GenStore: A High-Performance In-Storage Processing System for Genome Sequence Analysis

Nika Mansouri Ghiasi (mnika@ethz.ch), 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

ETH zürich · Carnegie Mellon · University of Toronto

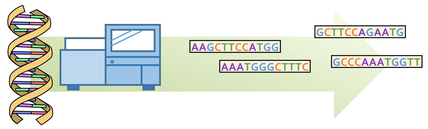
1: Summary

- There has been significant effort into improving read mapping performance through efficient heuristics, hardware acceleration, and 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**: 1) Read mapping workloads can exhibit **different behavior**
  2) 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/data movement overhead
- **Key Results**: GenStore provides significant **speedup (1.4x - 33.6x)** and **energy reduction (3.9x – 29.2x)** at **low cost**

2: Genome Sequence Analysis

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



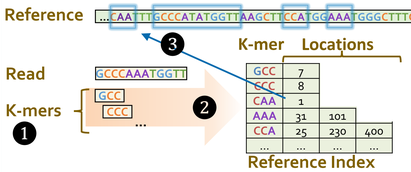
Read Mapping

- **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)

- Calculating the alignment score requires computationally-expensive approximate string matching (ASM) to account for differences between reads and the reference genome

3: Read Mapping Steps

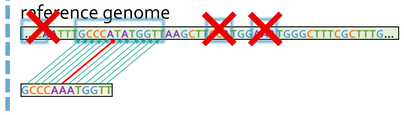
1. Seeding: finds potential matching locations (seeds) in the reference.



1. Generate k-mers from the read
2. Look up k-mers in an index of the reference genome
3. Mark the locations of k-mers in the reference as seeds.

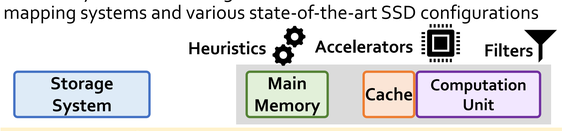
2. Chaining or Seed Filtering: prunes some seeds in the reference genome to which the reads would not align

3. Alignment: determines the exact differences between the read and the reference genome



4: Motivation

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



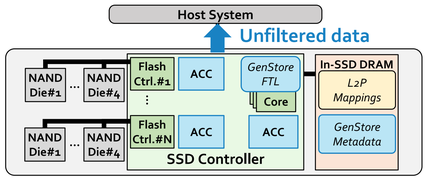
✓ Computation overhead

✗ Data movement overhead

The ideal in-storage filter significantly improves performance by reducing the computation and data movement overheads

5: 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

Exactly-matching reads
Do not need expensive approximate string matching during alignment
- Low sequencing error rates combined with Low genetic variation

Non-matching reads
Do not have potential matching locations, so they skip alignment
- High sequencing error rates or High genetic variation

6: GenStore-EM Overview

- 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

7: GenStore-EM Design

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

✓ Only one index lookup per read

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

✓ Sequential scan of the the index

Sorted Read Table

ID	Read
873	AAAAAAAAAA
232	AAAAAAAAAG
17	AAAAAAAACT
...	...

Sorted Read-Sized K-mer Index

K-mer	Loc.
AAAAAAAAAA	1, 8, ...
AAAAAAAAAC	51
AAAAAAAAAT	23, 37
...	...

Next → Comparator ← Next

Read = K-mer

Exact match → Filter the read

- 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

Strong Hash Value	Loc.
1	1, 8, ...
4	51
7	23, 37
16	...

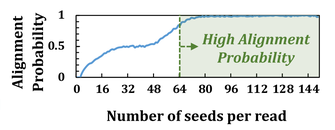
Using strong hash values instead of read-sized k-mers reduces the size of the index by 3.9x

8: GenStore-NM Overview

- Efficient chaining-based in-storage filter to prune most of the non-matching reads
- **Challenge:** to perform chaining inside the SSD

Costly dynamic programming on many seeds in each read is particularly challenging for long reads

- 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

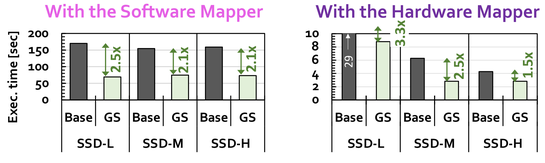
9: Evaluation

Methodology

Read Mappers
- **Base:** state-of-the-art software or hardware read mappers
  - Minimap2 [Bioinformatics'18]: software mapper
  - 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
- **SSD-M:** with PCIe Gen3 interface
- **SSD-H:** with PCIe Gen4 interface

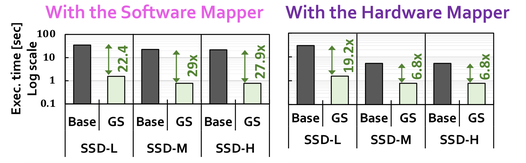
GenStore-EM: For a read set with 80% exactly-matching reads



2.1× - 2.5× speedup | 1.5× – 3.3× speedup

On average 3.92× energy reduction

GenStore-NM: For a read set with 99.7% non-matching reads



22.4× – 27.9× speedup | 6.8× – 19.2× speedup

On average 27.2× energy reduction

Other Results

- Effect of read set features on performance
- Performance benefit of an implementation of GenStore outside the SSD
- More detailed characterization of read mapping use cases

SAFARI