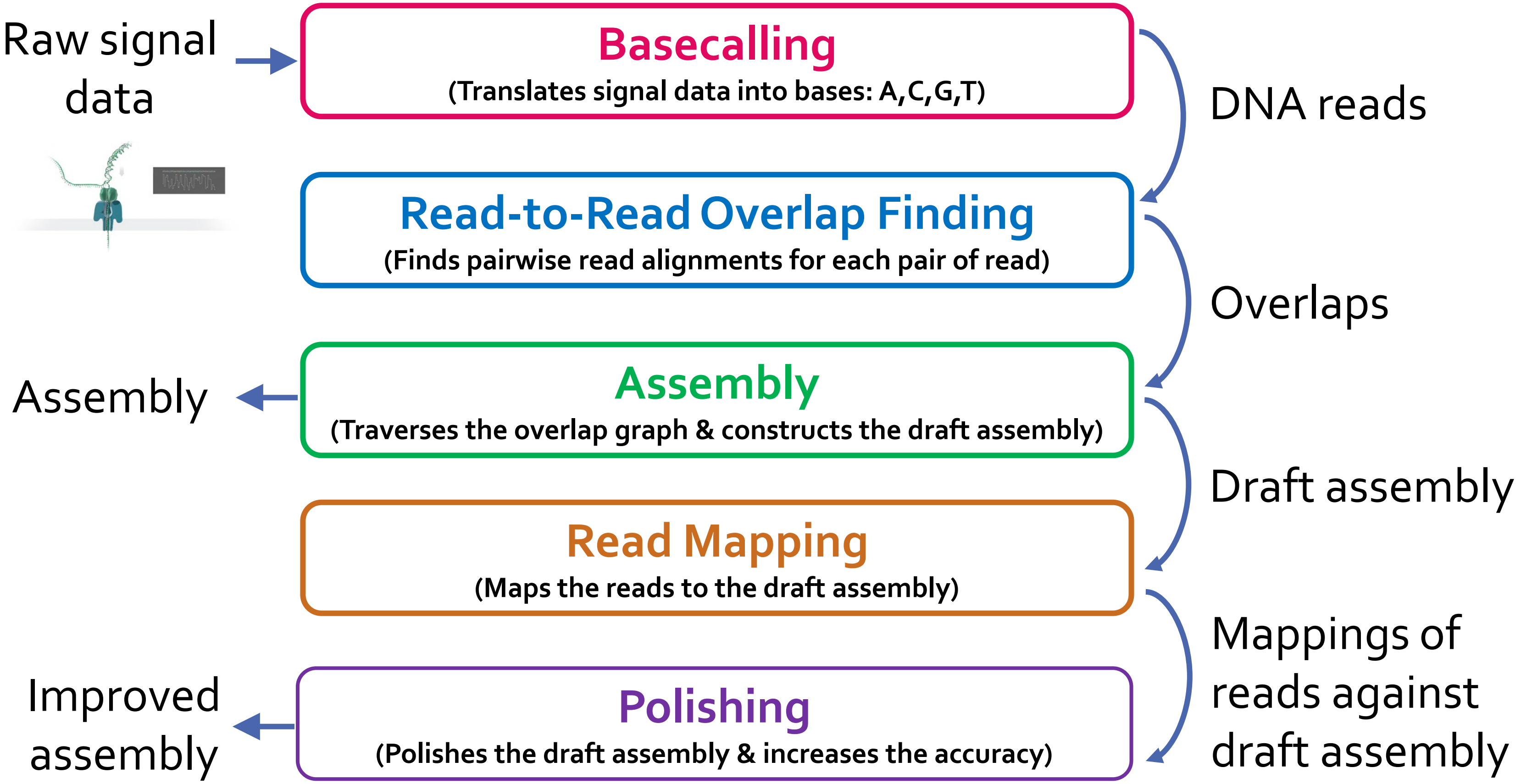
***De novo* Assembly,** method of merging the reads in order to **construct** the original sequence.





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

Assembly

(Traverses the overlap graph & constructs the draft assembly)

Draft assembly

Read Mapping

(Maps the reads to the draft assembly)

Mappings of reads against draft assembly

Improved assembly

Polishing

(Polishes the draft assembly & increases the accuracy)

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

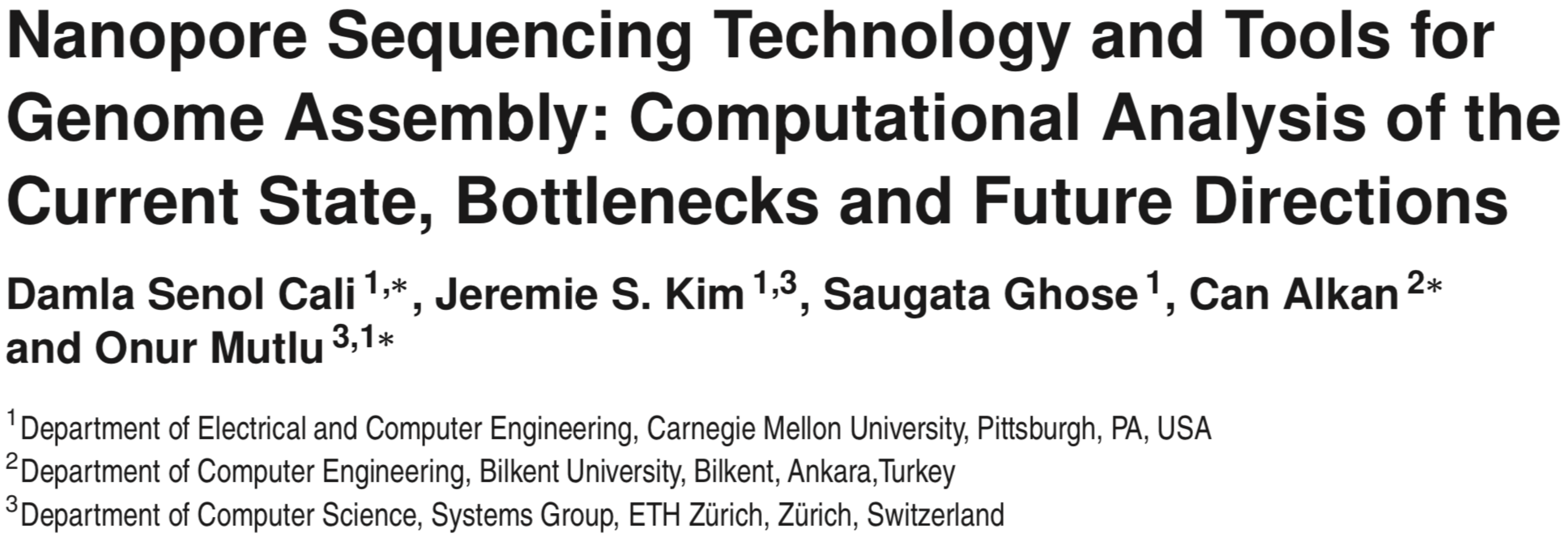
Goal 2:
Memory-efficient

Goal 3:
Scalable/highly-parallel

Nanopore Sequencing & Tools [BiB 2018]

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, April 2018.



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

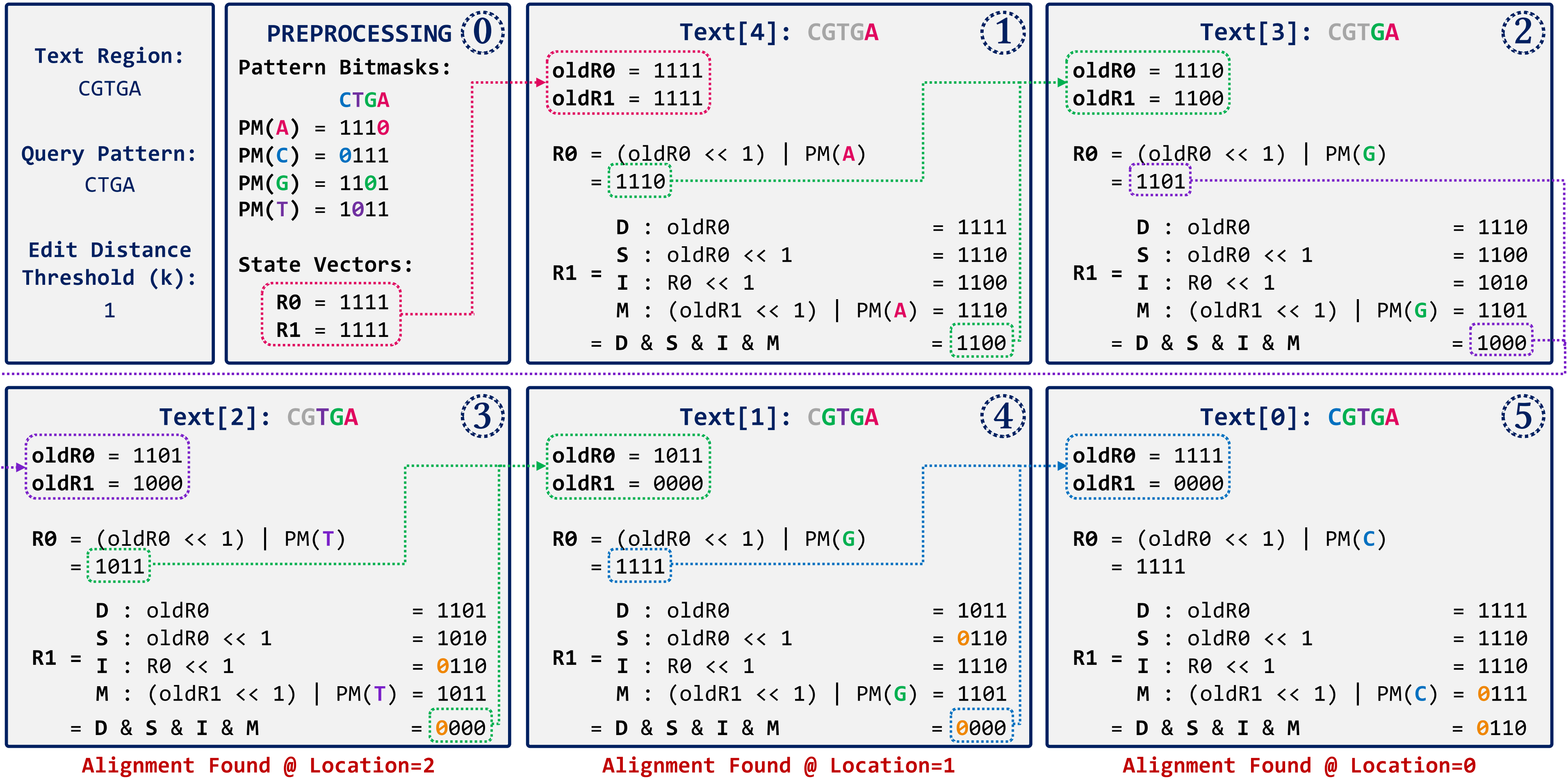
Damla Senol Cali [1,*], Jeremie S. Kim [1,3], Saugata Ghose [1], Can Alkan [2*]
and Onur Mutlu [3,1*]

[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

Backup Slides

(GenASM)

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

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

$\text{Cycle}_{\#}$	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

$\text{Cycle}_{\#}$	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

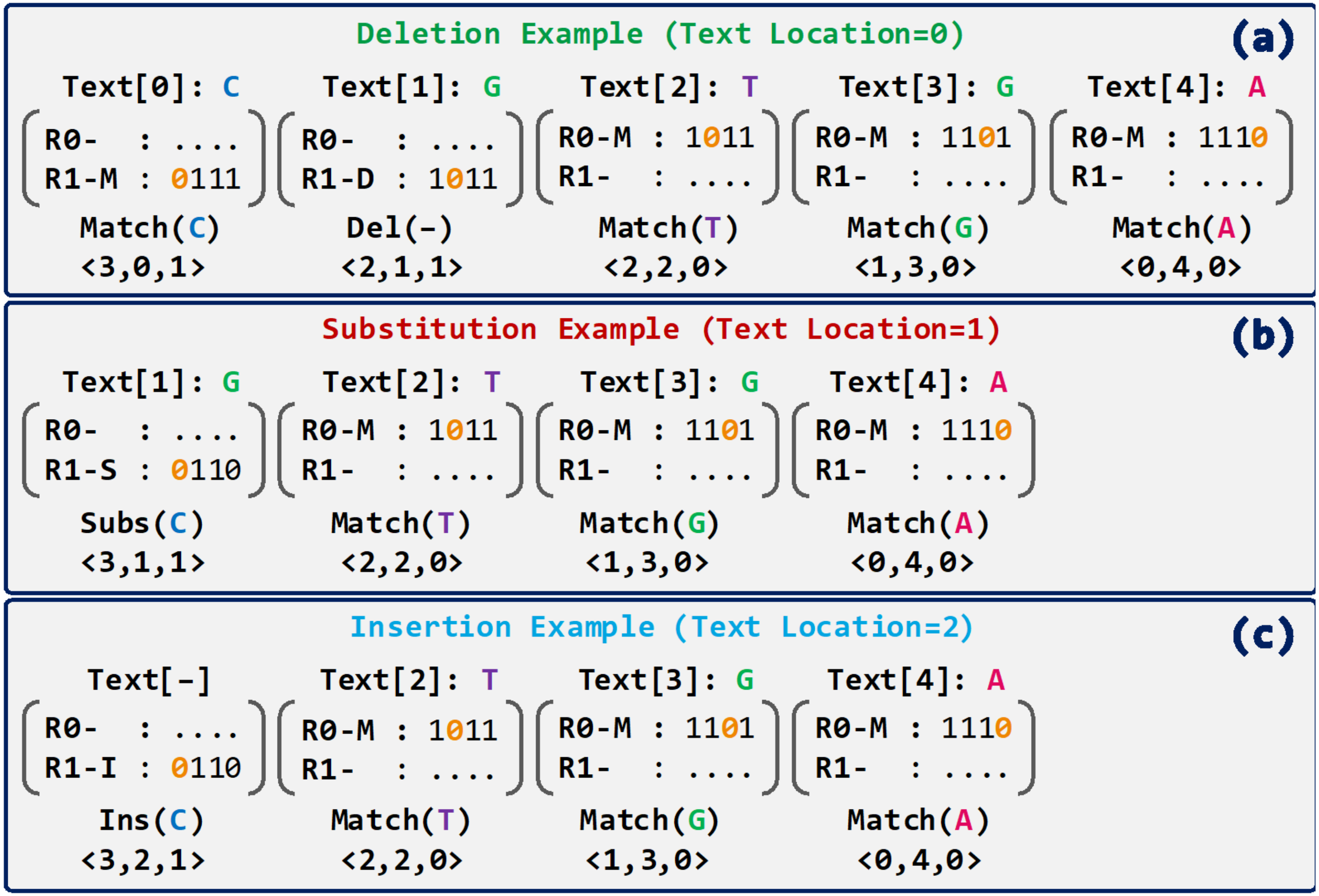
🟦 data *written to memory*

🟩 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       Text[1]: G       Text[2]: T       Text[3]: G       Text[4]: A
( R0-   : .... ) ( R0-   : .... ) ( R0-M : 1011 ) ( R0-M : 1101 ) ( R0-M : 1110 )
( R1-M  : 0111 ) ( R1-D  : 1011 ) ( R1-   : .... ) ( R1-   : .... ) ( R1-   : .... )
   Match(C)         Del(-)          Match(T)         Match(G)         Match(A)
   <3,0,1>          <2,1,1>         <2,2,0>          <1,3,0>          <0,4,0>
```

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

```
  Text[1]: G       Text[2]: T       Text[3]: G       Text[4]: A
( R0-   : .... ) ( R0-M : 1011 ) ( R0-M : 1101 ) ( R0-M : 1110 )
( R1-S  : 0110 ) ( R1-   : .... ) ( R1-   : .... ) ( R1-   : .... )
   Subs(C)         Match(T)         Match(G)         Match(A)
   <3,1,1>         <2,2,0>          <1,3,0>          <0,4,0>
```

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

```
  Text[-]          Text[2]: T       Text[3]: G       Text[4]: A
( R0-   : .... ) ( R0-M : 1011 ) ( R0-M : 1101 ) ( R0-M : 1110 )
( R1-I  : 0110 ) ( R1-   : .... ) ( R1-   : .... ) ( R1-   : .... )
   Ins(C)          Match(T)         Match(G)         Match(A)
   <3,2,1>         <2,2,0>          <1,3,0>          <0,4,0>
```

Key Results – Use Case 1

(1) Read Alignment Step of Read Mapping

- Find the optimal alignment of how reads map to candidate reference regions

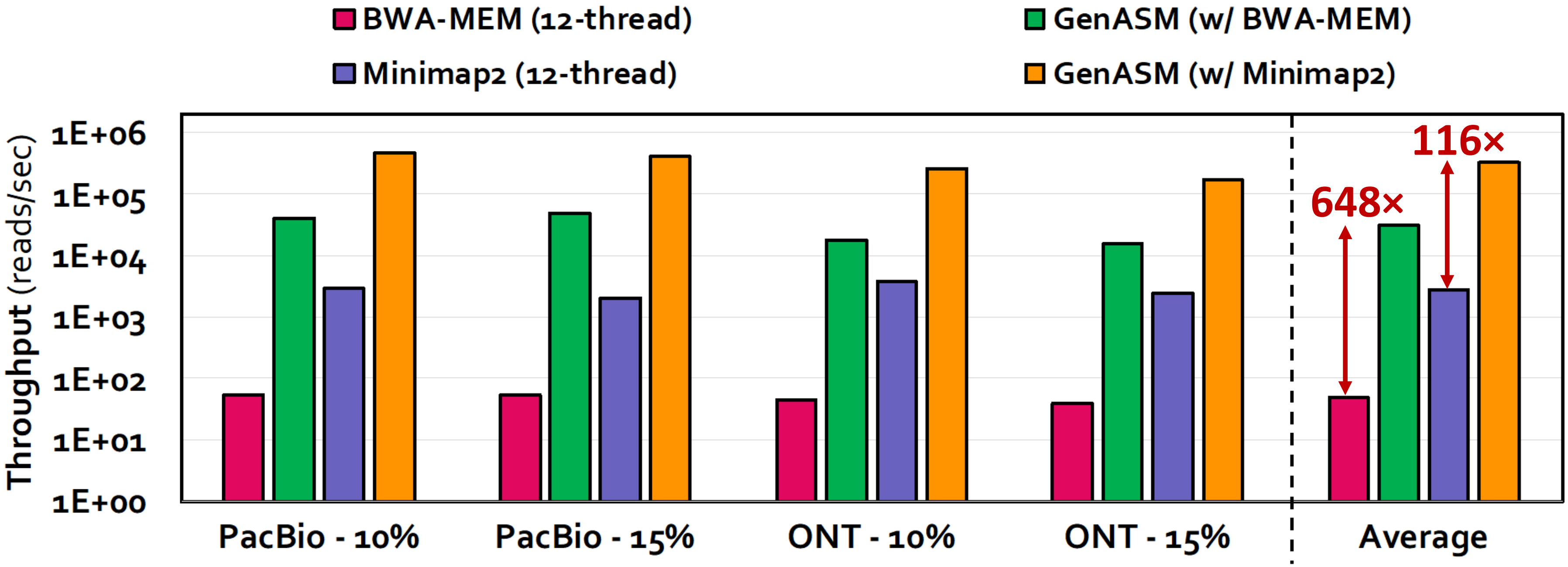
(2) Pre-Alignment Filtering for Short Reads

- 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

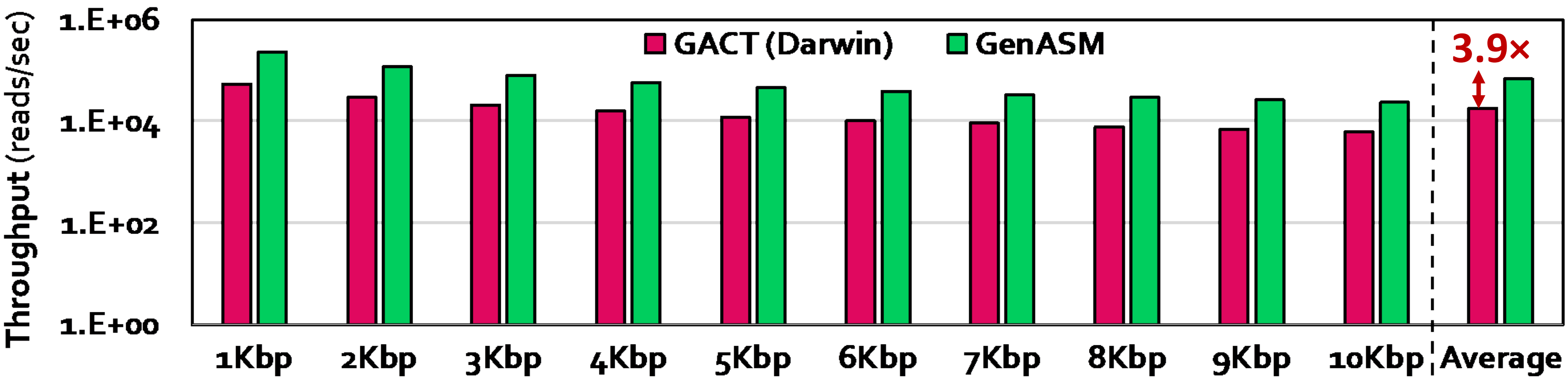
Key Results – Use Case 1 (Long Reads)



SW

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

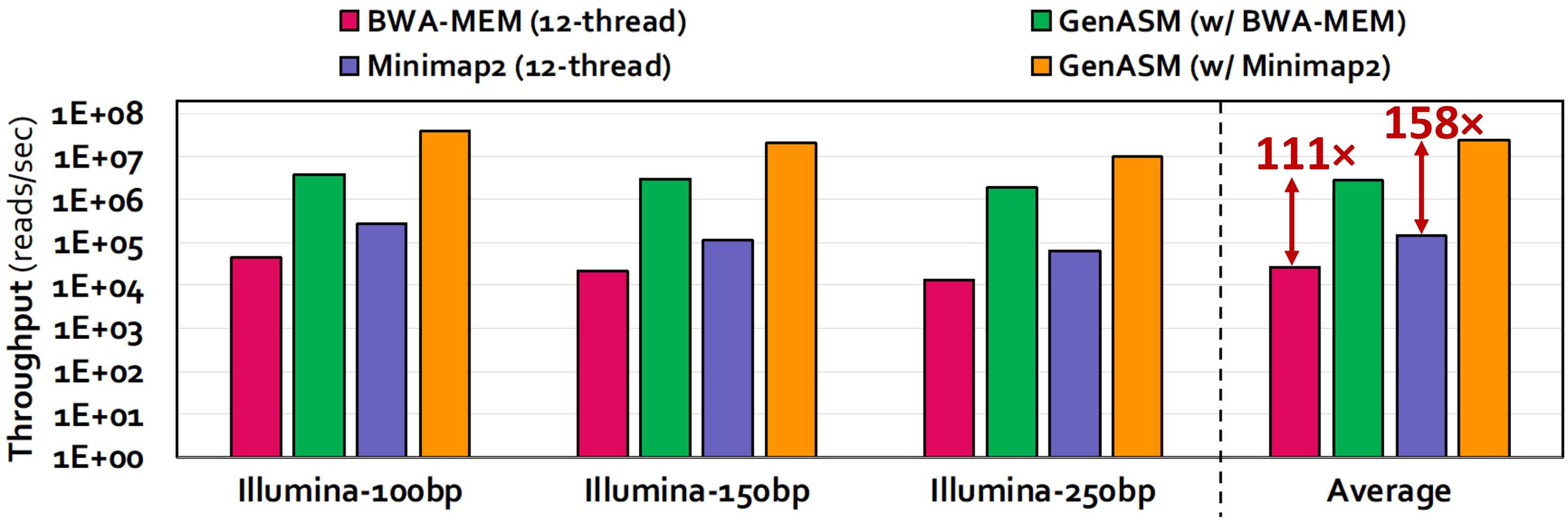
Key Results – Use Case 1 (Long Reads)



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

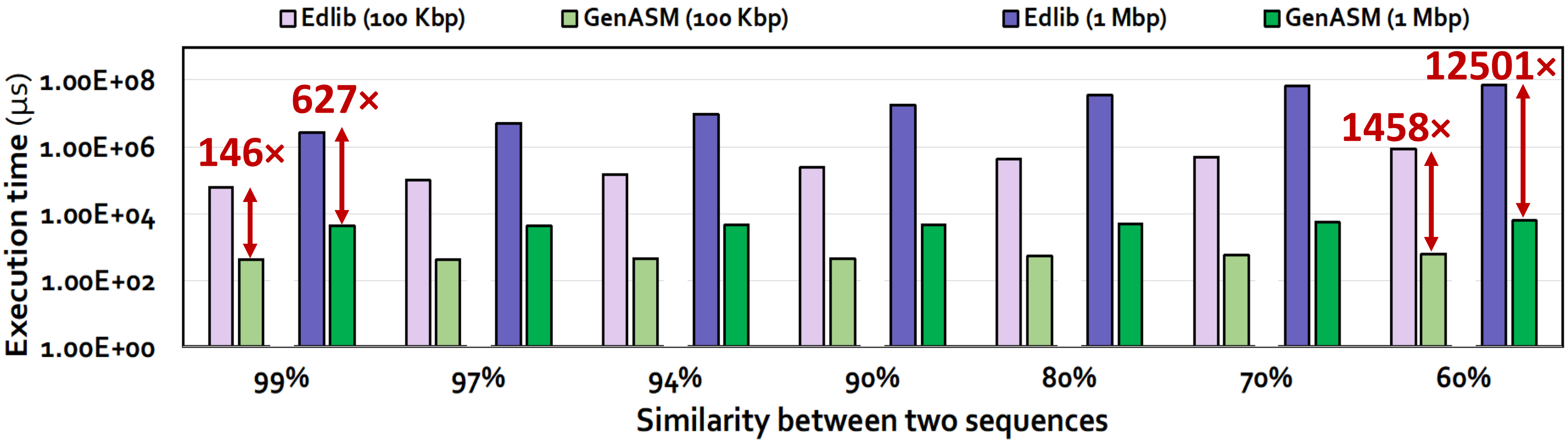
(2) Pre-Alignment Filtering for Short Reads

- ○ 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**

Sources of Improvement in GenASM

- ❑ Very simple computations GenASM performs

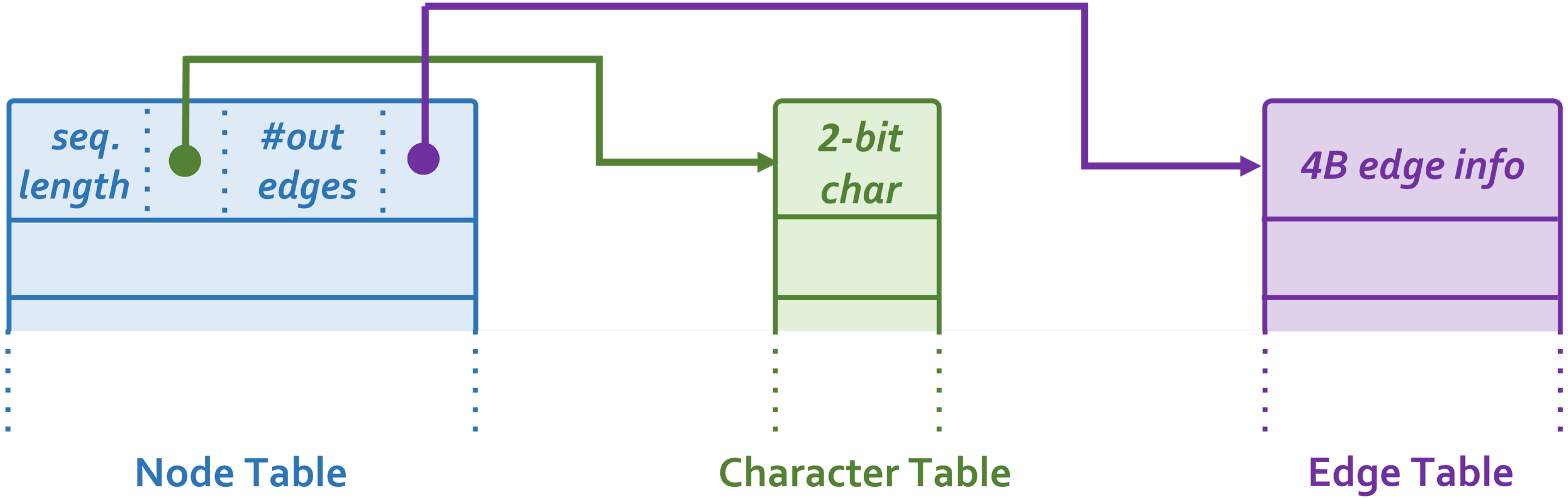
- ❑ Divide-and-conquer approach we follow, which makes our design efficient for both short and long reads despite their different error profiles

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

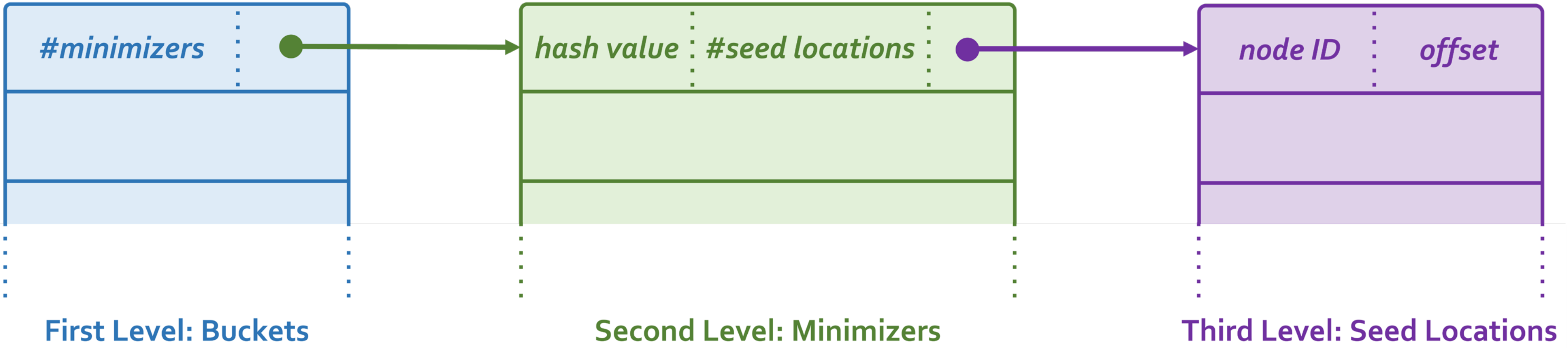
Backup Slides

(SeGraM)

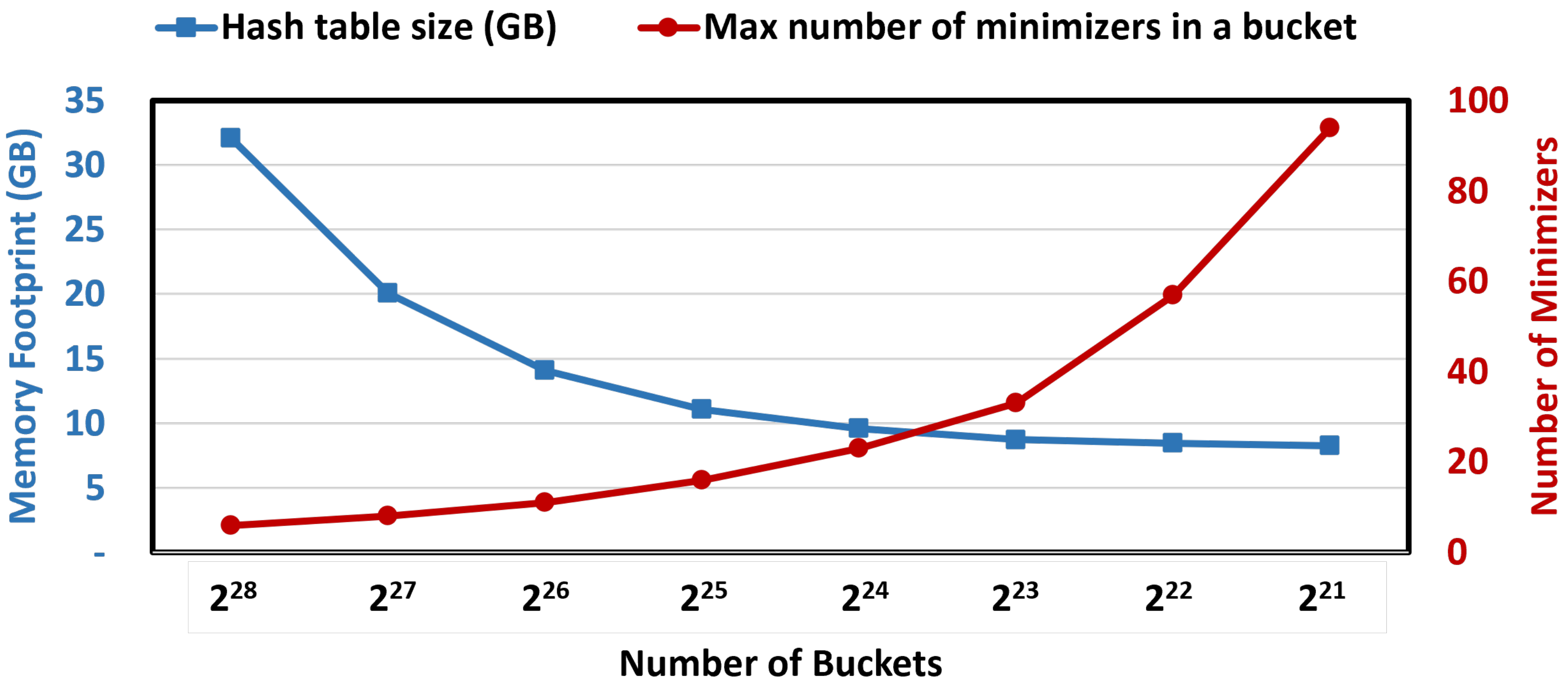
SeGraM – Graph Structure



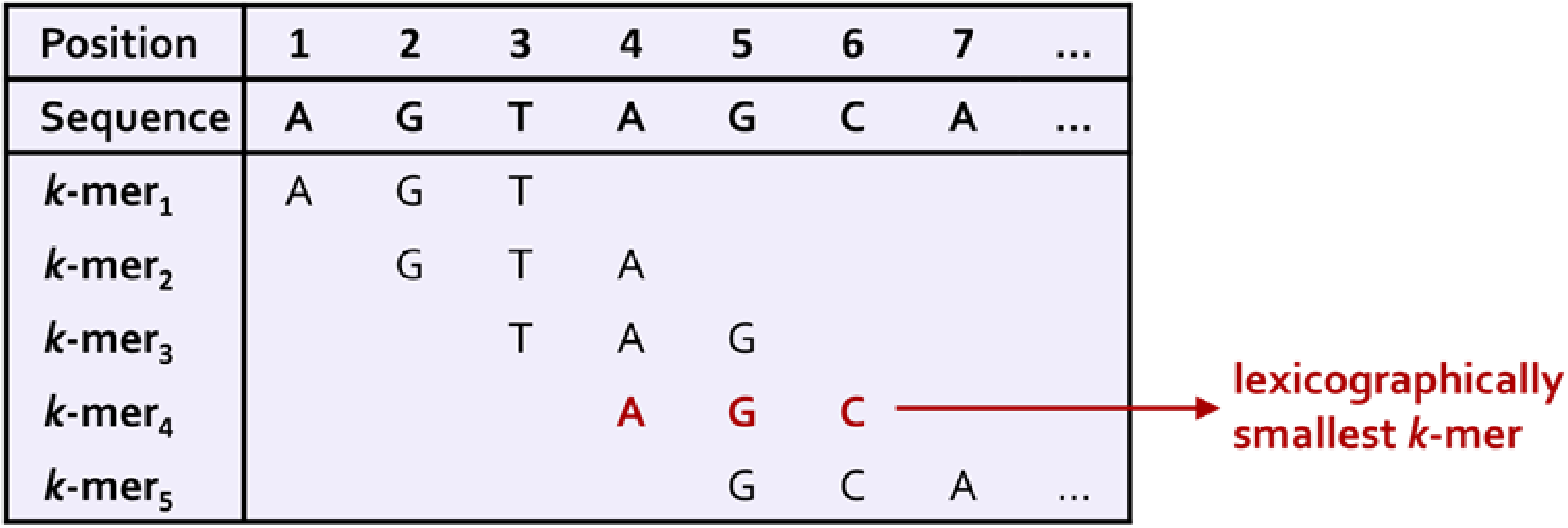
SeGraM – Index Structure



SeGraM – Selection of #Buckets



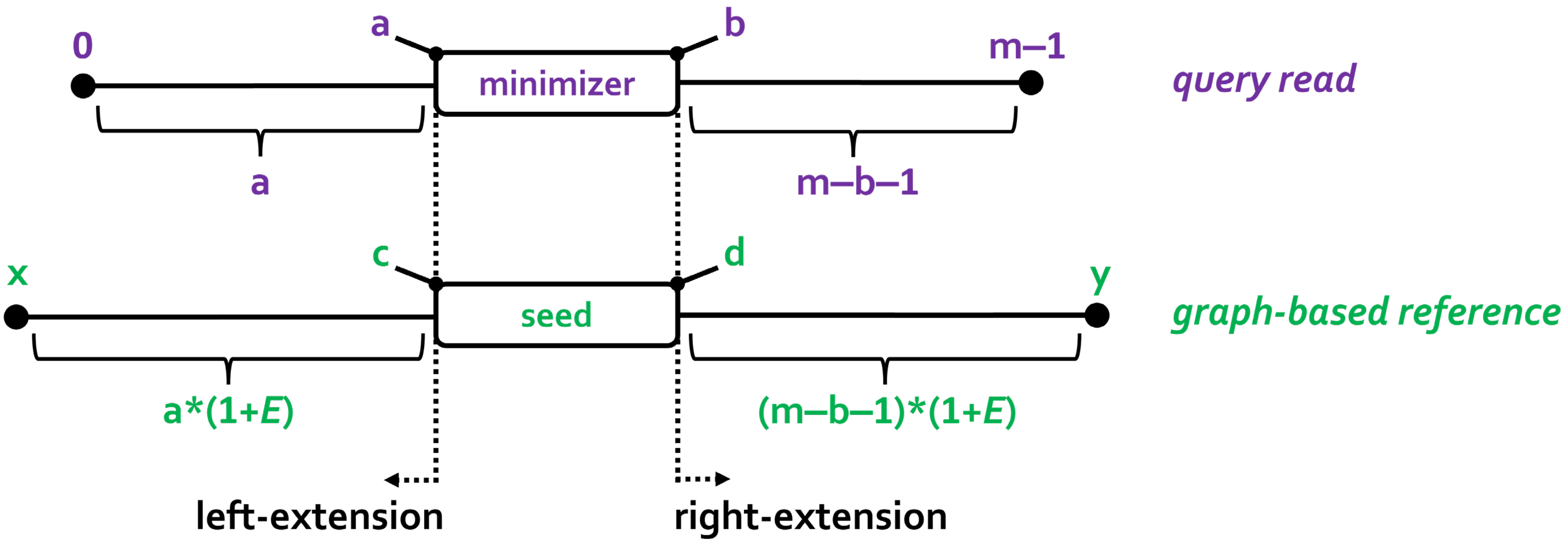
Minimizers



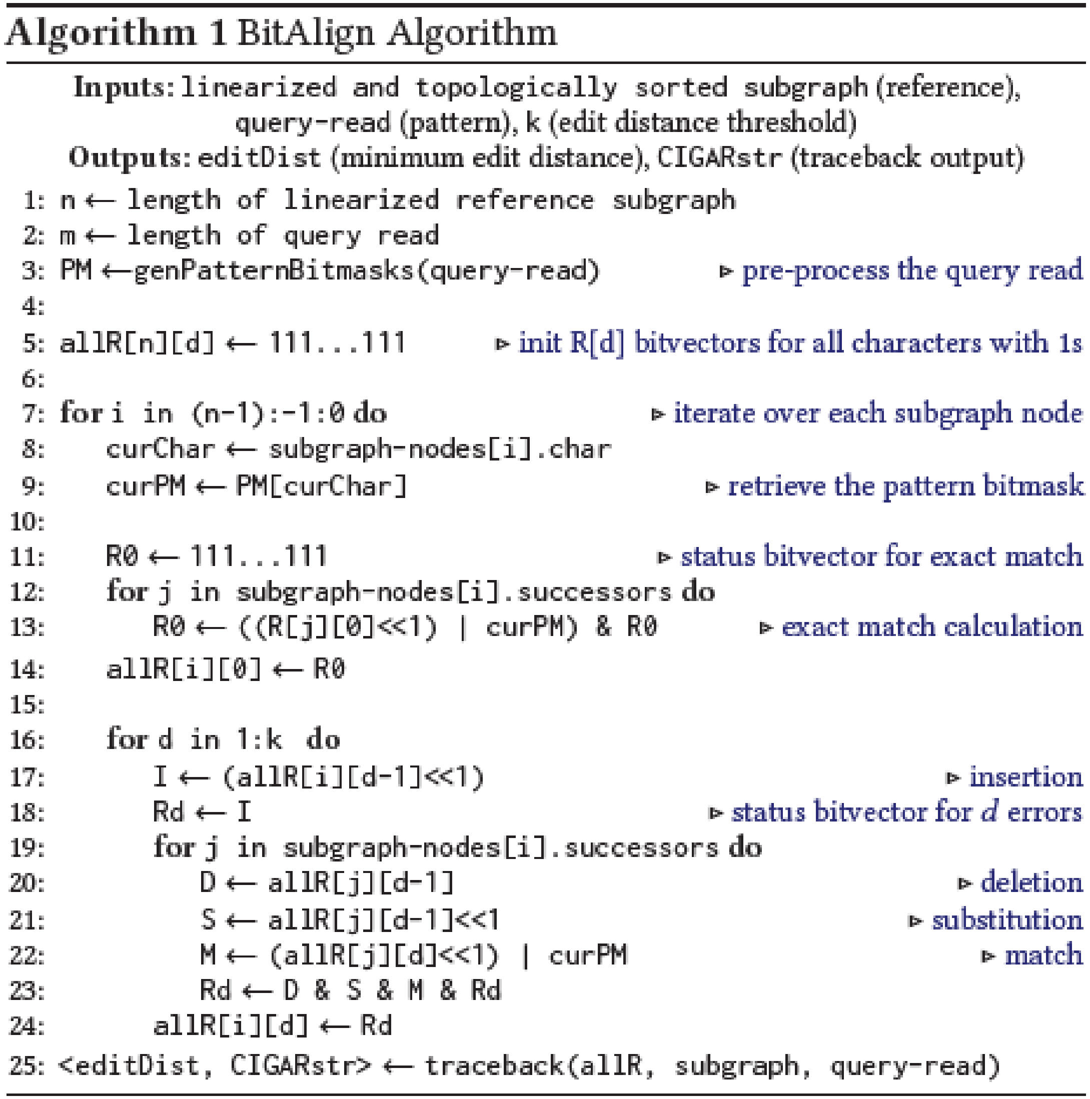
Position	1	2	3	4	5	6	7	...
Sequence	A	G	T	A	G	C	A	...
k-mer$_1$	A	G	T					
k-mer$_2$		G	T	A				
k-mer$_3$			T	A	G			
k-mer$_4$				A	G	C		
k-mer$_5$					G	C	A	...

lexicographically smallest k-mer

MinSeed – Region Calculation



BitAlign Algorithm

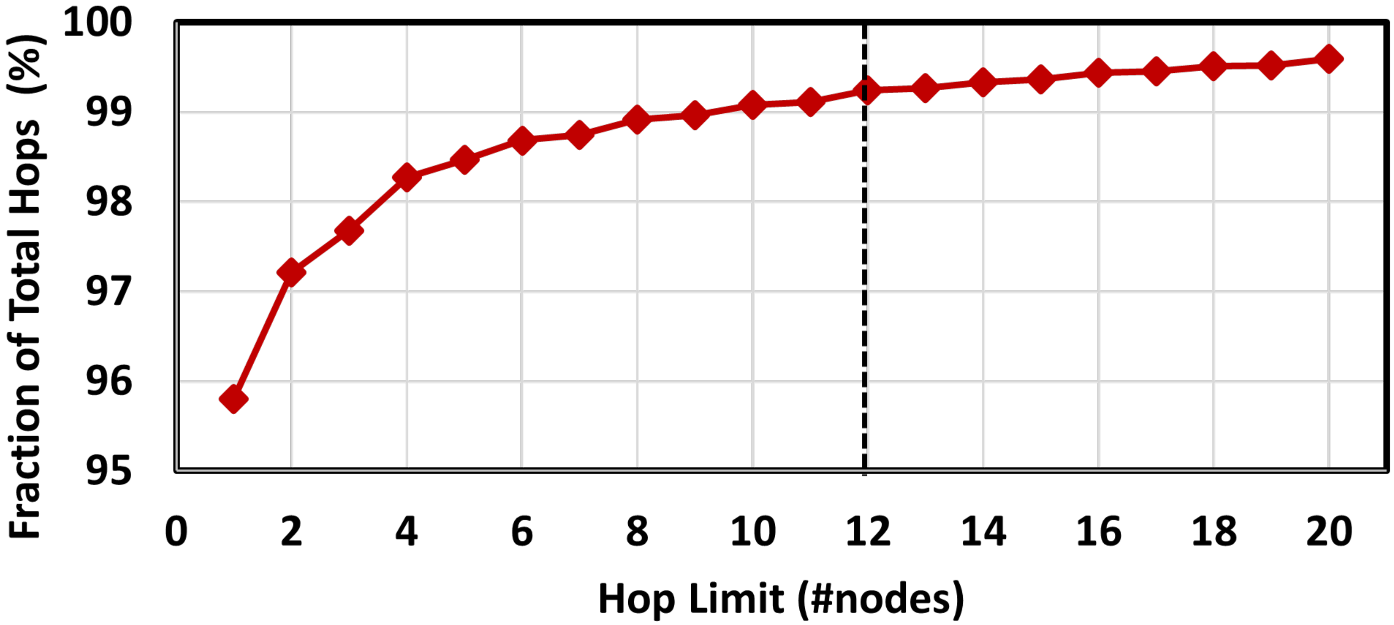


Algorithm 1 BitAlign Algorithm

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

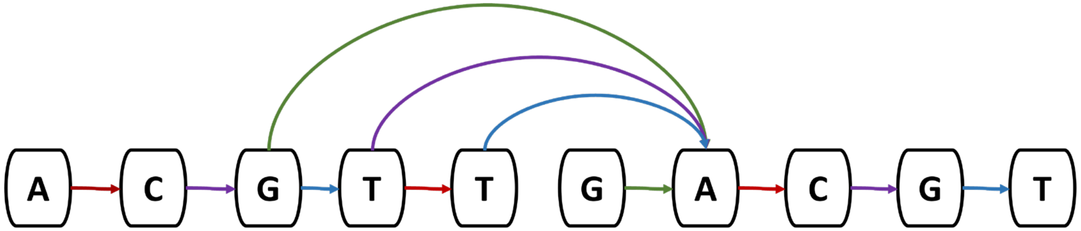
```
n ← length of linearized reference subgraph
m ← length of query read
PM ←genPatternBitmasks(query-read)              ▹ pre-process the query read

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 – Hop Length Selection



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	.

Sources of Improvement

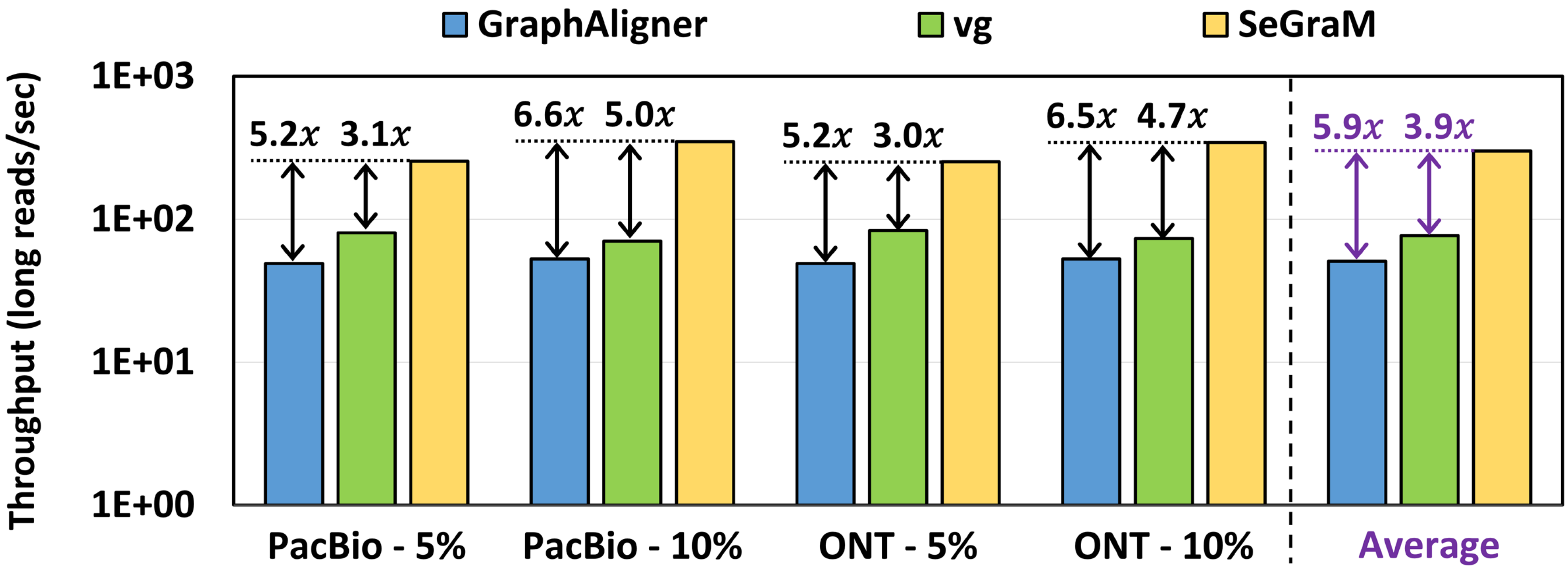
- **Co-design approach for both seeding and alignment:**
  - Efficient and hardware-friendly algorithms for seeding and for alignment
  - Eliminating the data transfer bottleneck between the seeding and alignment steps of the genome sequence analysis pipeline, by placing their individual accelerators (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

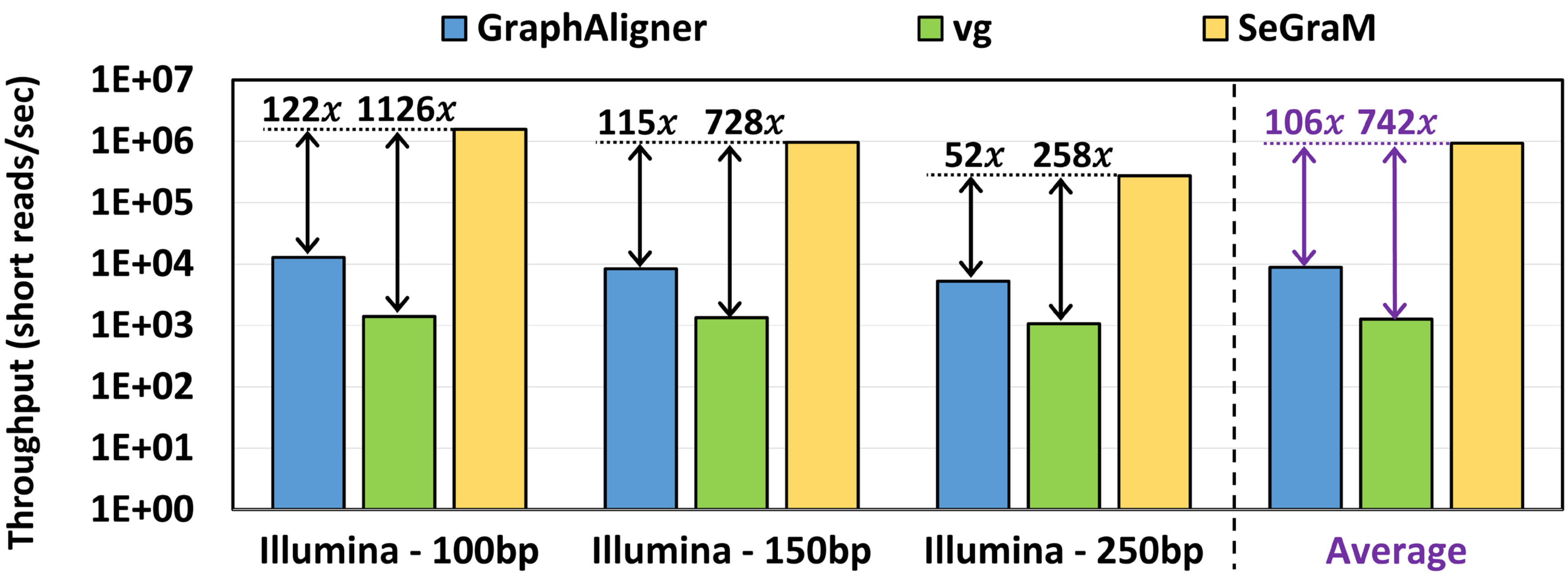
- ❑ **Scaling linearly across three dimensions:**
  - ○ Within a single BitAlign accelerator, by incorporating processing elements *(i.e., iteration-level parallelism)*,
  - ○ Executing multiple seeds in parallel by using pipelined execution with the help of our double buffering approach *(i.e., seed-level 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)*

Key Results – SeGraM with Long Reads



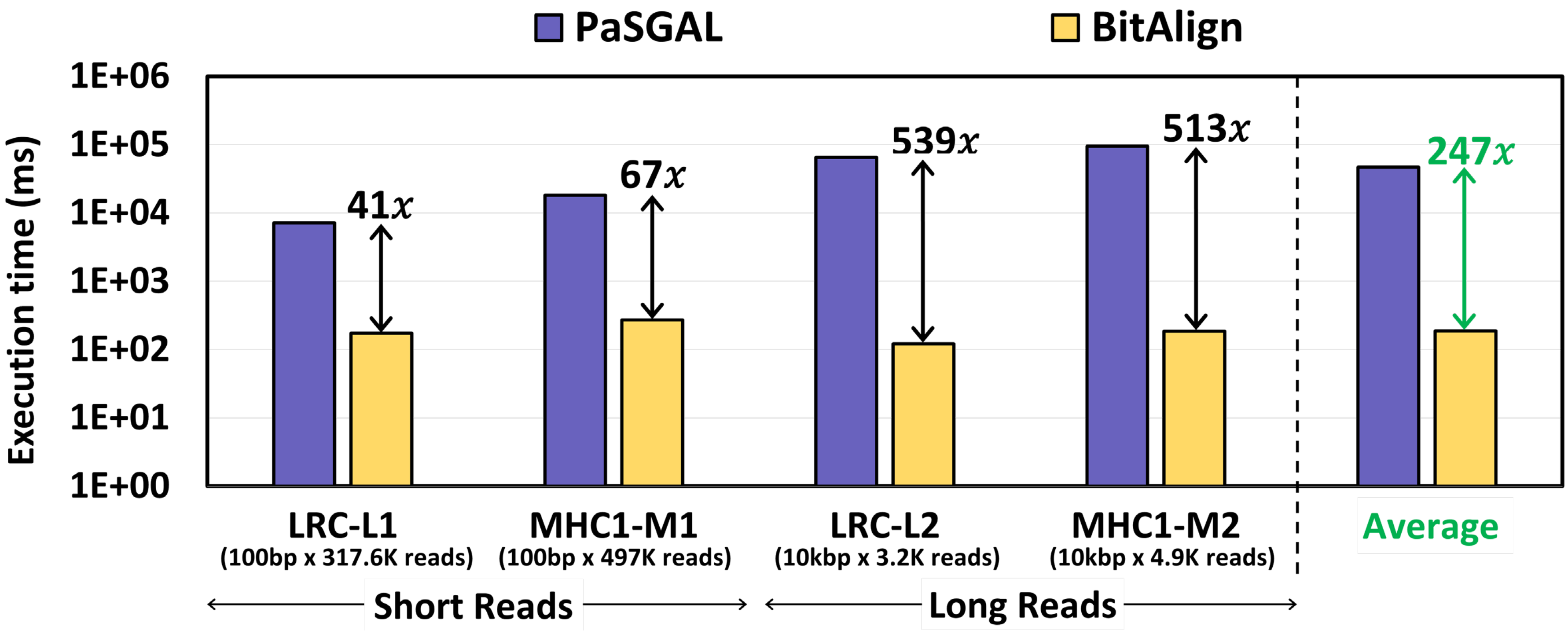
SeGraM provides **5.9× and 3.9× throughput improvement** over GraphAligner and vg, while **reducing the power consumption by 4.1× and 4.4×**

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 be used for both sequence-to-sequence alignment and sequence-to-graph alignment
  - ○ The cost of more functionality: Extra hop queue registers in BitAlign
  - ○ 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

Backup Slides

(BitMAc)

BitMAc: FPGA-based GenASM

Our Goal:

Map GenASM accelerators to an FPGA with HBM2, where **HBM2 offers high memory bandwidth** and **FPGA resources offer high parallelism** by instantiating multiple copies of the GenASM accelerators

- ❑ **Re-modified GenASM algorithms** for a better mapping to the FPGA resources
- ❑ **Intra-level parallelism** by instantiating multiple processing elements (PEs) for the DC execution
- ❑ **Inter-level parallelism** by running multiple independent GenASM executions in parallel

Key Findings

- □ Based on the FPGA resources, **the complete BitMAc design:**
  - ○ 4 BitMAc accelerators connected to each pseudo-channel (128 in total)
  - ○ Each BitMAc accelerator contains a DC accelerator with 16 PEs, a TB accelerator, an FSM, and 13.2KB of M20Ks
  - ○ Clocked at 200MHz

- □ BitMAc provides:
  - ○ **136× – 761× speedup** over the state-of-the-art CPU baselines
  - ○ **6.8× – 19.4× speedup** over the state-of-the-art GPU baseline

Key Findings (cont'd.)

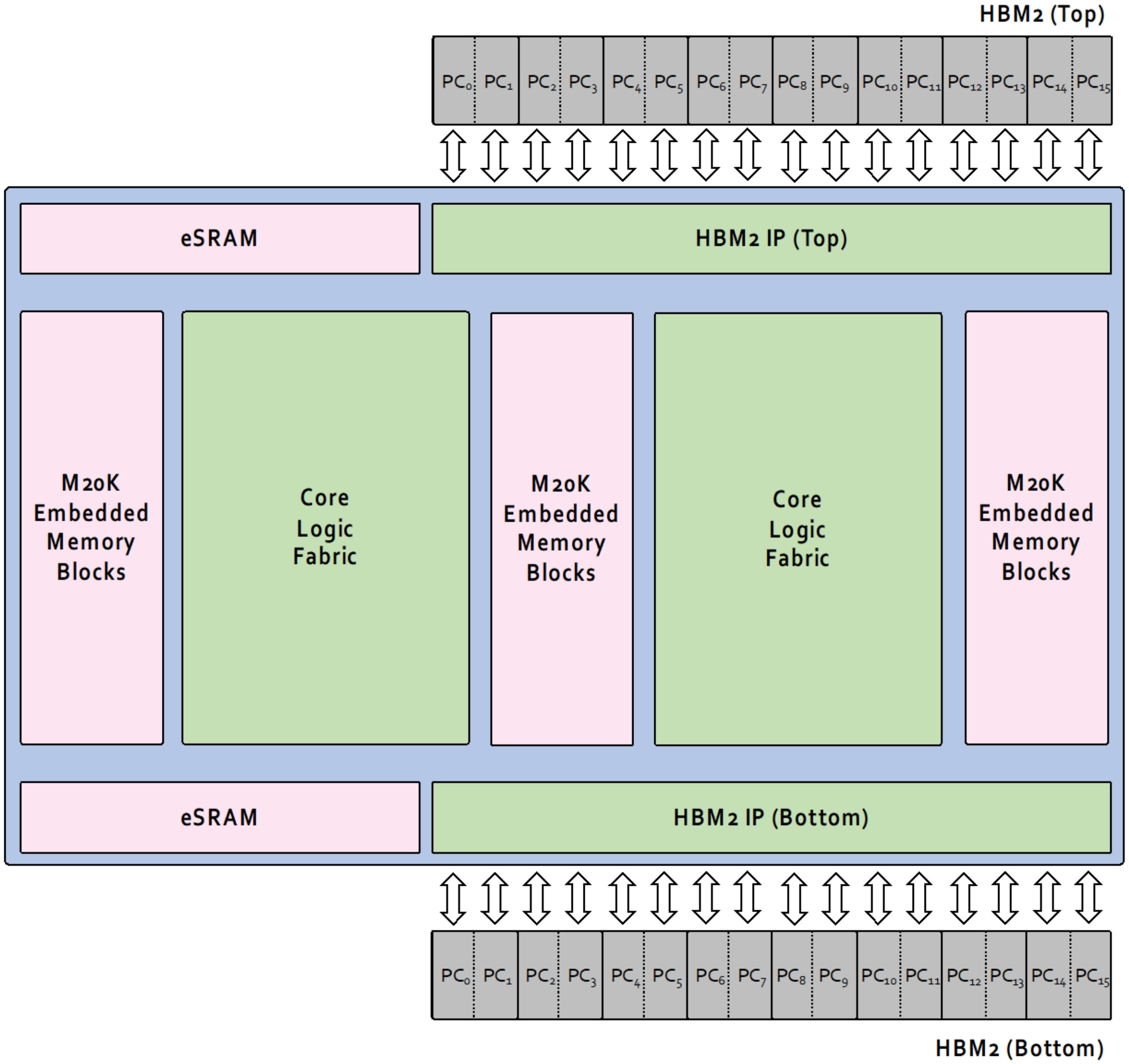
- ❑ BitMAc has:
  - ○ **64% logic utilization** and **90% on-chip memory utilization**
  - ○ Total power consumption of **48.9W**, where **59% accounts for the M20Ks**

- ❑ **Bottlenecked by the amount of on-chip memory** (i.e., M20Ks)

- ❑ **Cannot saturate the high bandwidth** that multiple HBM2 stacks on the FPGA provide

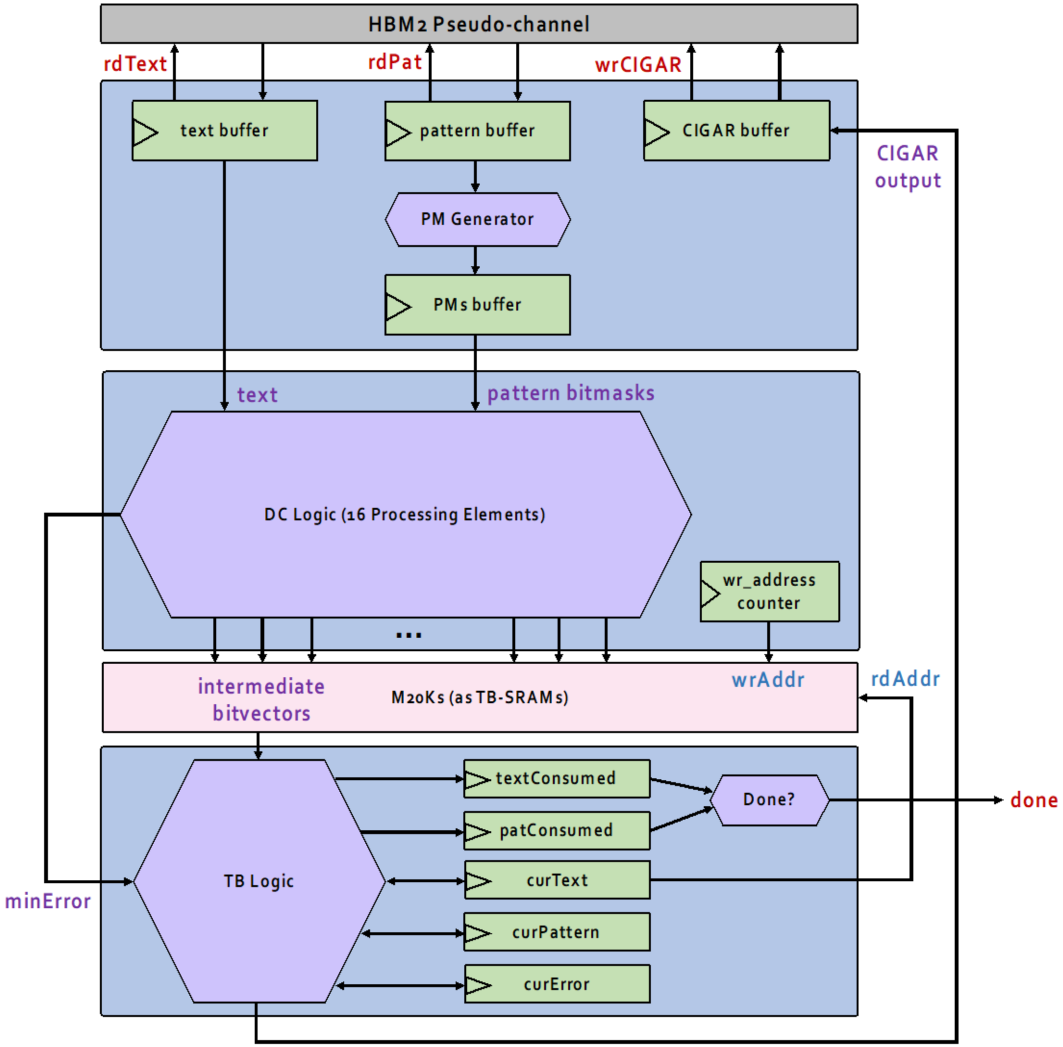
- ❑ Need (1) **algorithm-level modifications** to decrease the amount of data that need to be stored in M20Ks, and (2) **newer FPGA chips that provide a higher amount of on-chip memory capacity**

Intel Stratix 10 MX



HBM2 (Top)
PC0 PC1 PC2 PC3 PC4 PC5 PC6 PC7 PC8 PC9 PC10 PC11 PC12 PC13 PC14 PC15
eSRAM
HBM2 IP (Top)
M20K Embedded Memory Blocks
Core Logic Fabric
M20K Embedded Memory Blocks
Core Logic Fabric
M20K Embedded Memory Blocks
eSRAM
HBM2 IP (Bottom)
PC0 PC1 PC2 PC3 PC4 PC5 PC6 PC7 PC8 PC9 PC10 PC11 PC12 PC13 PC14 PC15
HBM2 (Bottom)

BitMAc Design



BitMAc – Results

Component	Dynamic On-Chip Power Dissipation	Total On-Chip Power Dissipation
DC Logic (16 PEs)	128.57 mW	
TB Logic	10.24 mW	
FSM Logic	3.15 mW	
M20Ks	211.61 mW	
Other	15.72 mW	
Total – 1 BitMAc Accelerator	**369.29 mW (0.4 W)**	**6043.24 mW (6.0 W)**
Total – 32 BitMAc Accelerators (1 per each pseudo-channel)	**11569.92 mW (11.6 W)**	**17234.67 mW (17.2 W)**
Total – 128 BitMAc Accelerators (4 per each pseudo-channel)	**43042.90 mW (43 W)**	**48935.65 mW (48.9 W)**

BitMAc – Results

Configuration	Logic Utilization	M20K	eSRAM	DSP
1 BitMAc Accelerator	0.5%	0.7%	0%	0%
32 BitMAc Accelerators (1 per each pseudo-channel)	17.7%	22.4%	0%	0%
128 BitMAc Accelerators (4 per each pseudo-channel)	64.3%	89.7%	0%	0%