Lecture 8c:
Storage-Centric Computing
for Genomics and Metagenomics

Nika Mansouri Ghiasi
Prof. Onur Mutlu

ETH Zürich
Summer 2024
06 June 2024
Brief Self Introduction

Nika Mansouri Ghiasi

- PhD student @ SAFARI Research Group, advised by Onur Mutlu
- n.mansorighiasi@gmail.com (Best way to reach me)
- https://safari.ethz.ch

Research in:

- Computer architecture, computer systems, bioinformatics
- Memory and storage systems
- Near-data processing
- Emerging technologies
- Hardware/software cooperation
- ...

...
Focus of this lecture

Apple M1 Ultra System (2022)

https://www.gsmarena.com/apple_announces_m1_ultra_with_20core_cpu_and_64core_gpu-news-53481.php
<table>
<thead>
<tr>
<th>Storage-Centric Computing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>In-Flash Processing</strong></td>
</tr>
<tr>
<td>“Flash-Cosmos: In-Flash Bulk Bitwise Operations Using Inherent Computation Capability of NAND Flash Memory,” in MICRO 2022</td>
</tr>
</tbody>
</table>

| Near-Flash Processing      |
| “GenStore: A High-Performance and Energy-Efficient In-Storage Processing System for Genome Sequence Analysis,” in ASPLOS 2022 |
| “MegIS: High-Performance, Energy-Efficient, and Low-Cost Metagenomic Analysis with In-Storage Processing,” in ISCA 2024 |
Storage-Centric Computing

In-Flash Processing

“Flash-Cosmos: In-Flash Bulk Bitwise Operations Using Inherent Computation Capability of NAND Flash Memory,”
in MICRO 2022

Near-Flash Processing

“GenStore: A High-Performance and Energy-Efficient In-Storage Processing System for Genome Sequence Analysis,”
in ASPLOS 2022

“MegIS: High-Performance, Energy-Efficient, and Low-Cost Metagenomic Analysis with In-Storage Processing,”
in ISCA 2024
GenStore: A High-Performance and Energy-Efficient In-Storage Processing System for Genome Sequence Analysis

Nika Mansouri Ghiası, Jisung Park, Harun Mustafa, Jeremie Kim, Ataberk Olgun, Arvid Gollwitzer, Damla Senol Cali, Can Firtina, Haiyu Mao, Nour Almadhoun Alserr, Rachata Ausavarungnirun, Nandita Vijaykumar, Mohammed Alser, and Onur Mutlu

SAFARI
Genome Sequence Analysis

- **Genome sequence analysis** is critical for many applications
  - Personalized medicine
  - Outbreak tracing
  - Evolutionary studies

- Genome sequencing machines extract smaller fragments of the original DNA sequence, known as **reads**
Genome Sequence Analysis

• **Read mapping:** first key step in genome sequence analysis
  - Aligns *reads* to potential *matching locations* in the *reference genome*
  - For each matching location, the *alignment step* finds the degree of similarity (alignment score)

```
...GCCCATATGGTTAAGCTTCCATGGAAATGGGCTTTCCGCTTCCACAAATG...
```

- **Differences**
  - AAGCTTCCATGG
  - AAATGGGCTTTG

- **Differences**
  - GCCCAAATGGTY

• Calculating the alignment score requires computationally-expensive approximate string matching (ASM) to account for differences between reads and the reference genome due to:
  - Sequencing errors
  - Genetic variation
Genome Sequence Analysis

Data Movement from Storage

Storage System

Main Memory

Cache

Computation Unit (CPU or Accelerator)

Alignment

Computation overhead

Data movement overhead
Accelerating Genome Sequence Analysis

- Storage System
- Heuristics
- Accelerators
- Filters
- Main Memory
- Cache
- Computation Unit (CPU or Accelerator)

✓ Computation overhead

✗ Data movement overhead
Key Idea

Filter reads that do not require alignment inside the storage system

Filtered Reads

Exactly-matching reads
Do not need expensive approximate string matching during alignment

Non-matching reads
Do not have potential matching locations and can skip alignment
Challenges

Filter reads that do not require alignment inside the storage system

Read mapping workloads can exhibit different behavior

There are limited hardware resources in the storage system
GenStore

Filter reads that do not require alignment inside the storage system

GenStore-Enabled Storage System

Main Memory

Cache

Computation Unit (CPU or Accelerator)

✓ Computation overhead

✓ Data movement overhead

GenStore provides significant speedup (1.4x - 33.6x) and energy reduction (3.9x – 29.2x) at low cost
Outline

Background

Motivation and Goal

GenStore

Evaluation

Conclusions
Read Mapping Process

Reference:

```
...AATTTGCCCCATATGGTTAAGCTTCTCTGGGCTTTTCGCTTTG...
```

Read:

```
GCCCCAAATGGTT
```

K-mers:

```
GCC
CCC
CAA
AAA
CCA
...
```

Index:

```
<table>
<thead>
<tr>
<th>K-mer</th>
<th>Locations</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCC</td>
<td>7</td>
</tr>
<tr>
<td>CCC</td>
<td>8</td>
</tr>
<tr>
<td>CAA</td>
<td>1</td>
</tr>
<tr>
<td>AAA</td>
<td>31, 101</td>
</tr>
<tr>
<td>CCA</td>
<td>25, 230, 400</td>
</tr>
</tbody>
</table>

Seeding:
Determine potential matching locations (seeds) in the reference genome

Seed Filtering (e.g., Chaining):
Prune some seeds in the reference genome

Alignment:
Determine the exact differences between the read and the reference genome

SAFARI
Outline

Background

Motivation and Goal

GenStore

Evaluation

Conclusions
Motivation

• Case study on a real-world genomic read dataset
  - Various read mapping systems
  - Various state-of-the-art SSD configurations

The ideal in-storage filter significantly improves performance by

1) reducing the computation overhead
2) reducing the data movement overhead
Motivation

- Case study on a real-world genomic read dataset
  - Various read mapping systems
  - Various state-of-the-art SSD configurations

Filtering outside SSD provides lower performance benefit since it

1) does not reduce the data movement overhead
2) must compete with read mapping for system resources

A HW accelerator reduces the computation bottleneck, which makes I/O a larger bottleneck in the system
Our Goal

Design an in-storage filter for genome sequence analysis in a cost-effective manner

Design Objectives:

**Performance**
Provide high in-storage filtering performance to overlap the filtering with the read mapping of unfiltered data

**Applicability**
Support reads with 1) different properties and 2) different degrees of genetic variation in the compared genomes

**Low-cost**
Do not require significant hardware overhead
Outline

Background

Motivation and Goal

GenStore

Evaluation

Conclusions
**GenStore**

- **Key idea:** Filter reads that do not require alignment inside the storage system

- **Challenges**
  - Different behavior across read mapping workloads
  - Limited hardware resources in the SSD
Filtering Opportunities

• Sequencing machines produce one of two kinds of reads
  - Short reads: highly accurate and short
  - Long reads: less accurate and long

Reads that do not require the expensive alignment step:

**Exactly-matching reads**
Do not need expensive approximate string matching during alignment

• Low sequencing error rates (short reads) combined with
• Low genetic variation

**Non-matching reads**
Do not have potential matching locations, so they skip alignment

• High sequencing error rates (long reads) or
• High genetic variation (short or long reads)
GenStore

GenStore-EM for Exactly-Matching Reads

GenStore-NM for Non-Matching Reads
GenStore

GenStore-EM for Exactly-Matching Reads

GenStore-NM for Non-Matching Reads
GenStore-EM

• Efficient in-storage filter for reads with at least one exact match in the reference genome

• Uses simple operations, without requiring alignment

• Challenge: large number of random accesses per read to the reference genome and its index

Expensive random accesses to flash chips

Limited DRAM capacity inside the SSD
GenStore-EM: Data Structures

• **Read-sized k-mers**: to reduce the number of accesses per each read

  ![Reader](image1)

  ![K-mers](image2)

  Only one index lookup per read

  ✓

• **Sorted read-sized k-mers**: to avoid random accesses to the index

  ✓

  Sequential scan of the read set and the index
GenStore-EM: Data Structures

Sorted Read Table

<table>
<thead>
<tr>
<th>Read</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAAAAAAAAAAA</td>
</tr>
<tr>
<td>AAAAAAAAAG</td>
</tr>
<tr>
<td>AAAAAAAACT</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

Sorted K-mer Index

<table>
<thead>
<tr>
<th>K-mer</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAAAAAAAAAAA</td>
</tr>
<tr>
<td>AAAAAAAAAC</td>
</tr>
<tr>
<td>AAAAAAAAT</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

Read-sized K-mers
GenStore-EM: Finding a Match

<table>
<thead>
<tr>
<th>Sorted Read Table</th>
<th>Sorted K-mer Index</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Read</strong></td>
<td><strong>K-mer</strong></td>
</tr>
<tr>
<td>AAAAAAAAAAAAAA</td>
<td>AAAAAAAAAAAAAA</td>
</tr>
<tr>
<td>AAAAAAAAAAAAG</td>
<td>AAAAAAAAAAAAAC</td>
</tr>
<tr>
<td>AAAAAAAAAAACT</td>
<td>AAAAAAAAAAAAT</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Comparator

Read = K-mer

Exact match ➔ Filter the read
GenStore-EM: Not Finding a Match

**Sorted Read Table**

<table>
<thead>
<tr>
<th>Read</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAAAAAAAAAAAAA</td>
</tr>
<tr>
<td>AAAAAAAAAAAAAG</td>
</tr>
<tr>
<td>AAAAAAAAAACT</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

**Sorted K-mer Index**

<table>
<thead>
<tr>
<th>K-mer</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAAAAAAAAAAAAA</td>
</tr>
<tr>
<td>AAAAAAAAAAAAAC</td>
</tr>
<tr>
<td>AAAAAAAAAAAAAT</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

**Comparator**

Read > K-mer

**Next**
GenStore-EM: Not Finding a Match

Sorted Read Table

<table>
<thead>
<tr>
<th>Read</th>
<th>1, 8, …</th>
<th>23, 37, …</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAAAAAAAAAAAA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AAAAAAAAAAAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AAAAAAAAAC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>…</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Sorted K-mer Index

<table>
<thead>
<tr>
<th>K-mer</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAAAAAAAAAAA</td>
<td></td>
</tr>
<tr>
<td>AAAAAAAAAAC</td>
<td></td>
</tr>
<tr>
<td>AAAAAAAAAAT</td>
<td></td>
</tr>
<tr>
<td>…</td>
<td></td>
</tr>
</tbody>
</table>

Comparator

Read < K-mer

Not an exact match ➔ Send to read mapper
GenStore-EM: Not Finding a Match

Sorted Read Table

<table>
<thead>
<tr>
<th>Read</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAAAAAAAAAAA</td>
</tr>
</tbody>
</table>

Sorted K-mer Index

<table>
<thead>
<tr>
<th>K-mer</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAAAAAAAAAAA</td>
</tr>
</tbody>
</table>

✓ Avoids random accesses
✓ Simple low-cost logic

Comparator

Read < K-mer

Not an exact match ➔ Send to read mapper
GenStore-EM: Optimization

• Read-sized k-mer index takes up a large amount of space (126 GB for human index) due to the larger number of unique k-mers

Sorted K-mer Index

<table>
<thead>
<tr>
<th>Strong Hash Value</th>
<th>Loc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1, 8, ...</td>
</tr>
<tr>
<td>4</td>
<td>51</td>
</tr>
<tr>
<td>7</td>
<td>23, 37</td>
</tr>
<tr>
<td>16</td>
<td>...</td>
</tr>
</tbody>
</table>

Using strong hash values instead of read-sized k-mers reduces the size of the index by 3.9x
Steps 1 and 2 are pipelined. During filtering, GenStore-EM sends the unfiltered reads to the host system.

Data is evenly distributed between channels, dies, and planes to leverage the full internal bandwidth of the SSD.
GenStore

GenStore-EM for Exactly-Matching Reads

GenStore-NM for Non-Matching Reads
GenStore-NM

- Efficient chaining-based in-storage filter to prune most of the non-matching reads

<table>
<thead>
<tr>
<th>Seeding</th>
<th>Determine potential matching locations (seeds) in the reference genome</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seed Filtering (e.g., Chaining)</td>
<td>Prune some seeds in the reference genome</td>
</tr>
<tr>
<td>Alignment</td>
<td>Determine the exact differences between the read and the reference genome</td>
</tr>
</tbody>
</table>

- **Challenge:** how to perform chaining inside the SSD
  - Costly dynamic programming on many seeds in each read
  - Particularly challenging for long reads with many seeds
GenStore-NM: Mechanism

- GenStore-NM uses a light-weight chaining filter
  - Selectively performs chaining only on reads with a small number of seeds
  - Directly sends reads that require more complex chaining to the host system

 Reads with a sufficiently large number of seeds are very likely to align to the reference genome

Filters many non-aligning reads without costly hardware resources in the SSD
GenStore-NM: Mechanism

- GenStore-NM uses a **light-weight chaining filter**
- Selectively performs chaining only on reads with a **small number of seeds**
- Directly sends reads that require more **complex chaining to the host** system

Details on GenStore-NM’s design are in the paper.
Outline

Background

Motivation and Goal

GenStore

Evaluation

Conclusions
Evaluation Methodology

Read Mappers

- **Base**: state-of-the-art software or hardware read mappers
  - Minimap2 [Bioinformatics’18]: software mapper for short and long reads
  - GenCache [MICRO’19]: hardware mapper for short reads
  - Darwin [ASPLOS’18]: hardware mapper for long reads

- **GS**: Base integrated with GenStore

SSD Configurations

- **SSD-L**: with SATA3 interface (0.5 GB/s sequential read bandwidth)
- **SSD-M**: with PCIe Gen3 interface (3.5 GB/s sequential read bandwidth)
- **SSD-H**: with PCIe Gen4 interface (7 GB/s sequential read bandwidth)
Performance – GenStore-EM

For a read set with 80% exactly-matching reads

With the Software Mapper

With the Hardware Mapper

2.1× - 2.5× speedup compared to the software Base

1.5× – 3.3× speedup compared to the hardware Base

On average 3.92× energy reduction
Performance – GenStore-NM

For a read set with 99.7% non-matching reads

With the Software Mapper

With the Hardware Mapper

22.4× – 27.9× speedup compared to the software Base

6.8× – 19.2× speedup compared to the hardware Base

On average 27.2× energy reduction
Area and Power

- Based on **Synthesis** of **GenStore** accelerators using the Synopsys Design Compiler @ 65nm technology node

<table>
<thead>
<tr>
<th>Logic unit</th>
<th># of instances</th>
<th>Area [mm²]</th>
<th>Power [mW]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comparator</td>
<td>1 per SSD</td>
<td>0.0007</td>
<td>0.14</td>
</tr>
<tr>
<td>K-mer Window</td>
<td>2 per channel</td>
<td>0.0018</td>
<td>0.27</td>
</tr>
<tr>
<td>Hash Accelerator</td>
<td>2 per SSD</td>
<td>0.008</td>
<td>1.8</td>
</tr>
<tr>
<td>Location Buffer</td>
<td>1 per channel</td>
<td>0.00725</td>
<td>0.37375</td>
</tr>
<tr>
<td>Chaining Buffer</td>
<td>1 per channel</td>
<td>0.008</td>
<td>0.95</td>
</tr>
<tr>
<td>Chaining PE</td>
<td>1 per channel</td>
<td>0.004</td>
<td>0.98</td>
</tr>
<tr>
<td>Control</td>
<td>1 per SSD</td>
<td>0.0002</td>
<td>0.11</td>
</tr>
<tr>
<td><strong>Total for an 8-channel SSD</strong></td>
<td>-</td>
<td><strong>0.2</strong></td>
<td><strong>26.6</strong></td>
</tr>
</tbody>
</table>

Only **0.006%** of a 14nm Intel Processor, less than **9.5%** of the three ARM processors in a SATA SSD controller.
Other Results in the Paper

• Effect of read set features on performance
  - Data size (up to 440 GB)
  - Filter ratio

• Performance benefit of an implementation of GenStore outside the SSD
  - In some cases, it provides performance benefits due more efficient streaming accesses
  - Provides significantly lower benefit compared to GenStore

• More detailed characterization of non-matching reads across different read mapping use cases and species
Outline

Background

Motivation and Goal

GenStore

Evaluation

Conclusions
Conclusion

• There has been significant effort into improving read mapping performance through efficient heuristics, hardware acceleration, accurate filters

• **Problem**: while these approaches address the computation overhead, none of them alleviate the **data movement overhead** from storage

• **Goal**: improve the performance of genome sequence analysis by effectively reducing unnecessary data movement from the storage system

• **Idea**: filter reads that **do not require the expensive alignment** computation in the storage system to fundamentally reduce the data movement overhead

• **Challenges**:
  - Read mapping workloads can exhibit **different behavior**
  - There are **limited available hardware resources** in the storage system

• **GenStore**: the **first** in-storage processing system designed for genome sequence analysis to reduce both the computation and data movement overhead

• **Key Results**: GenStore provides significant **speedup** (1.4x - 33.6x) and **energy reduction** (3.9x – 29.2x) at low cost
GenStore: A High-Performance and Energy-Efficient In-Storage Processing System for Genome Sequence Analysis

Nika Mansouri Ghiasi, Jisung Park, Harun Mustafa, Jeremie Kim, Ataberk Olgun, Arvid Gollwitzer, Damla Senol Cali, Can Firtina, Haiyu Mao, Nour Almadhoun Alserr, Rachata Ausavarungnirun, Nandita Vijaykumar, Mohammed Alser, and Onur Mutlu
## Storage-Centric Computing

### In-Flash Processing

```
“Flash-Cosmos: In-Flash Bulk Bitwise Operations
Using Inherent Computation Capability of NAND Flash Memory,”
in MICRO 2022
```

### Near-Flash Processing

```
“GenStore: A High-Performance and Energy-Efficient
In-Storage Processing System for Genome Sequence Analysis,”
in ASPLOS 2022
```

```
“MegIS: High-Performance, Energy-Efficient, and Low-Cost
Metagenomic Analysis with In-Storage Processing,”
in ISCA 2024
```
MegIS: High-Performance and Low-Cost Metagenomic Analysis with In-Storage Processing

Nika Mansouri Ghiasi, Mohammad Sadrosadati, Harun Mustafa, Arvid Gollwitzer, Can Firtina, Julien Eudine, Haiyu Mao, Joël Lindegger, Meryem Banu Cavlak, Mohammed Alser, Jisung Park, Onur Mutlu

SAFARI

ETH Zürich

POSTECH
Outline

Background

Motivation and Goal

MegIS

Evaluation

Conclusions
What is Metagenomics?

- **Metagenomics**: Analysis of genome sequences of diverse organisms within a shared environment (e.g., blood, soil, oceans)

- Overcomes the limitations of traditional genomics
  - Bypasses the need for isolating and culturing individual species

- Has led to ground-breaking advancements
  - Precision medicine
  - Understanding microbial diversity of an environment
  - Discovering early warnings of communicable
Metagenomic Analysis

Metagenomic sample with species that are not known in advance

A large database of many known species

1. Presence/Absence Identification

K-mers

- GCTTA
- CTTAC
- TTACG
- ...

2. Abundance Estimation

- Species#1
- Species#2
- Species#3

© Tax IDs in the Sample

- 1638
- 23
- 421
- 23
- ...

Presence/Absence Identification

Tax IDs in the Sample
Outline

Background

Motivation and Goal

MegIS

Evaluation

Conclusions
Motivation

Case study on a metagenomic read dataset

• Tools with different access patterns to the database
  - **RQ**: Random queries
  - **SQ**: Sequential queries

• Various state-of-the-art SSD configurations

I/O data movement overhead causes significant performance overhead
Motivation

Case study on a metagenomic read dataset

• Tools with different access patterns to the database
  - **RQ**: Random queries
  - **SQ**: Sequential queries

• Various state-of-the-art SSD configurations

The I/O impact on end-to-end performance becomes even more prominent in emerging systems in which other bottlenecks are alleviated

<table>
<thead>
<tr>
<th>Database Size (Terabyte)</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>0.6</td>
<td>0.6</td>
</tr>
<tr>
<td>0.7</td>
<td>0.7</td>
</tr>
<tr>
<td>1.4</td>
<td>1.4</td>
</tr>
</tbody>
</table>

I/O data movement overhead causes significant performance overhead
Challenge and Goal

• **Challenge**: Despite the benefits of ISP, none of the existing approaches to metagenomic analysis can be effectively implemented as an ISP system due to the limited hardware resources available in current storage devices.

**Our Goal:**

*Improve metagenomic analysis performance by reducing the large data movement overhead from the storage system in a cost-effective manner*
MegIS

- The *first* ISP system for end-to-end metagenomic analysis
- **Key Idea:** Cooperative ISP for metagenomics
  - Enabled via a synergistic hardware/software co-design between the storage system and the host system
  - Capitalize on the strengths of processing both inside and outside the storage system
MegIS

• The first ISP system for end-to-end metagenonomic analysis

hardware/software co-design with specialized and efficient

1. Task partitioning
2. Data/computation flow coordination
3. Storage-aware algorithms
4. Lightweight in-storage accelerators
5. Data mapping
MegIS Steps

Step 1: Preparing the Input Queries

Step 2: Finding Candidate Species

Step 3: Abundance Estimation (Optional)
MegIS Steps

Step 1: Preparing the InputQueries

Step 2: Finding Candidate Species

Step 3: Abundance Estimation (Optional)
Step 1: Preparing the Input Queries

**Functionality**
- Extract k-mers from the reads
- Sort the k-mers (database is sorted offline)
- If needed, prune some k-mers

**Execute on the host system**
- Benefits from the relatively larger DRAM and compute resources
- Fewer writes
- Pipeline and overlap this step with Step 2

**Requirements**
- Partitioning the application should not incur significant overhead due to data transfer time
- Improve performance, lifespan, and endurance even when host DRAM can't hold all sample k-mers
Step 1: Preparing the Input Queries

- Specialized bucketing scheme
- Host-pinned and SSD-pinned buckets
- Bucket buffers

1. **Reads**
   - ACGTTACGATTAG
   - ACGTT
   - CGTTA
   - GTTAC

2. **Extract K-mers**
   - [AAA, CAA)
   - [CAA, CGG)
   - [GCC, TCA)
   - [TCA, TTT]

3. **Partition to Buckets**
   - Bucket#1
   - Bucket#2
   - Bucket#3

4. **Store the Bucket**
   - MegIS-Enabled SSD
MegIS Steps

Step 1: Preparing the Input Queries

Step 2: Finding Candidate Species

Step 3: Abundance Estimation (Optional)
Step 1: Preparing the Input Queries

**Functionality**
- **Step 2.1:** Identify the intersecting k-mers between the query k-mers and the database
- **Step 2.2:** Retrieves the species tax IDs of the intersecting k-mers

**Execute in the storage system**
- Accesses large data with low reuse

**Requirements**
- Leverage internal bandwidth efficiently
- Does not require expensive hardware inside the SSD (e.g., large DRAM bandwidth and costly logic units)
Step 2.1: Intersection Finding

- Avoid bottlenecking internal DRAM bandwidth
  - Compute directly on the NAND flash data streams
  - Using only two k-mer registers per channel
Step 2.2: Retrieving Tax IDs

- Retrieves the species tax IDs of the intersecting k-mers amongst the pre-built k-mer sketches representing each species.

- Requires many pointer-chasing operations on a large data structure that may not fit in the internal DRAM.

### a Baseline K-mer Sketch Tables

<table>
<thead>
<tr>
<th>K-mer</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAAAA</td>
<td>1</td>
</tr>
<tr>
<td>AAAAC</td>
<td>6</td>
</tr>
<tr>
<td>AATCC</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

### b Ternary Search Tree

- A → A → A/8 → A → A/1 → C/6 → T/5 → C/3 → C/2

### c K-mer Sketch Streaming Tables

#### 4-mer i

<table>
<thead>
<tr>
<th>K-mer</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAAAA</td>
<td>1</td>
</tr>
<tr>
<td>AAAAC</td>
<td>6</td>
</tr>
<tr>
<td>AATCC</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

#### 4-mer i+1

<table>
<thead>
<tr>
<th>K-mer</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAAAA</td>
<td>-</td>
</tr>
<tr>
<td>AAAAC</td>
<td>8</td>
</tr>
<tr>
<td>AATCC</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Step 2.2: Retrieving Tax IDs

- KSS example when retrieving 5- and 4-mers
Step 2.2: Retrieving Tax IDs

- KSS example when retrieving 5- and 4-mers
MegIS Steps

Step 1: Preparing the Input Queries

Step 2: Finding Candidate Species

Details on this step and MegIS-FTL are in the paper

Step 3: Abundance Estimation (Optional)
Outline

Background

Motivation and Goal

MegIS

Evaluation

Conclusions
Evaluation Methodology

Metagenomic Analysis Tools

- **Performance-Optimized SW:** Kraken2 [Genome Biology’19]
- **Accuracy-Optimized SW:** Metalign [Genome Biology’20]
- **PIM HW:** Sieve [ISCA’21]

SSD Configurations

- **SSD-C:** with SATA3 interface (0.5 GB/s sequential read bandwidth)
- **SSD-P:** with PCIe Gen4 interface (7 GB/s sequential read bandwidth)
Performance

• With inputs with Low, Medium, and High genetic diversity

![Graph showing speedup and average energy reduction]

- 2.7x – 6.4x speedup compared to P-Opt
- 6.9x – 20.4x speedup compared to A-Opt
- 5.4x average energy reduction compared to P-Opt
- 15.2x average energy reduction compared to A-Opt
Performance Compared to PIM

- With inputs with Low, Medium, and High genetic diversity

1.5x – 5.1x speedup compared to the PIM baseline

1.9x (3.5x) average (max) energy reduction compared to the PIM baseline
System Cost Efficiency

- MegIS on a cost-optimized system
  - With SSD-C and 64-GB DRAM
- Baselines on a performance-optimized system
  - With SSD-P and 1-TB DRAM

\[ \text{GMean} \]

2.4x average speedup compared to P-Opt
7.2x average speedup compared to A-Opt
Area and Power

- Based on Synthesis of MegIS accelerators using the Synopsys Design Compiler at 65nm technology node

<table>
<thead>
<tr>
<th>Logic unit</th>
<th># of instances</th>
<th>Area [mm²]</th>
<th>Power [mW]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intersect (120-bit)</td>
<td>1 per channel</td>
<td>0.001361</td>
<td>0.284</td>
</tr>
<tr>
<td>k-mer Registers (2×120-bit)</td>
<td>1 per channel</td>
<td>0.002821</td>
<td>0.645</td>
</tr>
<tr>
<td>Index Generator (64-bit)</td>
<td>1 per channel</td>
<td>0.000272</td>
<td>0.025</td>
</tr>
<tr>
<td>Control Unit</td>
<td>1 per SSD</td>
<td>0.000188</td>
<td>0.026</td>
</tr>
</tbody>
</table>

Total for an 8-channel SSD | -              | **0.04**     | **7.658**  

Only **1.7%** of the area of three 28-nm ARM Cortex R4 cores in a SATA SSD controller
Other Results in the Paper

• MegIS’s performance with the SSD cores
• MegIS’s performance outside SSD
• Performance with varying
  - Database sizes
  - Memory capacities
  - #SSDs
  - #Channels
  - #Samples
• MegIS’s performance for abundance estimation
Outline

Background

Motivation and Goal

MegIS

Evaluation

Conclusions
Conclusion

- **Problem**: Significant storage I/O data movement overhead due to accessing large amounts of low-reuse data

- **Goal**: Improve the performance of metagenomic analysis by reducing data movement overhead right from the storage system, in a cost-effective manner

- **Challenge**: While in-storage processing can be a promising direction, none of the existing approaches can be effectively implemented inside the storage system due to the limited resources available in storage devices

- **Idea**: Enable cooperative ISP for metagenomics, where we do not merely focus on the storage system and, instead, capitalize on the strengths of processing both inside and outside the storage system via a synergistic design

- **MegIS**: *The first in-storage processing system tailored for metagenomic analysis*
  - Leverages and orchestrates processing inside and outside the storage system
  - 1) Task partitioning, 2) Data/computation flow coordination, 3) Storage-aware algorithms, 4) Lightweight in-storage accelerators, and 5) Data mapping

- **Results**: Significant speedup (1.5x – 100.2x) and energy reduction (1.9x – 25.7x) with high accuracy and at low cost
MegIS: High-Performance and Low-Cost Metagenomic Analysis with In-Storage Processing

Nika Mansouri Ghiasi, Mohammad Sadrosadati, Harun Mustafa, Arvid Gollwitzer, Can Firtina, Julien Eudine, Haiyu Mao, Joël Lindegger, Meryem Banu Cavlak, Mohammed Alser, Jisung Park, Onur Mutlu

SAFARI

ETH Zürich

POSTECH
Lecture 8c:
Storage-Centric Computing for Genomics and Metagenomics

Nika Mansouri Ghiasi
Prof. Onur Mutlu

ETH Zürich
Summer 2024
06 June 2024