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

Damla Senol Cali[1]
Carnegie Mellon University
(dsenol@andrew.cmu.edu)

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

1 Carnegie Mellon
2 intel
3 Bilkent University
4 ETH zürich
5 King Mongkut's University of Technology North Bangkok
6 University of Illinois Urbana-Champaign
SAFARI

To appear in MICRO 2020.

Genome Sequencing

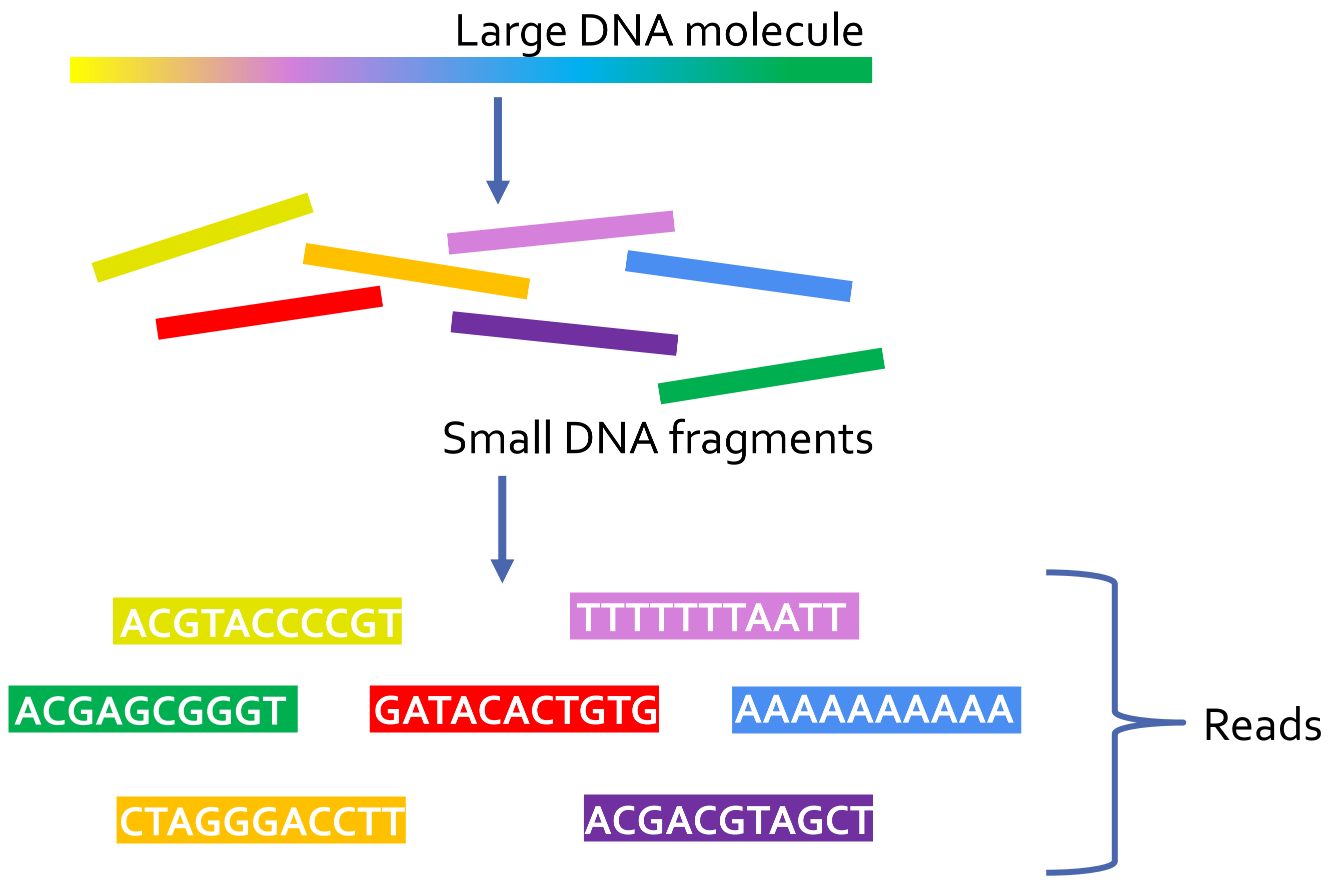
Genome

DNA

- Genome sequencing is the process of determining the order of the DNA sequence in an organism's genome.

- Genome sequencing is pivotal in:
  - Personalized medicine
  - Outbreak tracing
  - Evolution
  - Forensics

Genome Sequencing (cont.)



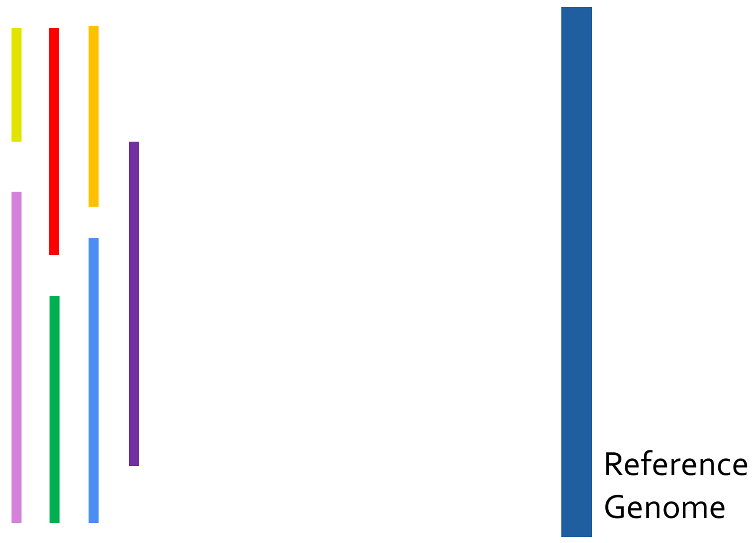
Genome Sequence Analysis

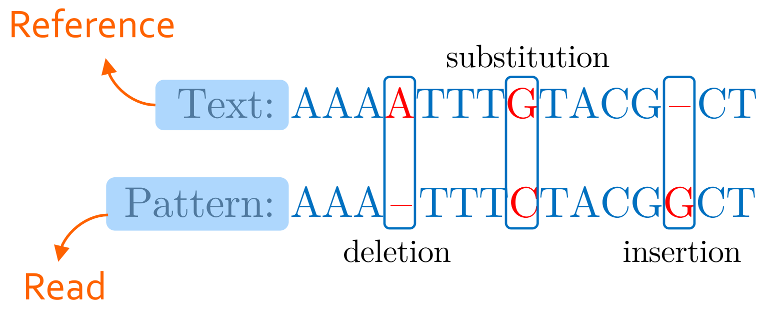
- *Genome sequence analysis* requires:
  1) Taking small DNA fragments from an organism
  2) Reorganizing them into the entire genome

- Success of all medical and genetic applications critically depends on:
  - Existence of computational techniques that can process and analyze the enormous amount of sequence data quickly and accurately

- Effectively leveraging genome sequencing as a tool:
  - Requires very high computational power
  - Requires processing a large amount of data
  - Bottlenecked by the current capabilities of computer systems

Read Mapping





- *Read mapping* is the method of aligning reads against a reference genome to detect matches and variations.
  - → One of the key components of genome sequence analysis.
- Goal is to identify the original location of each read in the reference genome.
- Sequenced genome may not exactly map to the reference genome
  - → **Reason:** mutations, variations, sequencing errors
- *Multiple steps of read mapping* must account for these errors.

Problem & Our Goal

- Multiple steps of read mapping are essentially a series of *approximate* (i.e., *fuzzy) string matches*
- Approximate string matching makes up a significant portion of read mapping (i.e., more than 70%).
- One of the key bottlenecks of the entire genome analysis pipeline.

Our Goal:

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

Outline

- ❑ Background
- ❑ Motivation
- ❑ **ASM with Bitap Algorithm**
- ❑ GenASM: ASM Acceleration Framework
- ❑ Use Cases of GenASM
- ❑ Evaluation
- ❑ Conclusion

Bitap Algorithm

- We have focused on the Bitap algorithm[1,2]

  → **Reason:** *Bitap* algorithm can perform ASM with fast and simple bitwise operations, which makes it amenable to acceleration

- **Step 1: Preprocessing**
  - For each character (A, C, G, T), generate a pattern bitmask
  - 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
    - Set of bitvectors that hold the status of the partial matches
    - Bitwise operations

[1] R. A. Baeza-Yates and G. H. Gonnet. "A new approach to text searching." *Communications of the ACM*, 1992.
[2] S. Wu and U. Manber. "Fast text searching: allowing errors." *Communications of the ACM*, 1992.

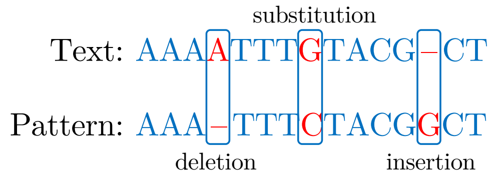
Bitap Algorithm (cont.)

- Each bitvector has a length equal to the length of the pattern (m)

- Semantics of 0 and 1 are reversed: 0 means match, 1 means mismatch

- **Step 1: Preprocessing**

Pattern:	A T T C G A T C
patternBitmask[A]:	0 1 1 1 1 0 1 1
patternBitmask[C]:	1 1 1 0 1 1 1 0
patternBitmask[G]:	1 1 1 1 0 1 1 1
patternBitmask[T]:	1 0 0 1 1 1 0 1



Bitap Algorithm (cont.)

- **Step 2: Searching**

```
For each character of the text (curr):
    Copy the current status of R to oldR
    R[0] = (oldR[0] << 1) | patternBitmask[curr]
    For d = 1...k:
        deletion      = oldR[d-1]
        substitution  = oldR[d-1] << 1
        insertion     = R[d-1] << 1
        match = (oldR[d] << 1) | patternBitmask[curr]
        R[d] = deletion & mismatch & insertion & match
        Check MSB of R[d]:
            If 1, no match.
            If 0, match with d many errors.
```

1) Large number of iterations

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

3) Simple bitwise operations

Limitations of Bitap on Existing Systems

- Data dependency between iterations
  - Limits the efficiency and the scalability of the algorithm on CPUs and GPUs
- Limited compute parallelism
  - Text-level parallelism
  - Limited by the number of compute units in existing systems
- Limited memory bandwidth
  - High memory bandwidth required to read and write the computed bitvectors to memory

Both CPU and GPU systems are imbalanced for this algorithm.

- No support for traceback
  - Finding the sequence of matches, substitutions, insertions and deletions, along with their positions
- No efficient support for both short and long reads
  - Each bitvector has a length equal to the length of the pattern

Outline

GenASM

- Approximate string matching (ASM) acceleration framework based on the Bitap algorithm

- Includes optimized ASM algorithm and new hardware
  - Highly-parallel Bitap with small memory footprint
  - Bitvector-based novel algorithm to perform traceback
  - Processing-in-Memory (PIM) accelerator for Bitap and traceback

- Fast, efficient and flexible framework which can accelerate *multiple steps* of the genome sequence analysis pipeline

- Optimized for both 1) short yet accurate and 2) long but noisy reads

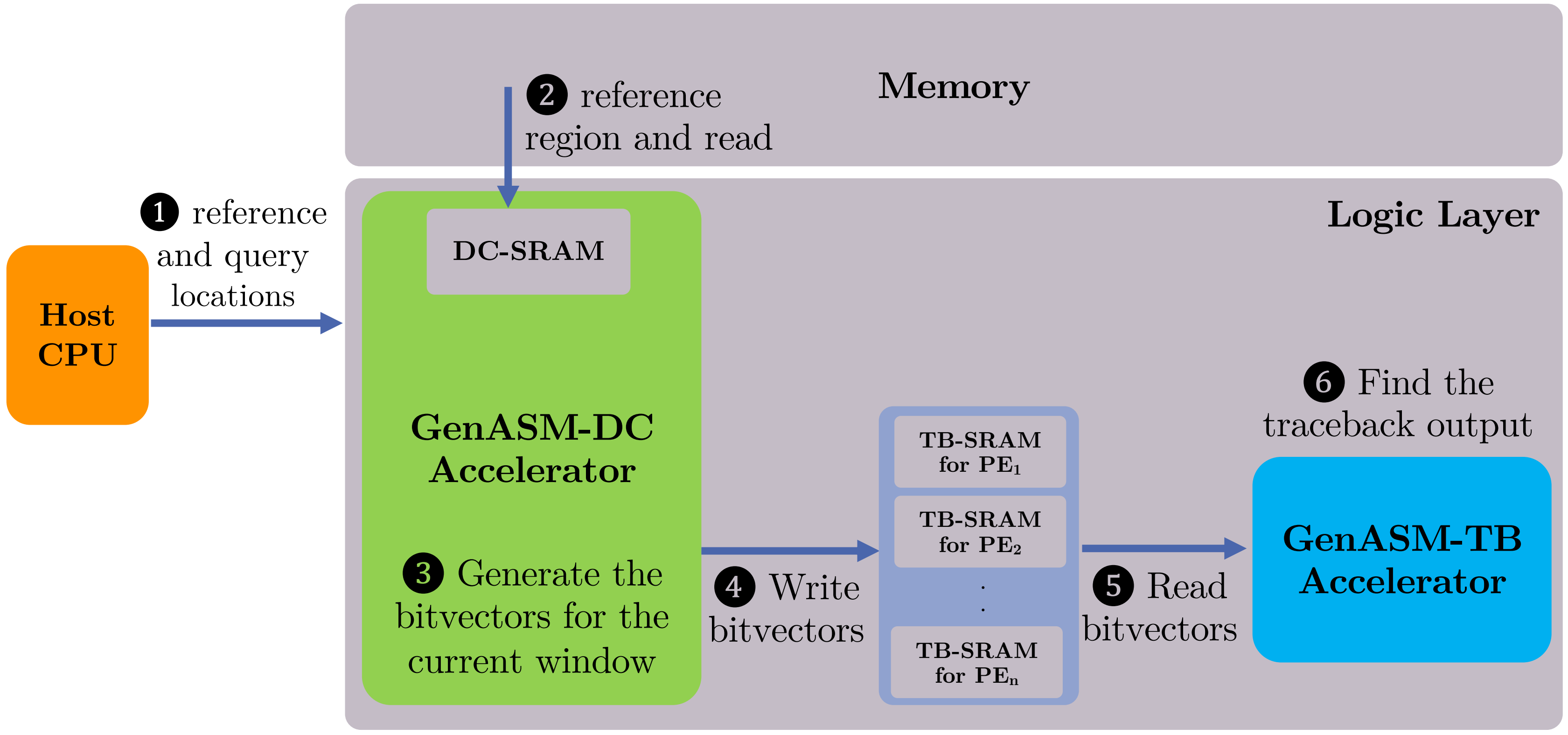
GenASM Algorithm

- We modify the baseline Bitap algorithm to:
  (1) Enable efficient alignment of longer patterns
  (2) Remove the data dependency between the iterations
  (3) Provide parallelism for the large amount of iterations
  (4) Provide support for traceback

- Both **modified Bitap algorithm** and the **novel Bitap-based traceback algorithm** represent the query reads as bitvectors and takes the advantage of *bit-parallelism* during the computation.

- Our traceback algorithm provides:
  (1) Full support for edit distance calculation (i.e., unit cost errors),
  (2) Minimal support for non-unit costs for edits and more complex scoring schemes.

GenASM Design

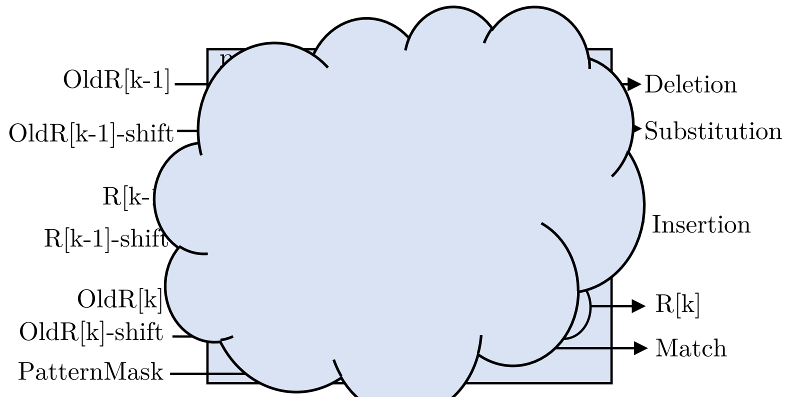


GenASM-DC:
generates bitvectors and performs edit **D**istance **C**alculation

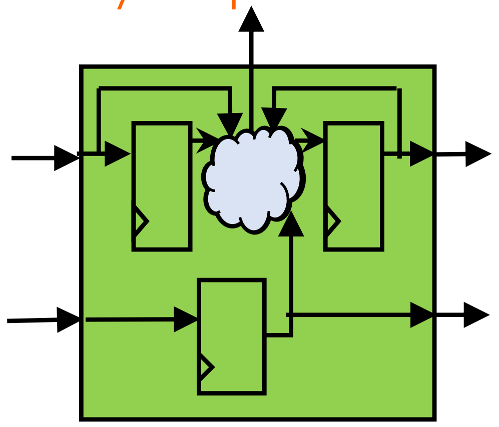
GenASM-TB:
performs **T**race**B**ack and assembles the optimal alignment

GenASM-DC: Hardware Design

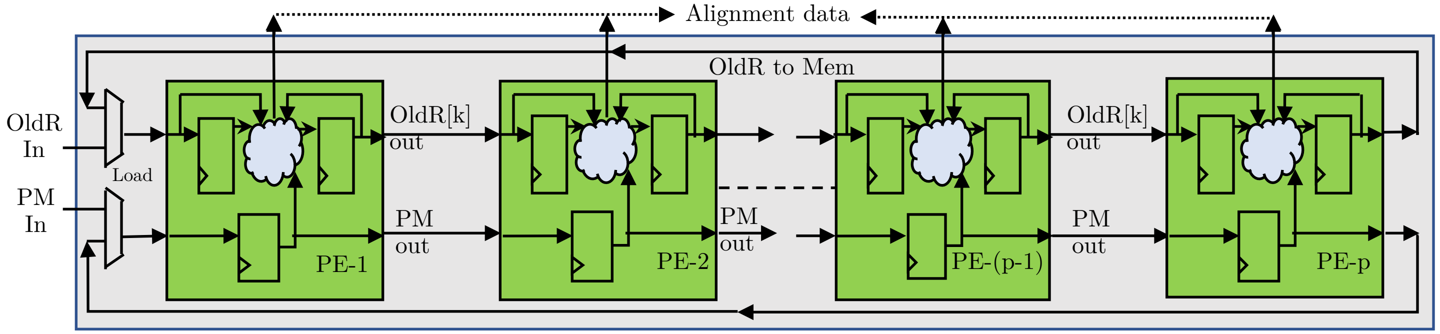
- GenASM-DC Hardware Accelerator (HWA) is implemented as a **linear cyclic systolic array**.
  - Optimized to reduce memory bandwidth and memory footprint



Processing Core (PC)

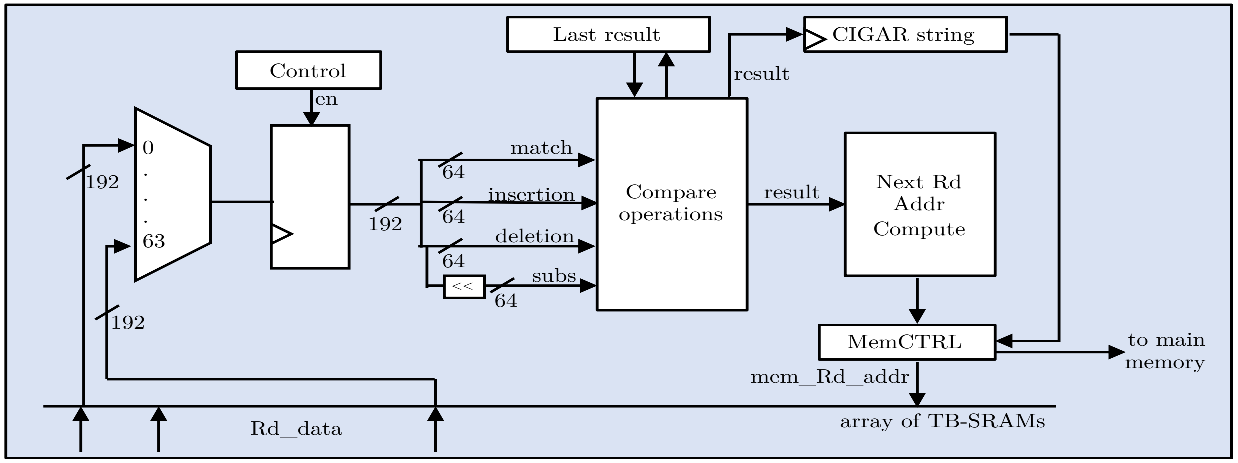


Processing Element (PE)



Processing Block (PB)

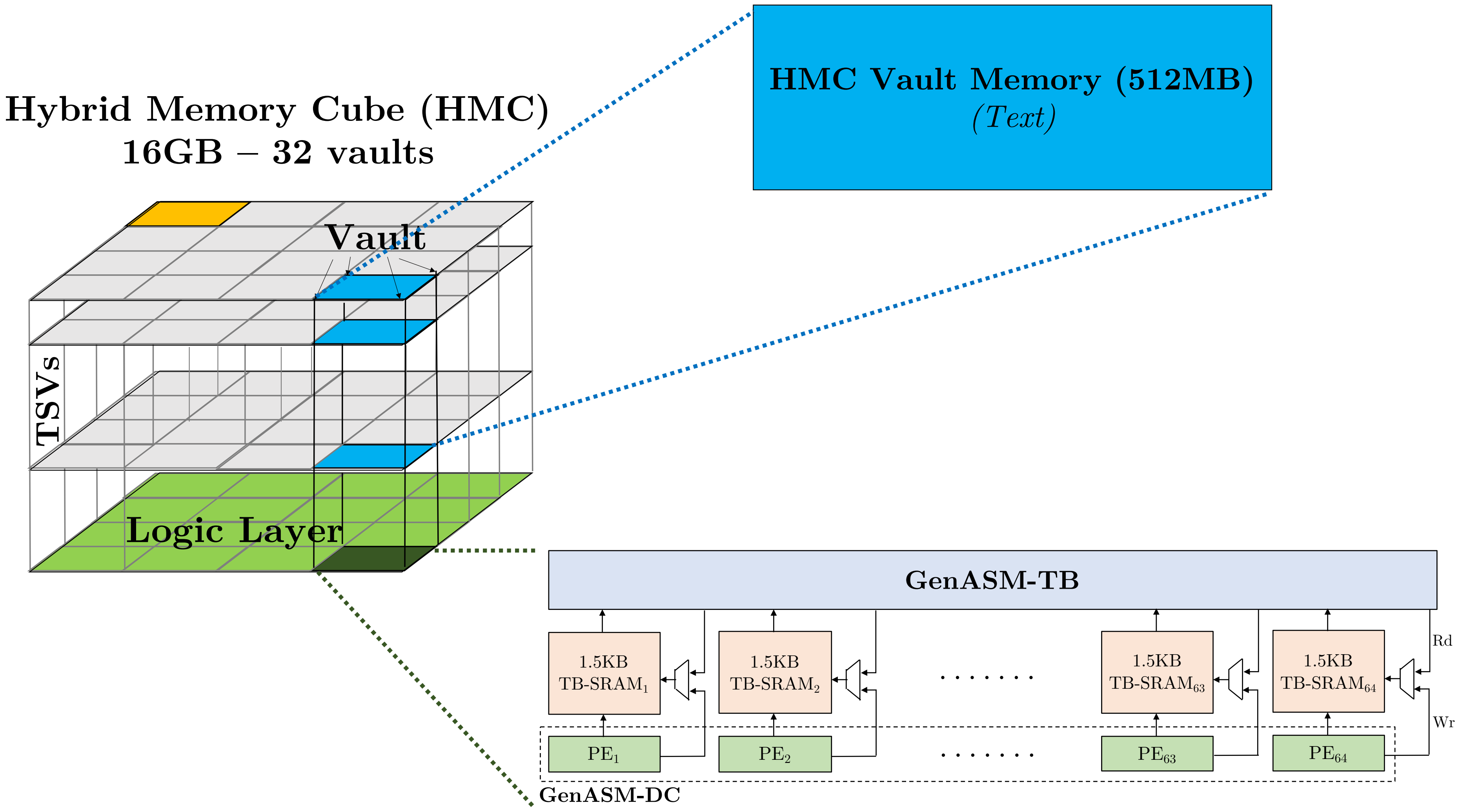
GenASM-TB: Hardware Design



- Very simple logic:
  1) Reads the bitvectors from one of the TB-SRAMs using the computed address
  2) Performs the required computation and comparisons to find the traceback output for the current position
  3) Computes the next TB-SRAM address to read the new set of bitvectors

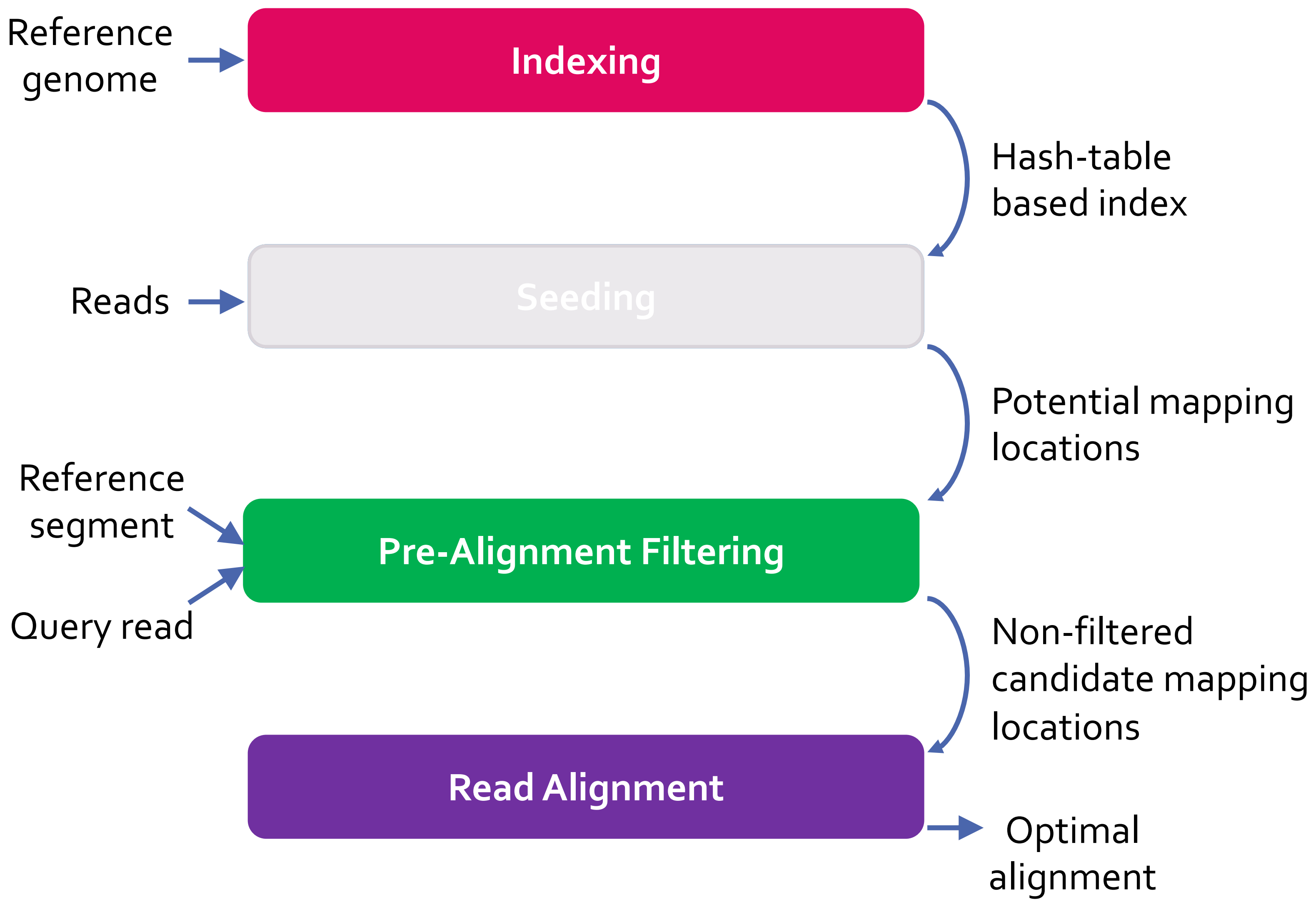
- After GenASM-TB finds the complete traceback output, it writes the output to main memory and completes its execution.

GenASM: Overall System



Outline

Use Cases of GenASM



Use Cases of GenASM (cont.)

(1) Read Alignment Step of Read Mapping

- Also called *verification* or *seed-extension*
- GenASM can perform ASM between the query reads and the candidate regions and report the optimal alignment.

(2) Pre-Alignment Filtering for Short Reads

- Filter out the dissimilar sequences
- GenASM can efficiently calculate the edit distance between the short read and the candidate text and decide whether it is above a user-defined threshold.

(3) Edit Distance Calculation Between Any Two Sequences

- Fundamental operation in genomics
  - Measure the similarity or distance between two sequences
- GenASM-DC is inherently an edit distance calculation accelerator

- We also discuss other possible use cases of GenASM in our paper:
  - Hash-table based indexing, whole genome alignment, generic text search

Outline

Evaluation Methodology

- 16GB HMC-like 3D-stacked DRAM architecture
  - 32 vaults
  - 256GB/s of internal bandwidth, and
  - a clock frequency of 1.25GHz

- Datasets:
  - Simulated long read datasets (ONT and PacBio)
    - 10Kbp reads with 10-15% error rate
  - Simulated short read datasets (Illumina)
    - 100-250bp reads with 5% error rate

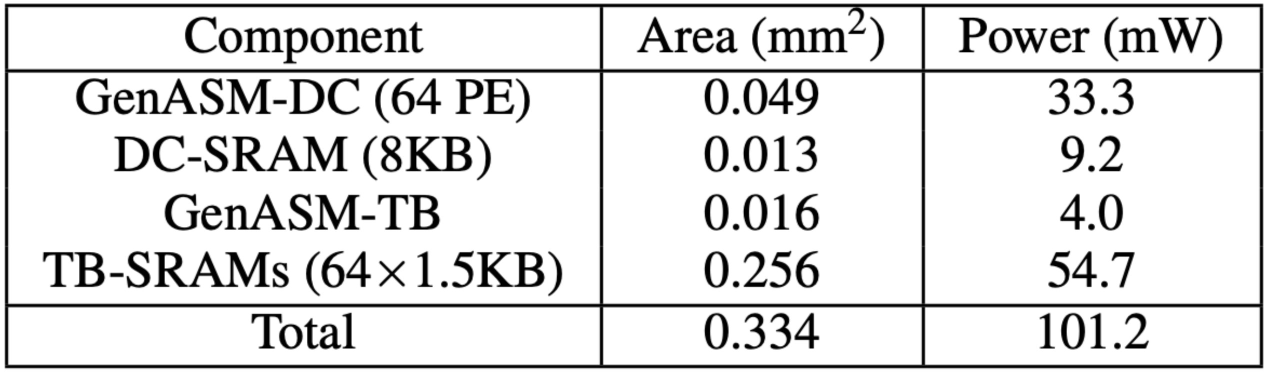
Evaluation Methodology (cont.)

- **For Use Case 1: Read Alignment**, we compare GenASM with:
  - Two state-of-the-art read mappers: Minimap2[1] and BWA-MEM[2]
    - Compare GenASM *only* with the alignment steps of these mappers
    - Running on Intel® Xeon® Gold 6126 CPU (12-core) operating @ 2.60GHz with 64GB DDR4 memory
  - Two state-of-the-art accelerators, Darwin[3] and GenAx[4]
    - Compare GenASM *only* with the alignment components of these accelerators (GACT for Darwin, SillaX for GenAx)

[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.

Key Results – Area and Power

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

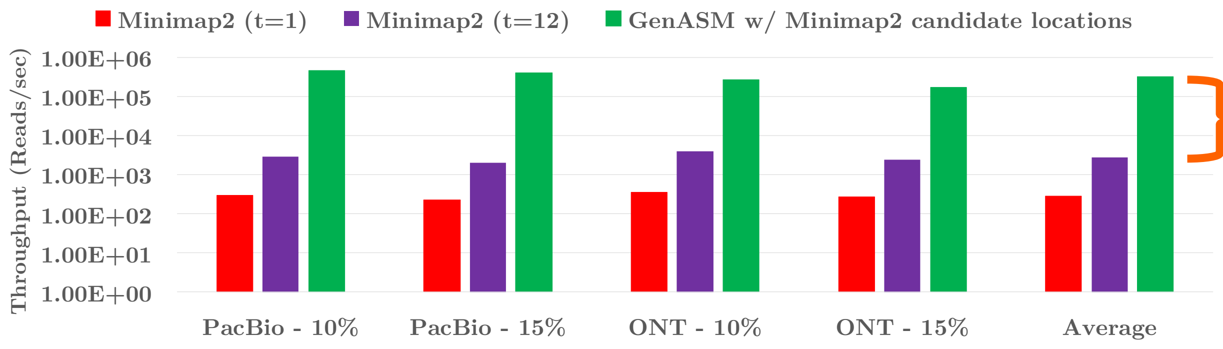


Component	Area (mm^2)	Power (mW)
GenASM-DC (64 PE)	0.049	33.3
DC-SRAM (8KB)	0.013	9.2
GenASM-TB	0.016	4.0
TB-SRAMs (64×1.5KB)	0.256	54.7
Total	0.334	101.2

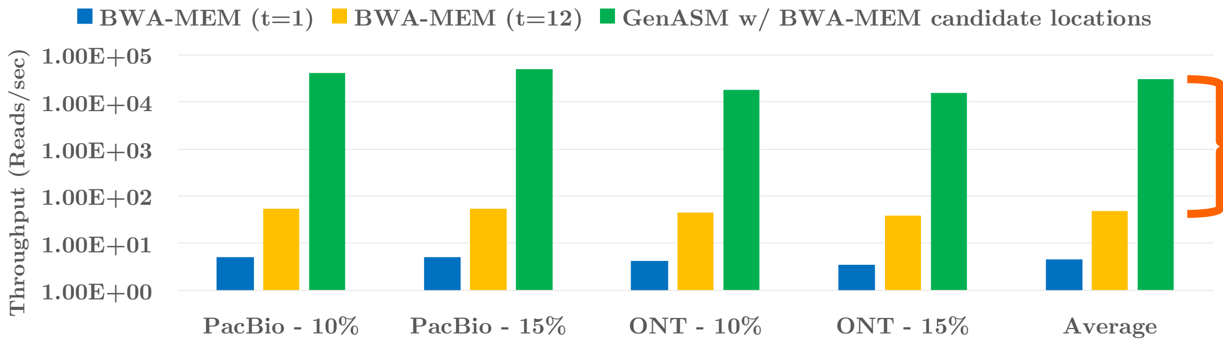
- Total power consumption of all 32 vaults **3.24W**
- Total area overhead of all 32 vaults is **10.69 mm²**

Key Results (Use Case 1) – Long Reads

- **Long Read Datasets:**
  - Compared to 12-thread runs of Minimap2 and BWA-MEM:



116× speedup and **37× less power consumption**

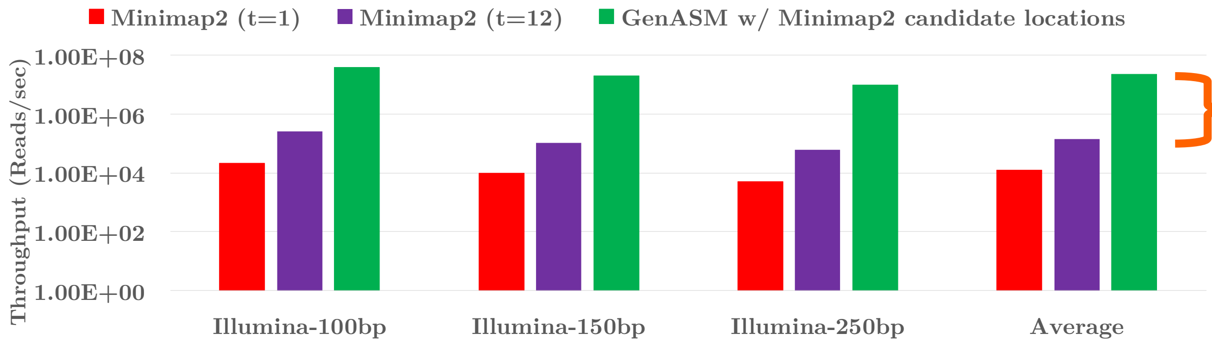


648× speedup and **34× less power consumption**

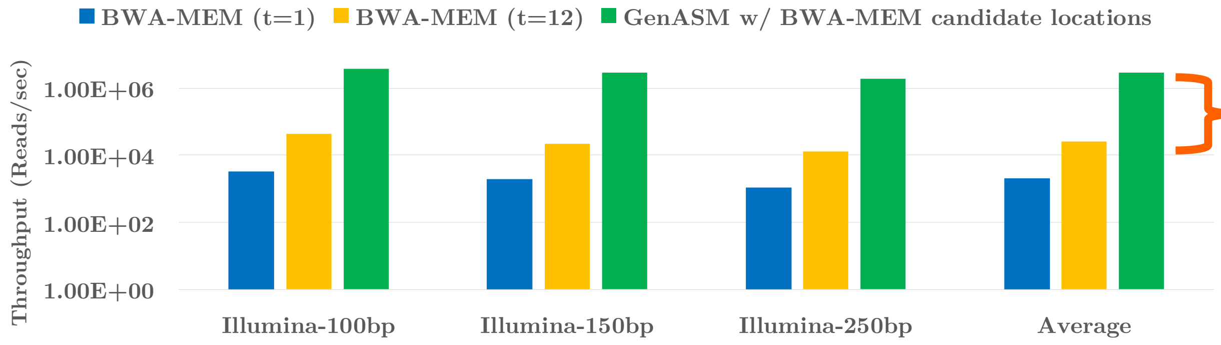
  - Compared to Darwin-GACT:
    - **3.8× better throughput**
    - **2.7× less power consumption**

Key Results (Use Case 1) – Short Reads

- **Short Read Datasets:**
  - Compared to 12-thread runs of Minimap2 and BWA-MEM:



158× speedup and **30× less power consumption**



111× speedup and **33× less power consumption**

  - Compared to GenAx-SillaX:
    - **1.9× better throughput**
    - **Comparable area and power consumption**

Key Results (Use Cases 2 & 3)

- **Pre-Alignment Filtering for Short Reads**
  - Use Case 2
  - **3.6× speedup** vs. Shouji
  - GenASM also significantly improves the filtering accuracy

- **Edit Distance Calculation**
  - Use Case 3
  - **246 – 5668× speedup** vs. Edlib

- See our MICRO 2020 paper for more details

Outline

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.*

- **Goal:** Provide an approximate string matching (ASM) acceleration framework in order to accelerate multiple steps of genome sequence analysis

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

- **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 Low-Power, Memory-Efficient Approximate String Matching Acceleration Framework for Genome Sequence Analysis

Damla Senol Cali[1]
Carnegie Mellon University
(dsenol@andrew.cmu.edu)

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

[1] Carnegie Mellon
[2] intel
[3] Bilkent University
[4] ETH zürich
[5] King Mongkut's University of Technology North Bangkok
[6] University of Illinois Urbana-Champaign

SAFARI

To appear in MICRO 2020.