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

Session 6A: Thursday 3 March, 3:00 PM CEST

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
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)

```
...GCCCATATGGTTAAGCTTCCATGGAATGGGCTTTCGCTTCCAGAATG...
```

**Reference Genome**

- **Differences**
  - AAGCTTCCATGG
  - AAATGGGCTTTCC

- **Differences**
  - GCCCAAATGGGTT

• 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)

Computation overhead

Data movement overhead
Accelerating Genome Sequence Analysis

- Heuristics
- Accelerators
- Filters

Storage System

- 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

**SAFARI**
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
<table>
<thead>
<tr>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Background</strong></td>
</tr>
<tr>
<td><strong>Motivation and Goal</strong></td>
</tr>
<tr>
<td><strong>GenStore</strong></td>
</tr>
<tr>
<td><strong>Evaluation</strong></td>
</tr>
<tr>
<td><strong>Conclusions</strong></td>
</tr>
</tbody>
</table>
Read Mapping Process

Reference

Read

K-mers

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
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
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:**

<table>
<thead>
<tr>
<th>Exactly-matching reads</th>
<th>Non-matching reads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do not need expensive approximate string matching during alignment</td>
<td>Do not have potential matching locations, so they skip alignment</td>
</tr>
<tr>
<td>• Low sequencing error rates <em>(short reads)</em> combined with</td>
<td>• High sequencing error rates <em>(long reads)</em> or</td>
</tr>
<tr>
<td>• Low genetic variation</td>
<td>• High genetic variation <em>(short or long reads)</em></td>
</tr>
</tbody>
</table>
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](GCCCAAATGGTT)

  ![K-mers](GCC, CCC)

  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>AAAAAAAAAAAAA</td>
</tr>
<tr>
<td>AAAAAAAAAAAAG</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>AAAAAAAAAAAAA</td>
</tr>
<tr>
<td>AAAAAAAAAAAAC</td>
</tr>
<tr>
<td>AAAAAAAAAAAT</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

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

#### Sorted Read Table

<table>
<thead>
<tr>
<th>Read</th>
<th>K-mer</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAAAAAAAAA</td>
<td>AAAAAAAAAAAAAAA</td>
</tr>
<tr>
<td>AAAAAAAAAAG</td>
<td>AAAAAAAAC</td>
</tr>
<tr>
<td>AAAAAAAAAACT</td>
<td>AAAAAAAAAT</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

#### Sorted K-mer Index

<table>
<thead>
<tr>
<th>K-mer</th>
<th>Read</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAAAAAAAAAA</td>
<td>AAAAAAAAAA</td>
</tr>
<tr>
<td>AAAAAAAAAAC</td>
<td>AAAAAAAAAG</td>
</tr>
<tr>
<td>AAAAAAAAAAT</td>
<td>AAAAAAAAACT</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

The Comparator compares the Read and the K-mer. If they match exactly, the read is filtered.

**Diagram:**

- **Next** from Read Table to Comparator
- **Next** from Comparator to K-mer Index
- **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>AAAAAAAAAA</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>AAAAAAAAAA</td>
</tr>
<tr>
<td>AAAAAAAAAAAC</td>
</tr>
<tr>
<td>AAAAAAAAAAT</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>Sorted K-mer Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAAAAAAA</td>
<td>AAAAAAAAAAAAAAA</td>
</tr>
<tr>
<td>AAAAAAAG</td>
<td>AAAAAAAAAC</td>
</tr>
<tr>
<td>AAAAAACT</td>
<td>AAAAAAAAAT</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Comparator

Read < K-mer

Not an exact match \(\rightarrow\) Send to read mapper
GenStore-EM: Not Finding a Match

Sorted Read Table

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

Sorted K-mer Index

Comparator

Read < K-mer

Not an exact match → Send to read mapper

✓ Avoids random accesses

✓ Simple low-cost logic
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
GenStore-EM: Design

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><strong>Seed Filtering (e.g., Chaining)</strong></td>
<td>Prune some seeds in the reference genome</td>
</tr>
<tr>
<td><strong>Alignment</strong></td>
<td>Determine the <strong>exact differences</strong> 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

![Probability vs Number of seeds per read diagram]

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

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

2.1x - 2.5x speedup compared to the software Base

1.5x – 3.3x speedup compared to the hardware Base

On average 3.92x 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 In-Storage Processing System for Genome Sequence Analysis

Session 6A: Thursday 3 March, 3:00 PM CEST

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 Ausavarunngnirun, Nandita Vijaykumar, Mohammed Alser, and Onur Mutlu
Backup Slides
End-to-End Workflow of Genome Sequence Analysis

- There are **three key initial steps** in a standard genome sequencing and analysis workflow:
  - Collection, preparation, and sequencing of a DNA sample in the laboratory
  - Basecalling
  - Read mapping

- Genomic read sets can be obtained by:
  - Sequencing a DNA sample and storing the generated read set into the SSD of a sequencing machine
  - Downloading read sets from publicly available repositories and storing them into an SSD

- We focus on optimizing the performance of read mapping because sequencing and basecalling are performed only once per read set, whereas read mapping can be performed many times:
  - Analyzing the differences between a reads from an individual and many reference genomes of other individuals
  - Repeating the read mapping step many times to improve the outcome of read mapping

- Improving read mapping performance is critical in almost all genomic analyses that use sequencing:
  - 45% of the execution time when discovering sequence variants in cancer genomics studies
  - 60% of the execution time when profiling the species composition of a multi-species (i.e., metagenomic) read
# Motivation

The table below shows the execution time in seconds for different configurations:

<table>
<thead>
<tr>
<th>Configuration</th>
<th>SSD-L</th>
<th>SSD-M</th>
<th>SSD-H</th>
<th>DRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>24.8</td>
<td>3.54</td>
<td>2.01</td>
<td>N/A</td>
</tr>
<tr>
<td>SW-filter</td>
<td>2.16</td>
<td>1.64</td>
<td>1.44</td>
<td>0.72</td>
</tr>
<tr>
<td>Ideal-ISF</td>
<td>10.1</td>
<td>1.44</td>
<td>1.44</td>
<td>N/A</td>
</tr>
<tr>
<td>ACC</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Ideal-ISF+ACC</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

The diagram illustrates the execution time for various configurations, comparing SSD-L, SSD-M, SSD-H, and DRAM. The bars represent the execution time in seconds for each configuration.
Motivation

State-of-the-art software read mapper, Minimap2

Base integrated with a software filter that prunes 80% of exactly-matching reads

Base integrated with an ideal in-storage filter
Motivation

Low-end SSD with SATA3 interface (0.5 GB/s)

High-end SSD with PCIe Gen4 interface (7 GB/s)

Data preloaded in DRAM, with no I/O overhead

Execution time [sec]

Base

SW-filter

Ideal-ISF

Low-end SSD with SATA3 interface (0.5 GB/s)

High-end SSD with PCIe Gen4 interface (7 GB/s)

Data preloaded in DRAM, with no I/O overhead
The ideal in-storage filter significantly improves performance by:

1) Reducing computation overhead
2) Reducing data movement overhead
Overheads of Software Mappers

I/O has a **significant impact** on application performance which can be alleviated at the cost of **expensive** storage devices and interfaces.
Overheads of Software Mappers

SW-filter provides limited benefits compared to Base

The filtering process outside the SSD must compete with the read mapping process for the resources in the system
Overheads of Hardware Mappers

Even the high-end SSD does not fully alleviate the storage bottleneck.

The ideal in-storage filter significantly improves performance.
Ideal-OSF

• Execution time of an **ideal in-storage filter**:

\[ T_{\text{Ideal-ISF}} = T_{\text{I/O-Ref}} + \max \{ T_{\text{I/O-Unfiltered}}, T_{\text{RM-Unfiltered}} \} \]

• Execution time of an **ideal outside-storage filter**:
  - 60% slower than Ideal-ISF in our analysis

\[ T_{\text{Ideal-OSF}} = T_{\text{I/O-Ref}} + \max \{ T_{\text{I/O-All-Reads}}, T_{\text{RM-Unfiltered}} \} \]
Comparison to PIM

- Even though read mapping applications could also benefit from other near-data, in-storage processing can fundamentally address the data movement problem by filtering large, low-reuse data where the data initially resides.

- Even if an ideal accelerator achieved a zero execution time, there would still exist the need to bring the data from storage to the accelerator.
  - 2.15x slower than the execution time that Ideal-ISF+ACC provides in our motivational analysis

In-storage filter can be integrated with any read mapping accelerator, including PIM accelerators, to alleviate their data movement overhead.
## Long Read Use Cases

<table>
<thead>
<tr>
<th>Use case</th>
<th>Input read set (Short/Long)</th>
<th>Size [GB]</th>
<th>Reference</th>
<th>Align [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequencing errors</td>
<td>ERR3988483 (L) [157]</td>
<td>54</td>
<td>hg38 [144]</td>
<td>47.4</td>
</tr>
<tr>
<td></td>
<td>HG002_ONT_20200204 (L) [158]</td>
<td>371</td>
<td></td>
<td>69.3</td>
</tr>
<tr>
<td>Rapidly evolving</td>
<td>SRR5413248 (L) [157]</td>
<td>1.69</td>
<td>NZ_NJEX02 [159]</td>
<td>60.0</td>
</tr>
<tr>
<td>samples</td>
<td>SRR12423642 (S) [157]</td>
<td>0.466</td>
<td>NC_045512.2 [160]</td>
<td>23.1</td>
</tr>
<tr>
<td>No reference</td>
<td>SRR6767727 (L) [157]</td>
<td>12.4</td>
<td>NZ_NJEX02 [159]</td>
<td>0.35</td>
</tr>
<tr>
<td></td>
<td>SRR9953689 (L) [157]</td>
<td>15.9</td>
<td></td>
<td>37.0</td>
</tr>
<tr>
<td>Contamination</td>
<td>SRR9953689 (L) [157]</td>
<td>15.9</td>
<td>hg38 [144]</td>
<td>1.0</td>
</tr>
</tbody>
</table>
FTL: Metadata

• GenStore metadata includes the mapping information of the data structures necessary for read mapping acceleration

• In accelerator mode, GenStore also keeps in internal DRAM other metadata structures of the regular FTL
  - Examples include the page status table and block read counts which need to be updated during the filtering process

• We carefully design GenStore to only sequentially access the underlying NAND flash chips while operating as an accelerator
  - Requires only a small amount of metadata to access the stored data
FTL: Data Placement

• GenStore needs to properly place its data structures to enable the full utilization of the internal SSD bandwidth

• When each data structure is initially written to the SSD, GenStore sequentially and evenly distributes it across NAND flash chips

• GenStore can specify the physical location of a 30-GB data structure by maintaining only the list of 1,250 (30 GB/24 MB) physical block addresses

• It significantly reduces the size of the necessary mapping information from 300 MB (with conventional 4-KiB page mapping) to only 5 KB (1,250 4 bytes)
FTL: SSD Management Tasks

• In accelerator mode, GenStore only reads data structures to perform filtering, and does not write any new data
  - GenStore does not require any write-related SSD-management tasks such as garbage collection and wear-leveling

• The other tasks necessary for ensuring data reliability can be done before or after the filtering process
  - GenStore significantly limits the amount of data whose retention age would exceed the manufacturer-specified threshold since GenStore’s filtering process takes a short time.
  - GenStore-FTL can easily avoid read disturbance errors for data with high read counts since GenStore sequentially reads NAND flash blocks only once during filtering
Data Sizes

• Conventional k-mer index in Minimap2 + reference genome: 7 GB (k = 15)

• Read-sized k-mer index before optimization: 126 GB (k = 150)

• Read-sized k-mer index after optimization: 32 GB (k = 150)
SSD Specs

- **SSD-L**: SATA3 interface (0.5 GB/s sequential read)
  - 1.2 GB/s per channel bandwidth
  - 8 channels

- **SSD-L**: PCIe Gen3 M.2 interface (3.5 GB/s sequential read)
  - 1.2 GB/s per channel bandwidth
  - 16 channels

- **SSD-L**: PCIe Gen4 interface (7 GB/s sequential read)
  - 1.2 GB/s per channel bandwidth
  - 16 channels
Evaluation Methodology

• **Performance modeling**
  - Ramulator for DRAM timing
  - MQSim for SSD timing
  - We model the end-to-end throughput of GenStore based on the throughput of each GenStore pipeline stage
    • Accessing NAND flash chips
    • Accessing internal DRAM
    • Accelerator computation
    • Transferring unfiltered data to the host

• **Real system results**
  - AMD EPYC 7742 CPU
  - 1TB DDR4 DRAM
  - AMD μProf
GenStore-NM

GenStore-Enabled SSD

1. Seed Finder
   - K-mer Window
   - Hash Acc.
   - Location Buffer

2. Seed Count-Based Filter
   - Filters if # of Seeds < M

3. Chaining-Based Filter
   - Filters low-score reads

# of Seeds ≥ N

High chaining score

M ≤ # of Seeds < N

Input Read Set

Flash Array

KmerIndex

DRAM

Host System

① Reads

② K-mers

③ Query

④ Seeds

⑤ Seeds

⑥ Query

⑦ K-mer Window

⑧ Hash Acc.

⑨ Location Buffer

⑩ SSD Controller

SAFARI
Chaining Processing Element

Chaining-Based Filter
GS-Ext provides significant performance improvements over both Base and SIMD in SSD-M and SSD-H.

GS-Ext provides limited benefits over SIMD in SSD-L due to low external I/O bandwidth.
GS-Ext performs significantly slower than Base (2.28x - 1.91x) on all systems.
Effect of Inputs on GenStore-EM

\[ DM\_Saving = \frac{Size_{Ref} + Size_{ReadSet}}{Size_{Ref} + Size_{ReadSet} \times (1 - Ratio_{Filter})} \]
Effect of Inputs on GenStore-NM

\[ DM_{\text{Saving}} = \frac{Size_{\text{Ref}} + Size_{\text{ReadSet}}}{Size_{\text{Ref}} + Size_{\text{ReadSet}} \times (1 - Ratio_{\text{Filter}})} \]