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

Damla Senol Cali

Carnegie Mellon University

(dsenol@andrew.cmu.edu)

Gurpreet S. Kalsi[2], Zulal Bingol[3], Can Firtina[4], Lavanya Subramanian[5], Jeremie Kim[1,4], Rachata Ausavarungnirun[6,1], Mohammed Alser[4], Juan Gomez-Luna[4], Amirali Boroumand[1], Anant Nori[2], Allison Scibisz[1], Sreenivas Subramoney[2], Can Alkan[3], Saugata Ghose[7,1], and Onur Mutlu[4,1,3]

[1] Carnegie Mellon [2] intel [3] Bilkent University [4] ETH zürich

[5] facebook [6] King Mongkut's University of Technology North Bangkok [7] University of Illinois Urbana-Champaign [1,4] SAFARI

Genome Sequencing

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

Genome

DNA

- 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

- **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

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
  - Based upon the *Bitap* algorithm
    - Uses fast and simple bitwise operations to perform ASM
  - Modified and extended ASM algorithm
    - Highly-parallel Bitap with long read support
    - Novel bitvector-based algorithm to perform *traceback*
  - Co-design of our modified scalable and memory-efficient algorithms with low-power and area-efficient hardware accelerators

Use Cases & 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**)

(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**)

Outline

- Introduction
- **Background**
  - **Approximate String Matching (ASM)**
  - **ASM with Bitap Algorithm**
- GenASM: ASM Acceleration Framework
  - GenASM Algorithm
  - GenASM Hardware Design
  - Use Cases of GenASM
- Evaluation
- Conclusion

Approximate String Matching

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

- **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):

Large number of iterations

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.

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]

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

R[d] = deletion & mismatch & insertion & match

Check MSB of R[d]:

If 1, no match.

If 0, match with *d* many errors.

Bitap Algorithm (cont'd.)

- **Step 2: Edit Distance Calculation**

For each character of the text (char):

Copy previous R bitvectors as oldR

R[0] = (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]

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

R[d] = deletion & mismatch & insertion & match

Check MSB of R[d]:

If 1, no match.

If 0, match with *d* many errors.

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

Outline

- Introduction
- Background
  - Approximate String Matching (ASM)
  - ASM with Bitap Algorithm
- **GenASM: ASM Acceleration Framework**
  - **GenASM Algorithm**
  - **GenASM Hardware Design**
  - **Use Cases of GenASM**
- Evaluation
- Conclusion

GenASM: ASM Framework for GSA

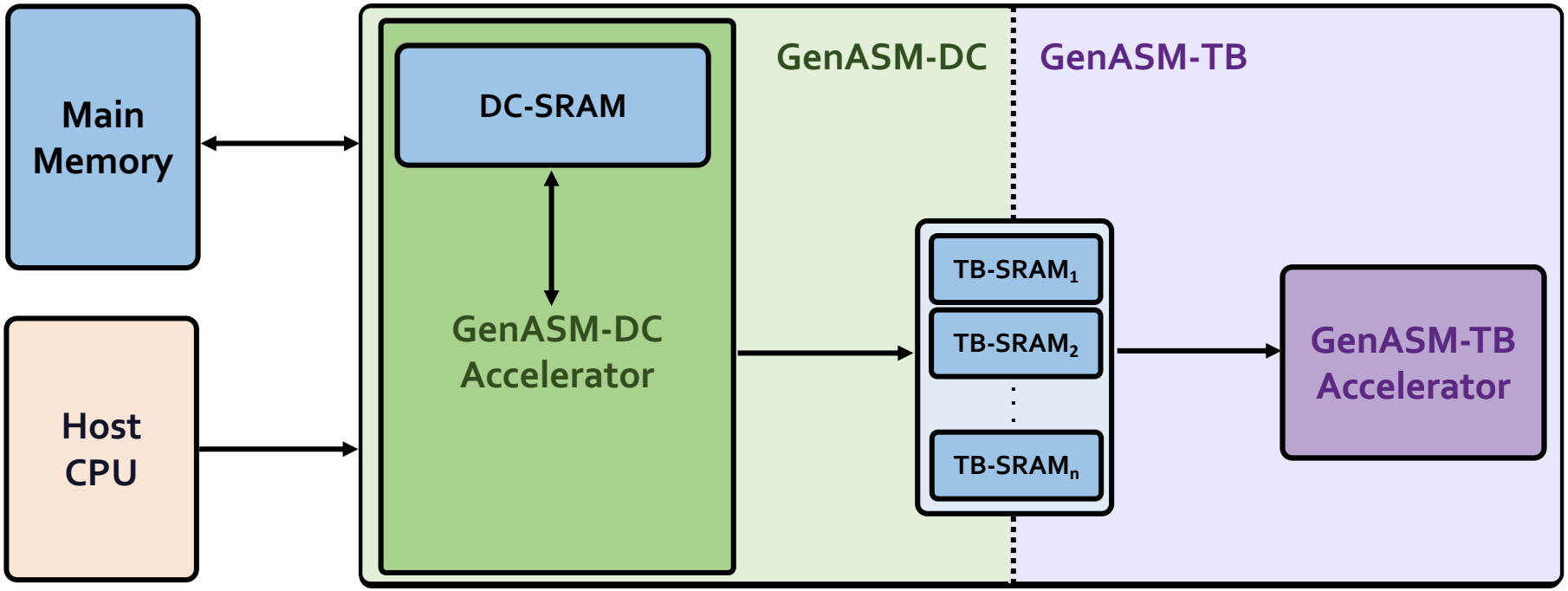
- Approximate string matching (ASM) acceleration framework based on the Bitap algorithm
- *First* ASM acceleration framework for genome sequence analysis
- We overcome the five limitations that hinder Bitap's use in genome sequence analysis:
  - Modified and extended ASM algorithm
    - Highly-parallel 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

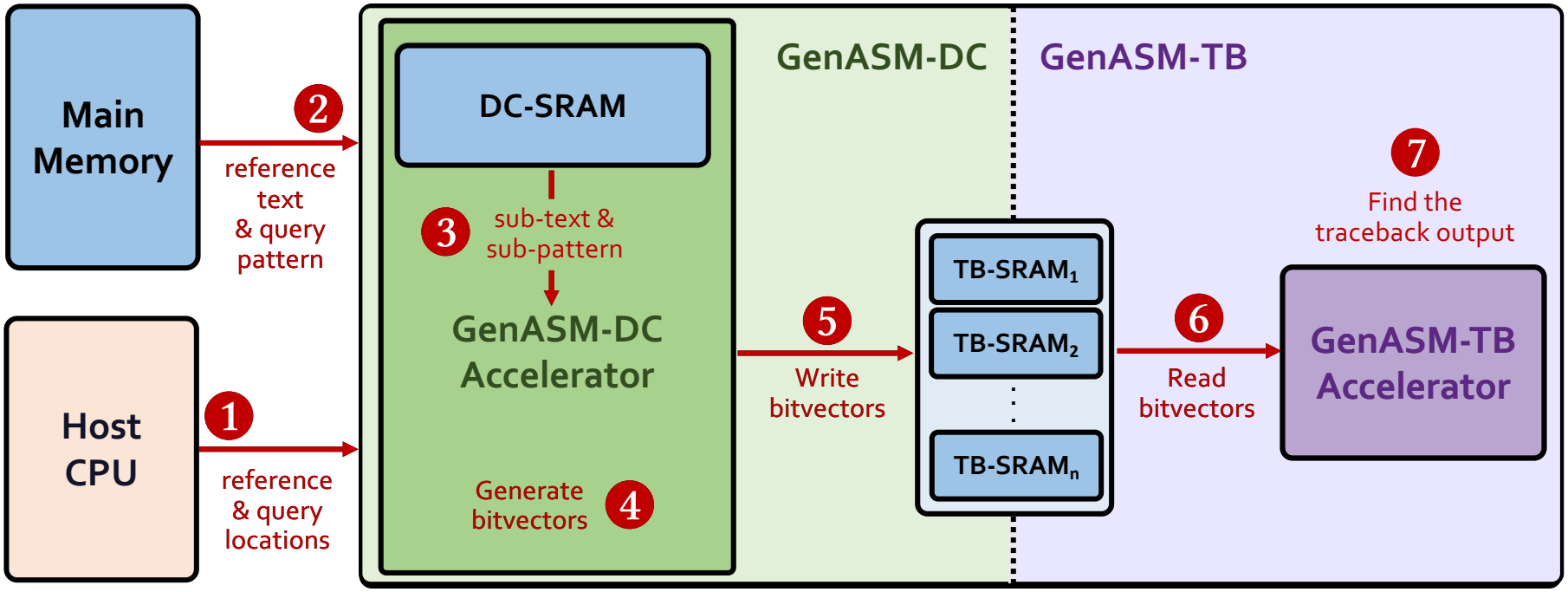
GenASM Hardware Design



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

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

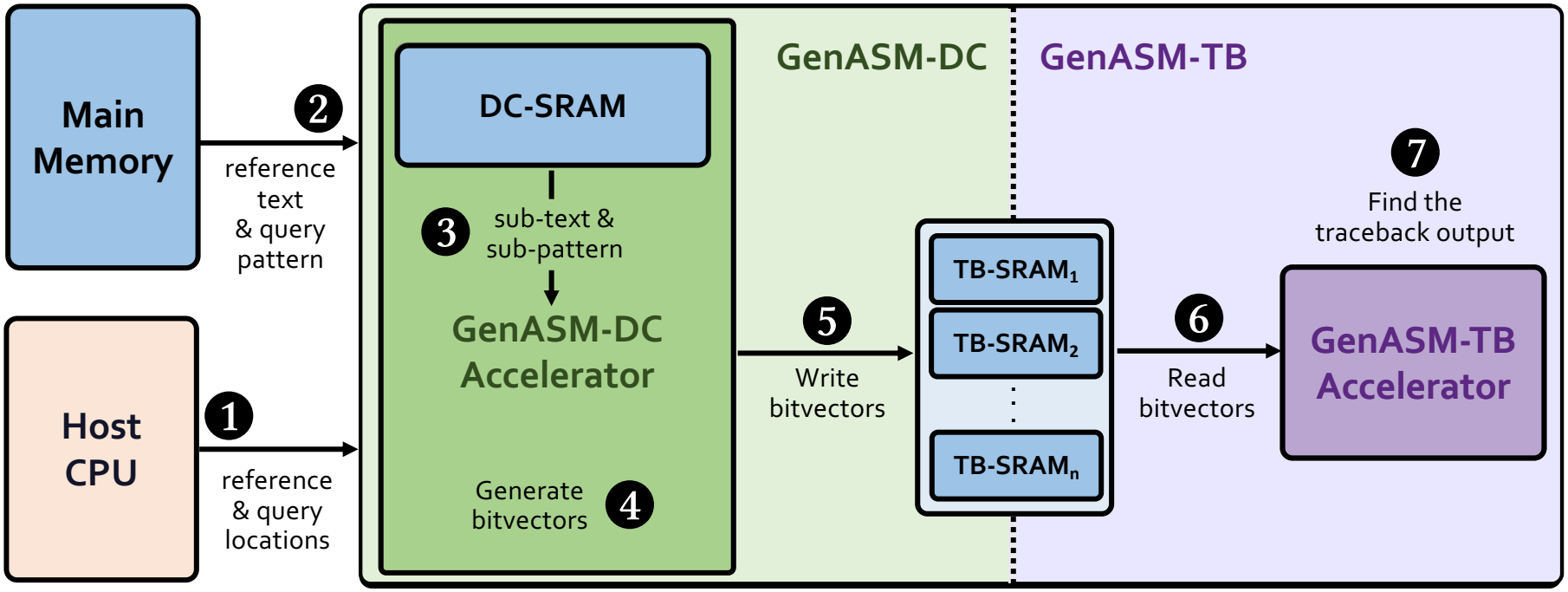
GenASM Hardware Design



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

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

GenASM Hardware Design



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

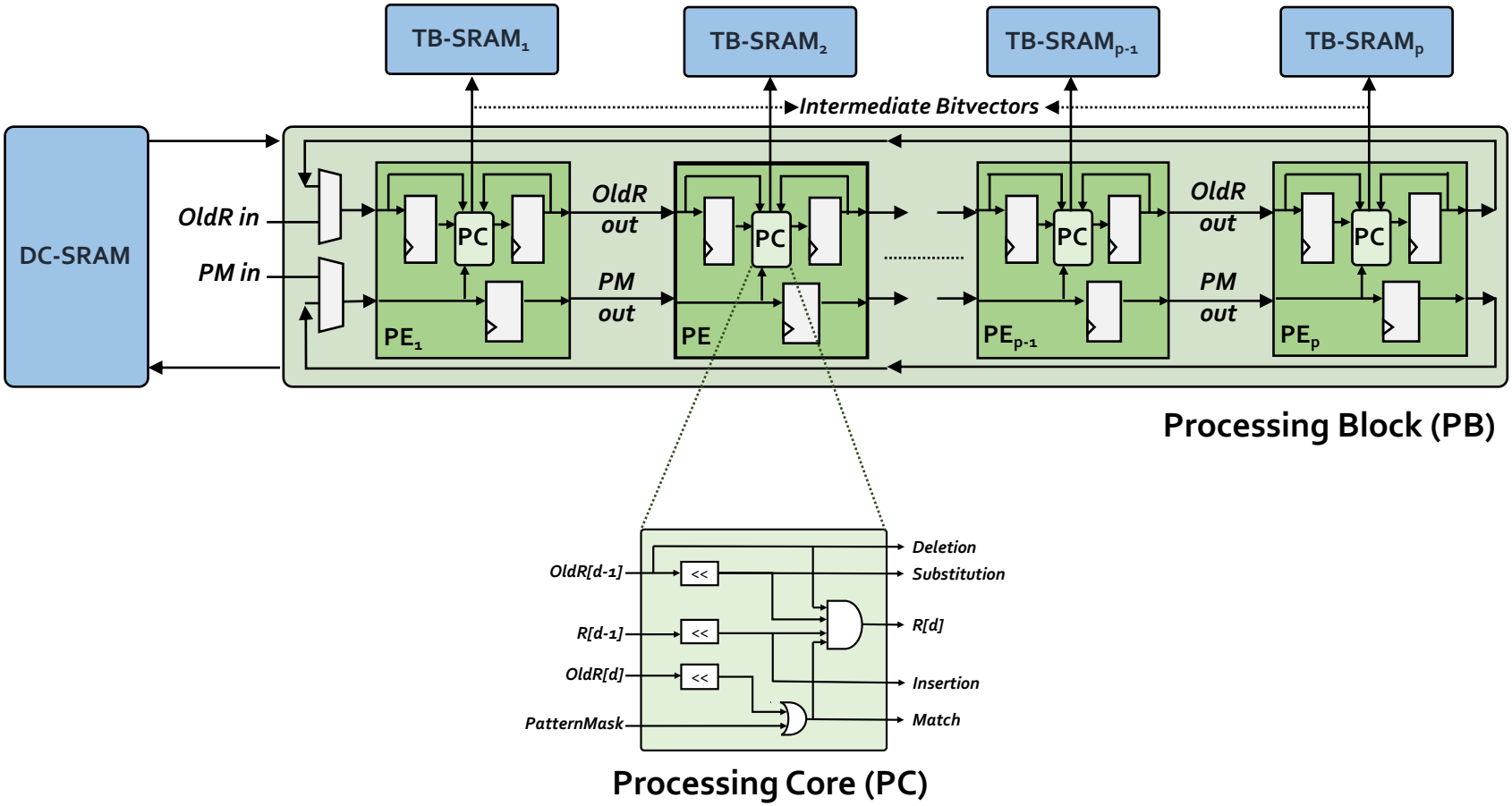
→ Match the rate of computation with memory capacity and bandwidth

→ Achieve high performance and power efficiency

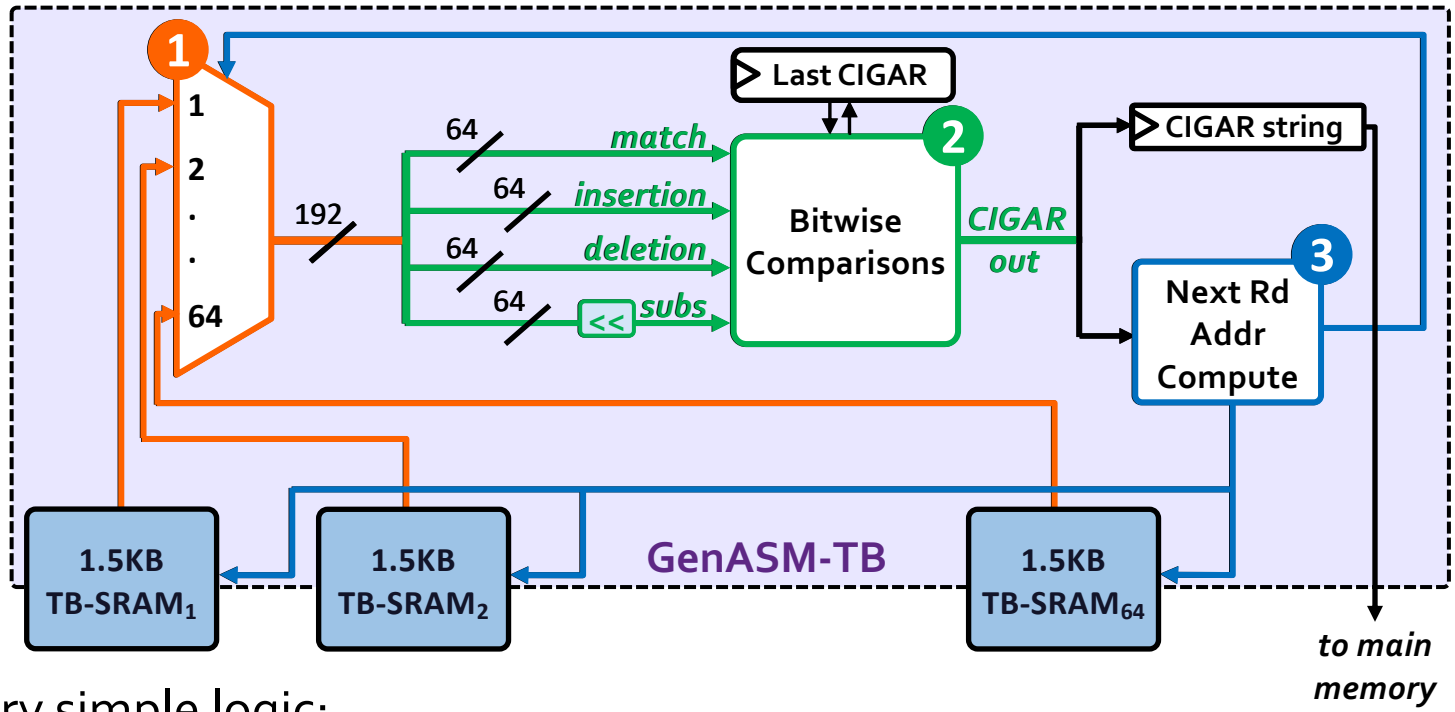
→ Scale linearly in performance with the number of parallel compute units that we add to the system

GenASM-DC: Hardware Design

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



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

Outline

- Introduction
- Background
  - Approximate String Matching (ASM)
  - ASM with Bitap Algorithm
- GenASM: ASM Acceleration Framework
  - GenASM Algorithm
  - GenASM Hardware Design
  - Use Cases of GenASM
- **Evaluation**
- Conclusion

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

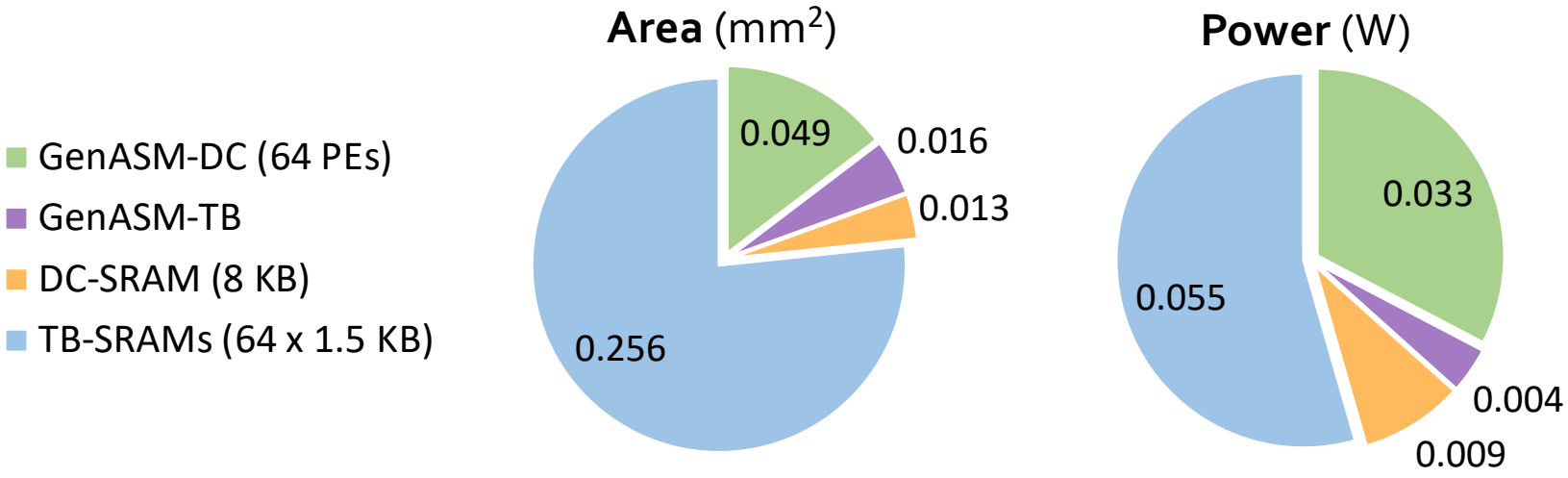
Evaluation Methodology (cont'd.)

- **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

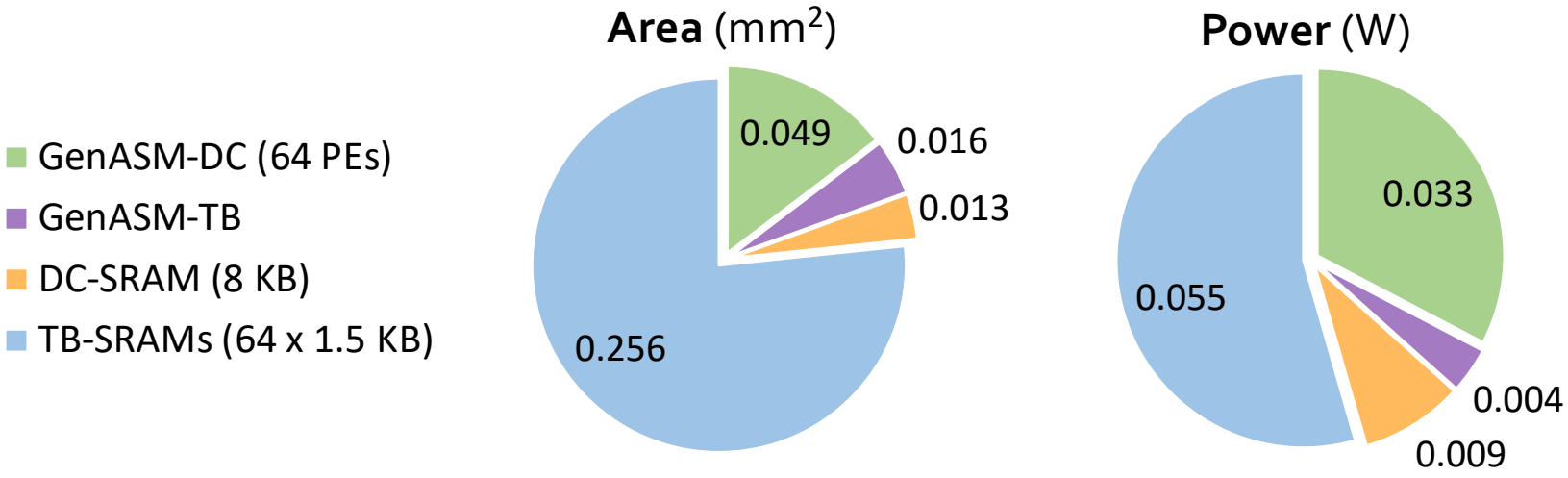
- Based on our **synthesis** of **GenASM-DC** and **GenASM-TB** accelerator datapaths using the Synopsys Design Compiler with a **28nm** LP process:
  - Both GenASM-DC and GenASM-TB operate **@ 1GHz**



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

Key Results – Area and Power

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



GenASM has low area and power overheads

Key Results – Use Case 1

(1) Read Alignment Step of Read Mapping

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

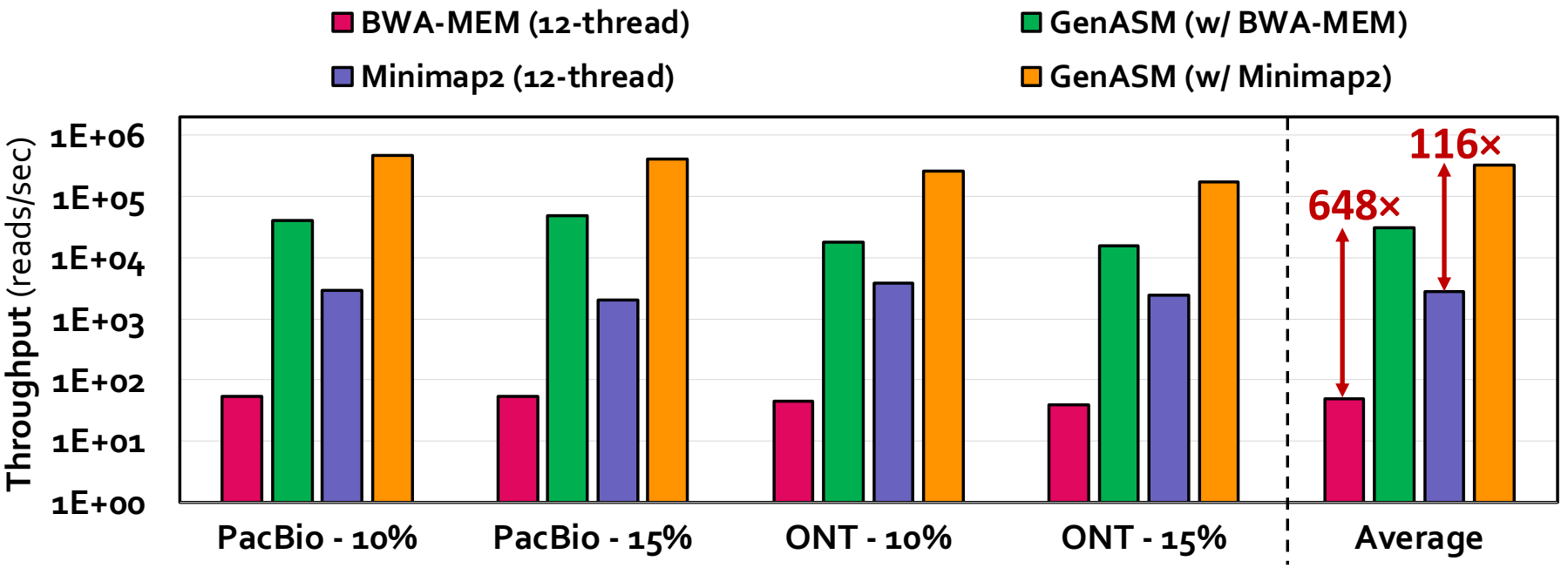
(2) Pre-Alignment Filtering for Short Reads

- Quickly identify and filter out the unlikely candidate reference regions for each read

(3) Edit Distance Calculation

- Measure the similarity or distance between two sequences

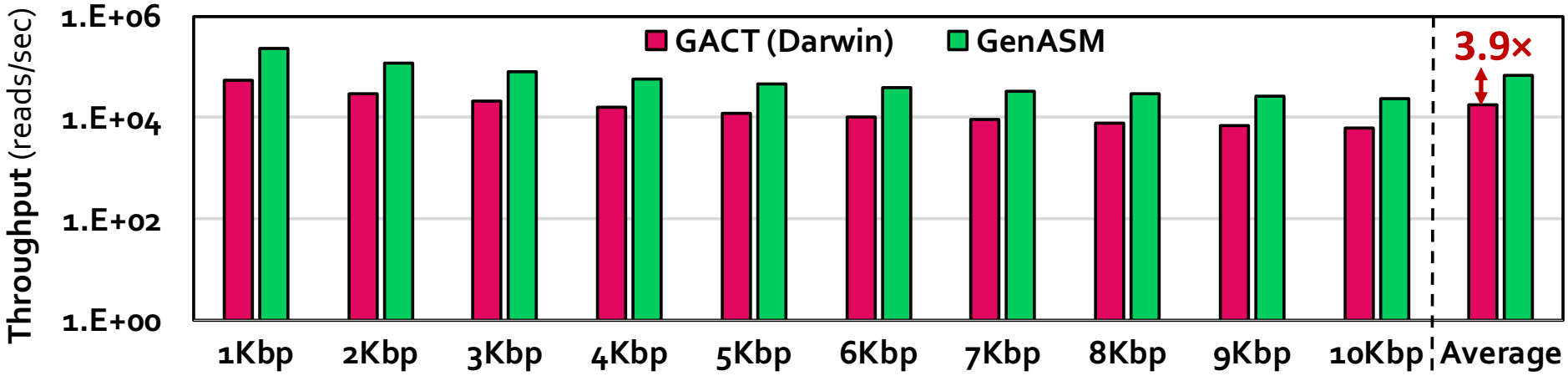
Key Results – Use Case 1 (Long Reads)



SW

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

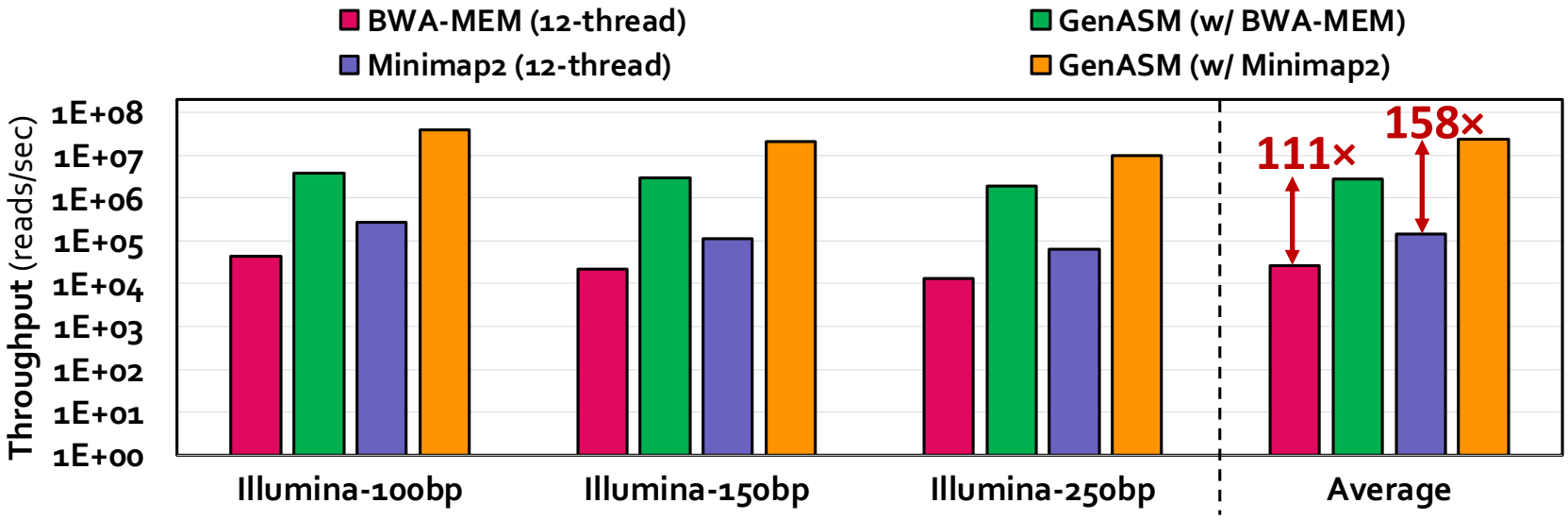
Key Results – Use Case 1 (Long Reads)



HW

GenASM provides **3.9× better throughput**,
6.6× the throughput per unit area, and
10.5× the throughput per unit power,
compared to GACT of Darwin

Key Results – Use Case 1 (Short Reads)



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

HW GenASM provides **1.9× better throughput** and uses **63% less logic area** and **82% less logic power**, compared to SillaX of GenAx

Key Results – Use Case 2

(1) Read Alignment Step of Read Mapping

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

(2) Pre-Alignment Filtering for Short Reads

- Quickly identify and filter out the unlikely candidate reference regions for each read

(3) Edit Distance Calculation

- Measure the similarity or distance between two sequences

Key Results – Use Case 2

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

HW

GenASM is **more efficient in terms of both speed and power consumption**, while **significantly improving the accuracy** of pre-alignment filtering

Key Results – Use Case 3

(1) Read Alignment Step of Read Mapping

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

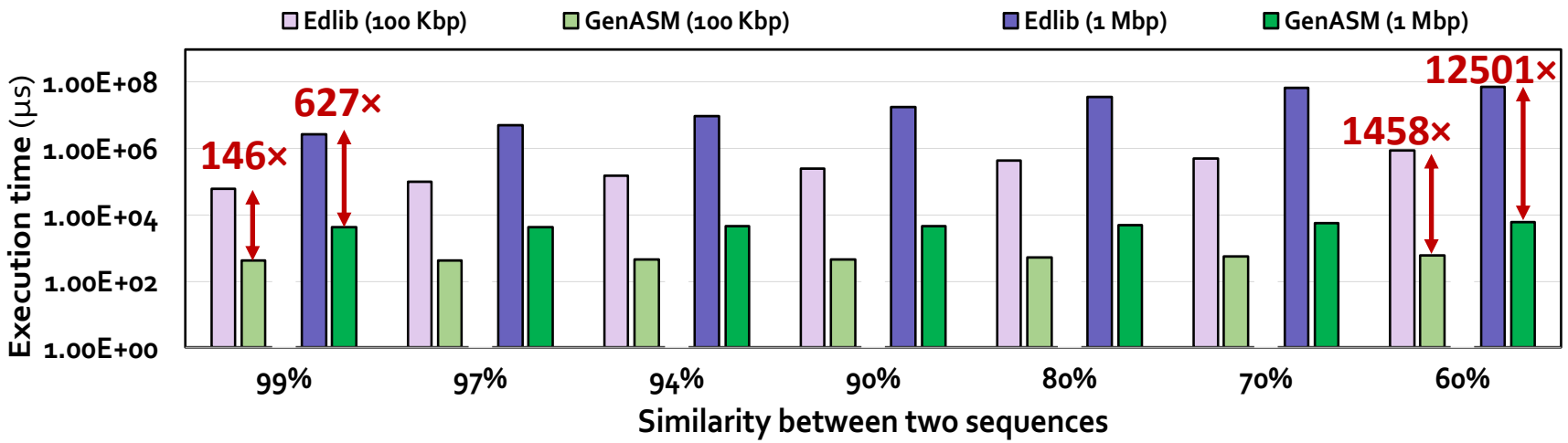
(2) Pre-Alignment Filtering for Short Reads

- Quickly identify and filter out the unlikely candidate reference regions for each read

(3) Edit Distance Calculation

- Measure the similarity or distance between two sequences

Key Results – Use Case 3



SW

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

HW

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

Outline

- Introduction
- Background
  - Approximate String Matching (ASM)
  - ASM with Bitap Algorithm
- GenASM: ASM Acceleration Framework
  - GenASM Algorithm
  - GenASM Hardware Design
  - Use Cases of GenASM
- Evaluation
- **Conclusion**

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

Conclusion

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

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

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

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

Damla Senol Cali

Carnegie Mellon University

(dsenol@andrew.cmu.edu)

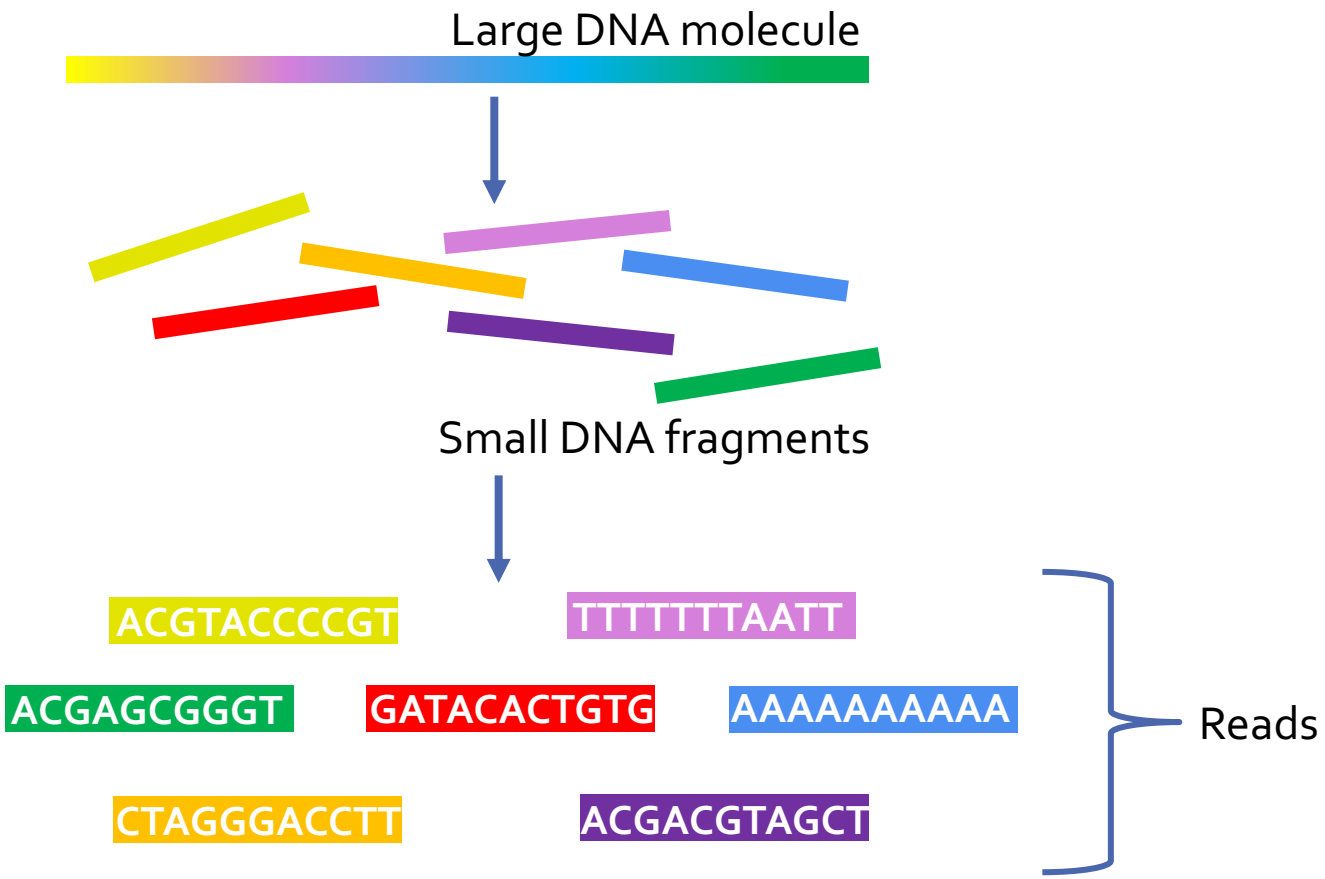
Gurpreet S. Kalsi[2], Zulal Bingol[3], Can Firtina[4], Lavanya Subramanian[5], Jeremie Kim[1,4], Rachata Ausavarungnirun[6,1], Mohammed Alser[4], Juan Gomez-Luna[4], Amirali Boroumand[1], Anant Nori[2], Allison Scibisz[1], Sreenivas Subramoney[2], Can Alkan[3], Saugata Ghose[7,1], and Onur Mutlu[4,1,3]

[1] Carnegie Mellon [2] intel [3] Bilkent University [4] ETH zürich

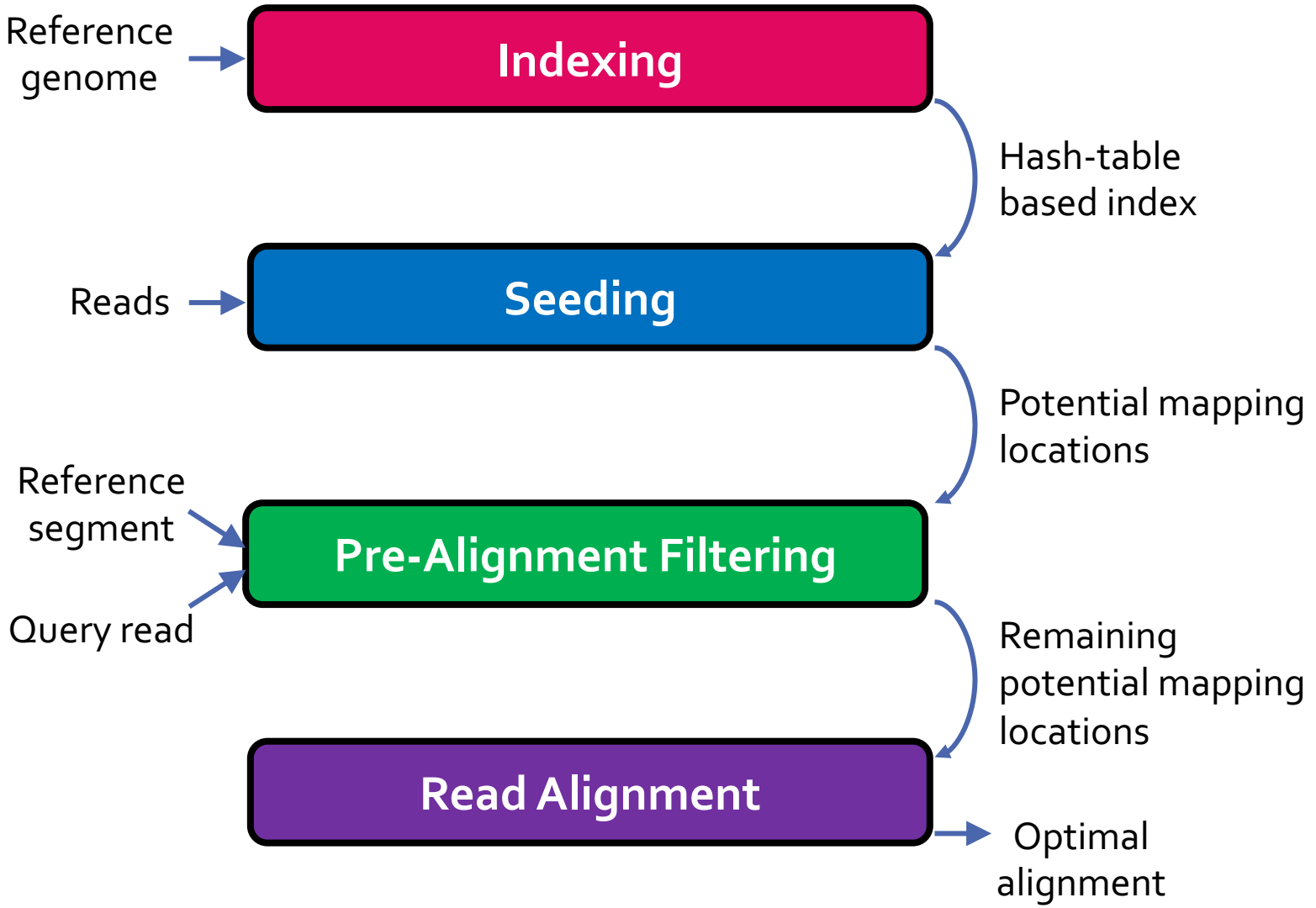
[5] facebook [6] King Mongkut's University of Technology North Bangkok [7] University of Illinois Urbana-Champaign [1,4] SAFARI

Backup Slides

Genome Sequencing



Read Mapping

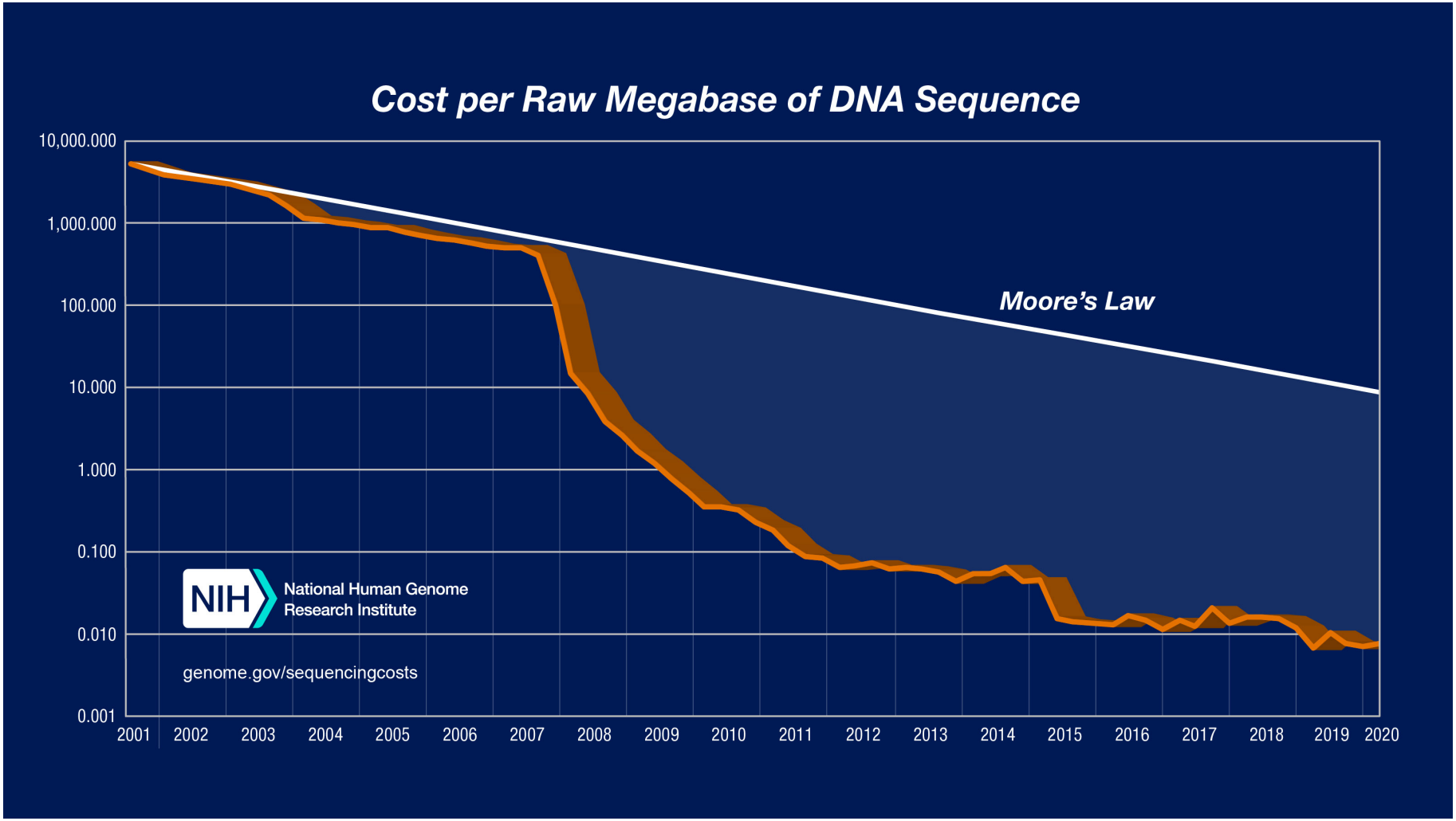


Short Reads vs. Long Reads

- **Short Reads**
  - Sequences with tens to hundreds of bases
  - Highly accurate sequences
  - Output of **SRS** technologies (*e.g.*, Illumina, Ion Torrent)

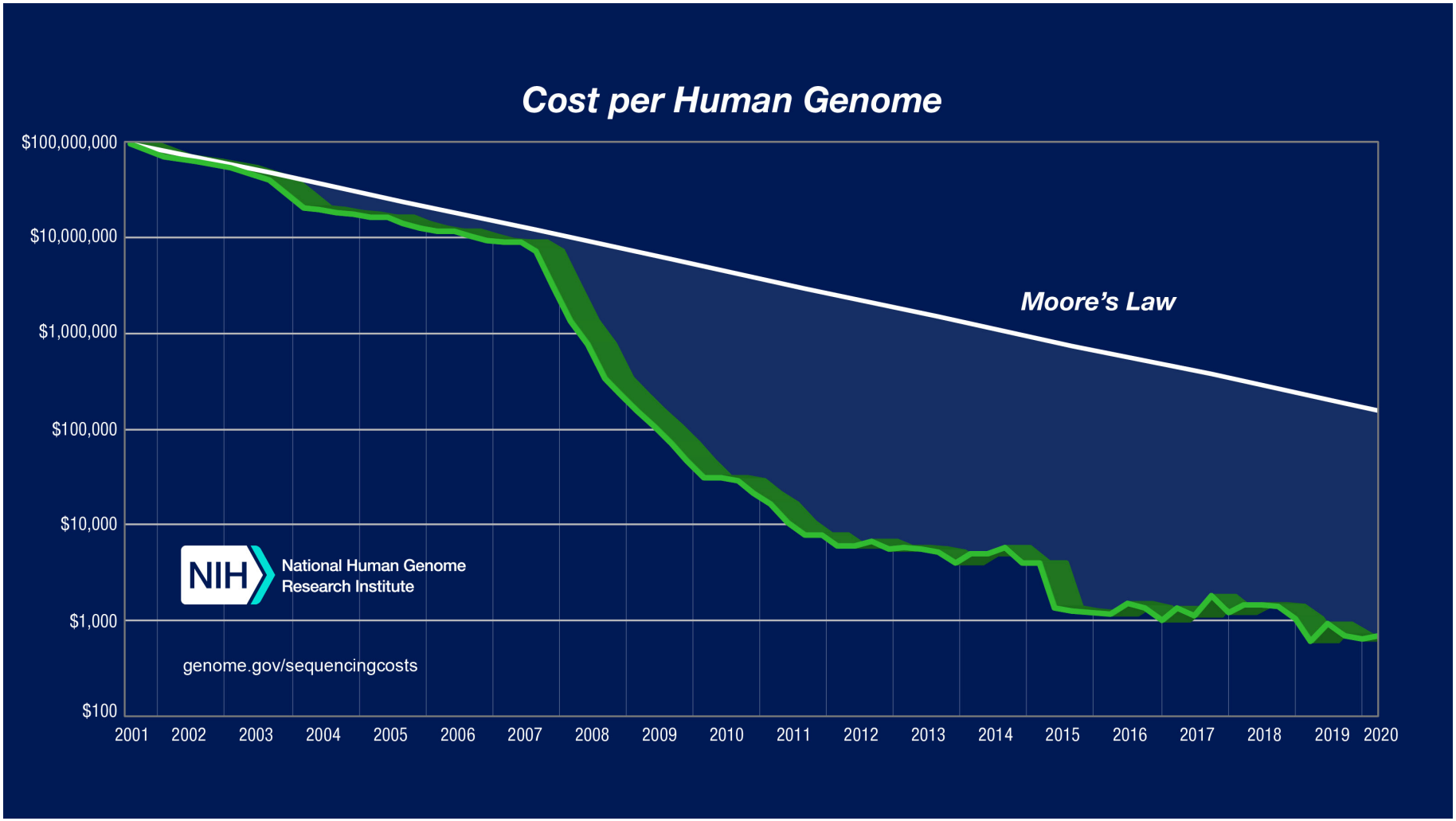
- **Long reads**
  - Sequences with thousands or millions of bases
  - Sequences with high error rates
  - Output of **LRS** technologies (*e.g.*, Oxford Nanopore Technologies, PacBio)

Cost of Sequencing



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

Cost of Sequencing (cont'd.)

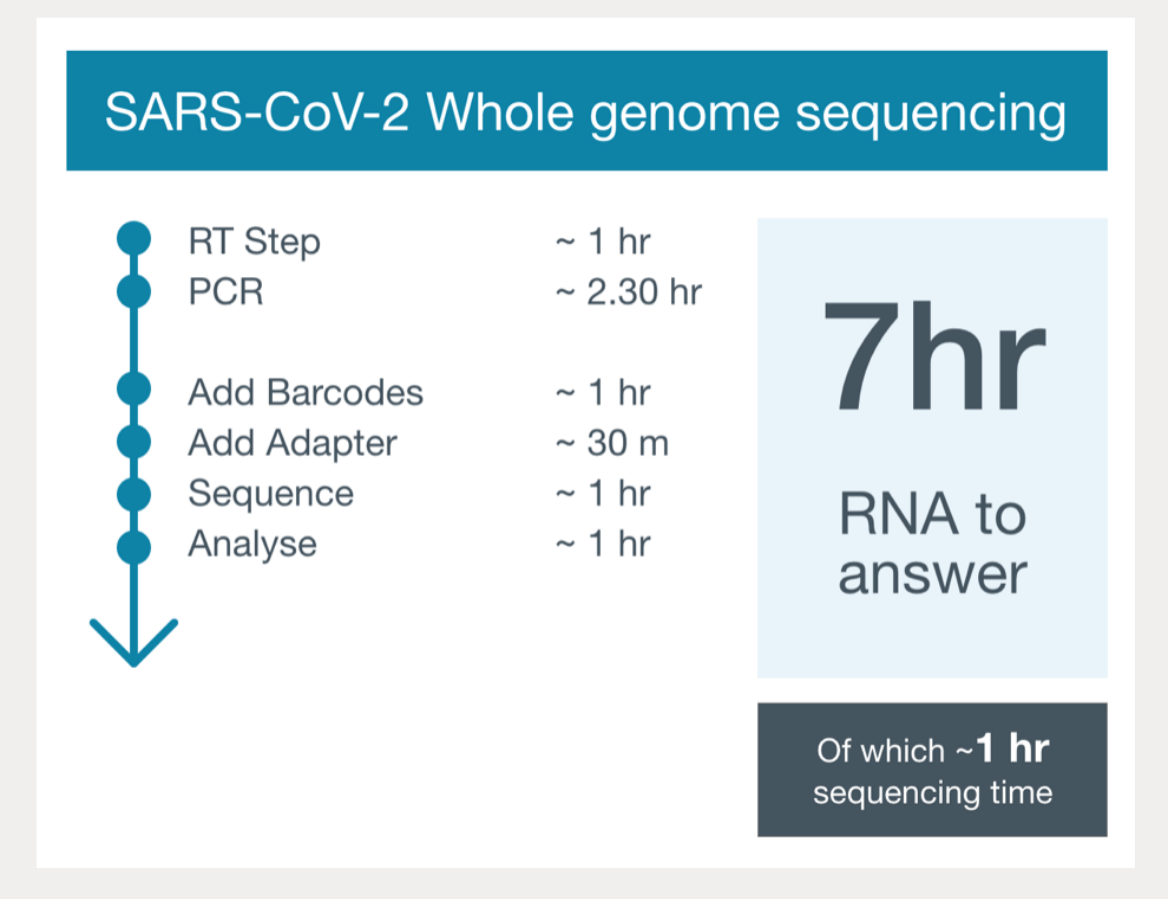


*From NIH (https://www.genome.gov/about-genomics/fact-sheets/DNA-Sequencing-Costs-Data)

Sequencing of COVID-19

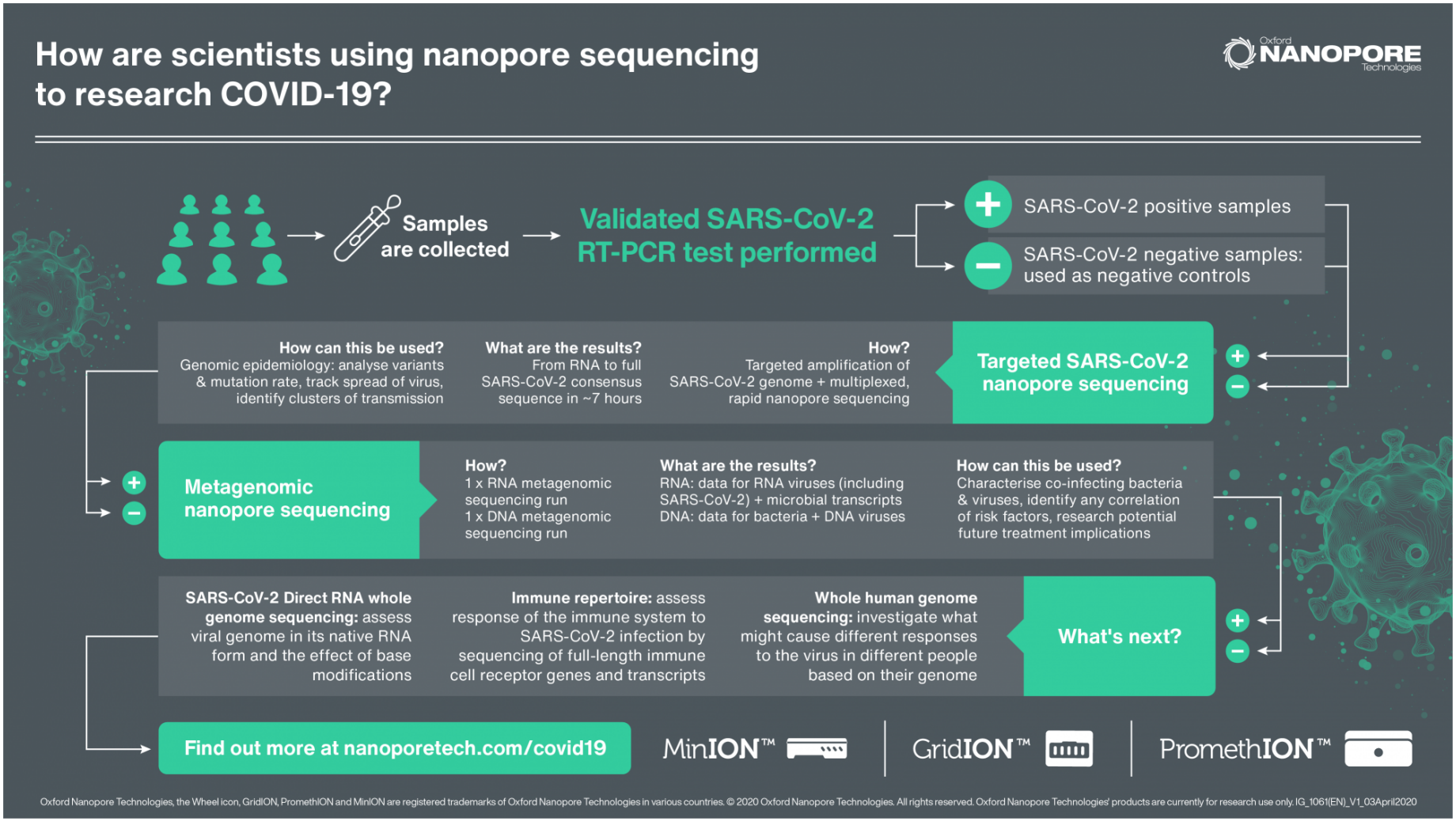
- **Why whole genome sequencing (WGS) and sequence data analysis are important:**
  - To detect the virus from a human sample such as saliva, Bronchoalveolar fluid etc.
  - To understand the sources and modes of transmission of the virus
  - To discover the genomic characteristics of the virus, and compare with the previous viruses (e.g., 02-03 SARS epidemic)
  - To design and evaluate the diagnostic tests
- **Two key areas of COVID-19 genomic research:**
  - To sequence the genome of the virus itself, COVID-19, in order to track the mutations in the virus.
  - To explore the genes of infected patients. This analysis can be used to understand why some people get more severe symptoms than others, as well as, help with the development of new treatments in the future.

COVID-19 Sequencing with ONT



- From ONT (https://nanoporetech.com/covid-19/overview)

COVID-19 Sequencing with ONT (cont'd.)



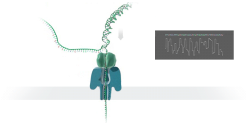
- From ONT (https://nanoporetech.com/covid-19/overview)

Future of Genome Sequencing & Analysis

MinION from ONT

SmidgION from ONT

Nanopore Genome Assembly Pipeline



Raw signal data

Basecalling

DNA reads

Read-to-Read Overlap Finding

Overlaps

Assembly

Assembly

Draft assembly

Read Mapping (Optional)

Mappings of reads against draft assembly

Improved assembly

Polishing (Optional)

Nanopore Sequencing & Tools

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

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

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

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

BiB Version

arXiv Version