GRIM-Filter: Fast Seed Location Filtering in DNA Read Mapping Using Processing-in-Memory Technologies

Jeremie S. Kim, Damla Senol Cali, Hongyi Xin, Donghyuk Lee, Saugata Ghose, Mohammed Alser, Hasan Hassan, Oguz Ergin, Can Alkan, Onur Mutlu

BMC Genomics 2018

*Presented by Jakob Genhart*
Executive Summary

Motivation
Speed up DNA-sequencing for preventative and personalized medicine.

Problem
DNA-sequencing is mainly bottlenecked by read mapping, an approximative string matching problem.

Key Idea
Improve performance of read mapping by avoiding redundant alignment using new PIM technologies.

Solution
A novel seed location filter utilizing 3D-stacked memory and PIM to speed up and improve filtering.

Results
Compared to a state-of-the-art read mapper mrFAST with FastHASH
Speedup: 2.08x (1.81x - 3.65x)  Lower false negative rate: 5.97x (5.59x – 6.41x)
### Outline

| Section          |
|------------------|
| Background       |
| GRIM-Filter      |
| Evaluation       |
| Conclusion       |
| Discussion       |
Background: DNA

- DNA is a sequence of A C G T bases
- Human genome consists of 3.2 billion bases pairs (bp)
- Current sequencing machines will break genome into reads of 100 – 20k bp
Problem: Alignment is expensive

Observation: Most locations don’t match

Idea: Discard locations early
Background: Sequencing

GRIM-Filter
Other filters: GateKeeper, ShiftedHammingDistance
Background: 3D-Stacked RAM

- High internal bandwidth
- Logic layer enables PIM
- High bandwidth over interposer
- Typical size 16GB (2018)

- Banks are organized into vaults
  - Read one row buffer per vault at a time
- Highly parallelizable memory access

GRIM-Filter can utilize this!
| Outline               |
|----------------------|
| Background           |
| GRIM-Filter          |
| Evaluation           |
| Conclusion           |
| Discussion           |
GRIM-Filter: Outline

Precomputation

Filtering

3D-Stacked RAM
GRIM-Filter: Outline

- Precomputation
- Filtering
- 3D-Stacked RAM
GRIM-Filter: Precomputation

Reference Genome

- Split reference genome into **bins**
- Compute **bit vector** for each bin
  - Bits indicate presence of **token**
  - $4^n$ bits long

Bin 1: TAGCCCAAGACTTCCCGTGTCCTTTCCACCGGGCCTTTGAGAGGTCACAGGGTCTTGATGAGTCAGAA...
Bin 2: ...
Bin 3: ...
Bin 4: ...

Token

- **AAAA** not present
- **CCCA** is present

**Split reference genome into bins**

**Compute bit vector for each bin**

- Bits indicate presence of token
- $4^n$ bits long

**Example:**

- **AAAA** not present
- **CCCA** is present
GRIM-Filter: Usage Of Bins

Reference Genome

TAGCCCAAGACTTCCCGTGTCCCTTTCCACCGGGCCTTTGAGAGGTCACAGGGTCTTGATGAGTCAGAA...

Bin 1

Bin 2 ✓

GTCCTT

Bin 3

GTCCTT

Bin 4 ✗

➢ Compare read with bit vector of possible bins

➢ Read must be completely in bin
  ▪ Bins must overlap by read size

b2

0

AAAA
AAAC
AAAG
AAAT

0

CCTG
CCTT

0

TTTT
GRIM-Filter: Outline

- Precomputation
- Filtering
- 3D-Stacked RAM
GRIM-Filter: Filtering Mechanism

Read

AGCGCAGGCTCGCAGCCTGTAGATT...

AGCG
GCGC
CGCA
GCAC
CACG
ACGG
CGGG
GGGG
...

\[ b_2 \]

0
0
1
0
0
0
1
1

Keep

Discard

\[ \text{Sum}_z \geq \text{Threshold} \]

False negative?
- **Ok!** Should be low

False positive?
- **Bad!** Must be 0%

➢ Result is store in seed location filter bitmask
GRIM-Filter: Sum Threshold

How to achieve 0% false positive rate?

- Alignment has an error tolerance $e \approx 0.05$
- More errors result in lower $Sum_z$
- **Overestimate** number of errors in reads just passing alignment
- Reads passing alignment must have less errors and will have $Sum_z \geq \text{Threshold}$

\[
\text{Threshold} = \text{len} - (n - 1) - n \times \lceil \text{len} \times e \rceil
\]

- Number of tokens in read
- Maximum number of token affected by error
- Maximum number of errors

AGCGCAC
GRIM-Filter: Read Mapper Integration

**Input:** Read

TAGCCCAAGACTTCCCGTGTCTTTTCCACCAGG...

**GRIM-Filter:**
Filter Bitmask Generator

0110101011010101010100001011

Seed Location Filter Bitmask

020128  020131  414415

**GRIM-Filter:**
Seed Location Checker

011010101010100100001011

**Read Mapper:**
Indexing & Seeding

**Read Mapper:**
Sequence Alignment

**Output:** Correct Mappings

Processing-in-memory
GRIM-Filter: Outline

- Precomputation
- Filtering
- 3D-Stacked RAM
GRIM-Filter: Mapping To 3D-Stacked RAM

➢ GRIM-Filter fits 3D-Stacked Memory well
  ▪ Only simple operations
  ▪ Bins can be checked in parallel

➢ Allow access to same token of many bins in parallel

➢ Simple hardware for each bin
  ▪ 4096 incrementor LUTs, 7-bit counters, and comparators

Uses only 3.8 GB
Evaluation: Methodology

- **Design parameters:**
  - Token size of 5
  - Bin count of $450 \times 2^{16}$

- **Simulated** using in-house 3D-Stacked DRAM simulator

- Evaluated on 10 real-world datasets
  - From 1000 Genomes Project
  - Reads of length 100

- Evaluated 2 key metrics
  - **Execution Time**
  - **False Negative Rate**

- Compared against state-of-the-art read mapper mrFAST with FastHASH
Evaluation: Results

➢ **2.08x (1.81x - 3.65x)** faster than mrFAST with FastHASH

![Execution Time Chart]

➢ **5.97x (5.59x – 6.41x)** lower false negative rate than FastHASH

![False Negative Rate Chart]
Executive Summary

Motivation
Speed up DNA-Sequencing for preventative and personalized medicine.

Problem
DNA-Sequencing is mainly bottlenecked by read mapping, an approximative string matching problem.

Key Idea
Improve performance of read mapping by avoiding redundant alignment using new PIM technologies.

Solution
A novel seed location filter utilizing 3D-stacked memory and PIM to speed up and improve filtering.

Results
Compared to a state-of-the-art read mapper mrFAST with FastHASH
Speedup: 2.08x (1.81x - 3.65x)  Lower false negative rate: 5.97x (5.59x – 6.41x)
Conclusion: Strengths

➢ **Novel idea**: use 3D-Stacked Memory for pre-alignment filtering

➢ GRIM-Filter is **well suited for PIM** and 3D-Stacked Memory

➢ **Orthogonal** to other attempts of improving read mappers
  ▪ Can be used in combination to achieve even better performance!

➢ Design parameter space is explored, and final **decisions are explained**

➢ Code is open source
Conclusion: Weaknesses

➢ GRIM-Filter is only tested with **short reads**
  - Limited to **Illumina** machines (150bp – 300bp)
  - **PacBio** (15kbp – 20kpb)
  - **ONT** (up to 4Mbp)

➢ Some information is **never mentioned**
  - **Bin size** is not mentioned, only bin count
  - **Exact memory layout** i.e., row buffer size, vault count

➢ Data used for graphs is not publicly available
  - Hard to read exact results from small graphs

➢ Minor mistakes
Conclusion: Minor Mistakes

errors. Figure 5a shows the equation that we use to calculate the threshold while accounting for errors. As shown in Figure 5b, a token of size $n$ in a bin overlaps with $n - 1$ other tokens. We calculate the lowest $\text{Sum}_x$ possible for a sequence alignment that includes only a single error (i.e., one insertion, deletion, or substitution) by
Outline

- Background
- GRIM-Filter
- Evaluation
- Conclusion
- Discussion
Discussion: Memory Usage & Long Reads

**memory use = bin count \times bit vector size**

- **Bin count constant**
- **Bin length increases** with read length
  - Bins must overlap by read size

| Read Length | Bin Length |
|-------------|------------|
| 100         | \approx 208 |
| 1'000       | \approx 1'108 |
| 20'000      | \approx 20'108 |

Ability to filter decreases with increasing bin size!

Bin of size 100

AGCGCACGCT...

97 tokens

\[ b_2 = \begin{array}{c}
0 \\
0 \\
\vdots \\
1 \\
0 
\end{array} \]

256 bits

Bin of size 1000

AGCGCACGGGGTCGGCGGCGTGGAATT...

997 tokens

\[ b_2 = \begin{array}{c}
1 \\
1 \\
\vdots \\
1 \\
1 
\end{array} \]

256 bits

Mostly ones!
Discussion: Memory Usage & Long Reads

memory use = bin count \times bit vector size

- **Bin count constant**
- **Bin length increases** with read length
  - Bins must overlap by read size

| Read Length | Bin Length |
|-------------|------------|
| 100         | \approx 208 |
| 1'000       | \approx 1'108 |
| 20'000      | \approx 20'108 |

- **Ability to filter decreases** with increasing bin size!

- **Increase token length** to keep expected bit vector occupancy constant

| Read Length | Token Length | Memory Used | Occupancy |
|-------------|--------------|-------------|-----------|
| 100         | 5            | 3.77        | 3.29      |
| 1'000       | 6            | 15.1        | 4.39      |
| 5'000       | 7            | 60.4        | 4.35      |
| 20'000      | 8            | 241.59      | 3.76      |
Discussion: How to deal with long reads?

➢ Idea 1: **Split long reads** into smaller reads
  ▪ Solves memory problem
  ▪ Loss of information

**Nanopore long-read RNAseq reveals widespread transcriptional variation among the surface receptors of individual B cells**

Ashley Byrne, Anna E. Beaudin, Hugh E. Olsen, Miten Jain, Charles Cole, Theron Palmer, Rebecca M. DuBois, E. Camilla Forsberg, Mark Akeson & Christopher Vollmers

*Nature Communications* 8, Article number: 16027 (2017) | Cite this article

“Short-read RNAseq is limited in its ability to resolve complex isoforms because it fails to sequence full-length cDNA copies of RNA molecules.”
Discussion: How to deal with long reads?

➢ Idea 1: **Split long reads** into smaller reads
  ▪ Solves memory problem
  ▪ **Loss of information**

➢ Idea 2: **Accept Tradeoffs**
  ▪ Higher **memory consumption** / lower **false negative rate**
  ▪ More **logic die space** consumption
  ▪ Longer **filtering time**

➢ Other ideas?
Discussion: Repetitive Memory Access

Read

- Load row buffer for each token in read
  - $O(\text{read len})$ memory accesses
- Might be more than distinct token count
  - Especially in repetitive DNA
Discussion: Avoiding Repetitive Memory Access

➢ Idea 1: **Count tokens first**, calculate sum later
  ▪ One memory access for each token appearing more than once
  ▪ Needs **more specialized hardware**
    ▪ Registers for token counts
    ▪ Full adders instead of incrementors
    ▪ Tokenization can be **parallelized** with summation

➢ Other ideas?
GRIM-Filter: Fast Seed Location Filtering in DNA Read Mapping Using Processing-in-Memory Technologies

Jeremie S. Kim, Damla Senol Cali, Hongyi Xin, Donghyuk Lee, Saugata Ghose, Mohammed Alser, Hasan Hassan, Oguz Ergin, Can Alkan, Onur Mutlu

BMC Genomics 2018

Presented by Jakob Genhart

Thank you!
Backup: Seed And Extend Mapper
Backup: Bins & Bitvectors

(a) Reference Genome

(b) Tokens

```
| token | b_1 | b_2 |
|-------|-----|-----|
| AAAA  | 1   | 0   |
| AAAA  | 1   | 1   |
| AAAAC | 0   | 0   |
| AAAG  | 0   | 0   |
| AAGAG | 0   | 0   |
| AGAAA | 0   | 1   |
| AGAAA | 1   | 1   |
| GAAAA | 1   | 0   |
| GACAG | 1   | 1   |
| GCATG | 0   | 1   |
| GCATG | 1   | 0   |
| TTTGA | 1   | 0   |
| TTTCA | 1   | 1   |
| TTTTG | 1   | 1   |
| TTTTT | 0   | 0   |
```

Length = 4^5

- GACAG exists in 2nd bin
- TTTTT doesn’t exist in 2nd bin

* $t$ = number of bins
Backup: Sum Threshold

**INPUT:** Read Sequence $r$

GAACTTGAGTCACGAG ... GTACGATT

1. Read bitvector for bin_num(z)

2. $\sum_{z} \geq \text{Threshold?}$

3. Decision: NO → Discard
   YES → Send to Read Mapper for Sequence Alignment
Backup: GRIM-Filter Integration

1. **INPUT:** Read Sequence
   GAACTTGGAGTCTACGAG ... GTACGATT

2. **GRIM-Filter:**
   Filter Bitmask Generator
   
   (see Figure 3)

   ... 0001010001110001010 ... 010011010 ...

   **Seed Location Filter Bitmask**

3. **INPUT:** All Potential Seed Locations
   ... 020128 ... 020131 ... 414415 ...

   **GRIM-Filter:**
   Seed Location Checker

   ... 0001010001110001010 ... 010011010 ...

   **Reference Segment Storage**

4. **Read Mapper:**
   Sequence Alignment

   reference segment @ 020131
   ... 414415

   **Edit-Distance Calculation**

   **OUTPUT:** Correct Mappings
Increasing token size reduces average read existence with all bin sizes
Token size > 5 has diminishing returns
Backup: Design Decisions

- Bin count $>300 \times 2^{16}$ has diminishing returns
- Linearly decreasing memory footprint
- Thus $450 \times 2^{16}$ bins was chosen
Backup: Results