GRIM-Filter:
Fast seed location filtering in DNA read mapping using processing-in-memory technologies

Published in BMC Genomics 2018

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

Presented by Alexander Frey
General Outline

- Summary of the paper
- Strength
- Weaknesses
- Discussion
Genome Read Mapping is a very important problem and is the first step in genome analysis.

Read Mapping is an approximate string matching problem:
- Find the best fit of 100 character strings into a 3 billion character dictionary
- Alignment is currently the best method for determining the similarity between two strings, but is very expensive.

We propose an algorithm called GRIM-Filter:
- Accelerates read mapping by reducing the number of required alignments
- GRIM-Filter can be accelerated using processing-in-memory
  - Adds simple logic into 3D-Stacked memory
  - Uses high internal memory bandwidth to perform parallel filtering

GRIM-Filter with processing-in-memory delivers a 3.7x speedup.
## GRIM-Filter Outline

1. Motivation and Goal
2. Background: Read Mappers
   a. Hash Table Based
   b. Hash Table Based with Filter
3. Our Proposal: GRIM-Filter
4. Mapping GRIM-Filter to 3D-Stacked Memory
5. Results
6. Conclusion
# GRIM-Filter Outline

1. Motivation and Goal

2. Background Read Mappers
   - a. Hash Table Based
   - b. Hash Table Based with Filter

3. Our Proposal: GRIM-Filter

4. Mapping GRIM-Filter to 3D-Stacked Memory

5. Results

6. Conclusion
Motivation and Goal

- **Sequencing**: determine the [A,C,G,T] series in DNA strand

- Today’s machines sequence short strands (**reads**)
  - Reads are on the order of 100 –2M base pairs (**bp**)
  - The human genome is approximately 3 billion **bp**

- Therefore genomes are cut into reads, which are sequenced independently, and then reconstructed
  - **Read mapping** is the first step in analyzing someone’s genome to detect predispositions to diseases, personalize medicine, etc.

- **Goal**: We want to **accelerate** end-to-end performance of **read mapping**
# GRIM-Filter Outline

1. Motivation and Goal

2. Background: Read Mappers
   - Hash Table Based
   - Hash Table Based with Filter

3. Our Proposal: GRIM-Filter

4. Mapping GRIM-Filter to 3D-Stacked Memory

5. Results

6. Conclusion
Background: Read Mappers

We now have **sequenced reads** and want a **full genome** via Read Mapping:

We map **reads** to a known **reference genome** (>99.9% similarity across humans) with some minor errors allowed:

Because of high similarity, long sequences in **reads** perfectly match in the **reference genome**:

```
G A C T G T G T C A A
✔ ✔ ✔ ✔ ✔ ✔ ✔ ✔ ✔
✘ ✔ ✔ ✔ ✔ ✔ ✔ ✔ ✔
```

```
... G A C T G T G T C G A ...
```

We can use a hash table to help quickly map the **reads**!
# GRIM-Filter Outline

1. Motivation and Goal

2. **Background: Read Mappers**
   - **a. Hash Table Based**
   - **b. Hash Table Based with Filter**

3. Our Proposal: GRIM-Filter

4. Mapping GRIM-Filter to 3D-Stacked Memory

5. Results

6. Conclusion
Generating Hash Tables

To map any reads, generate a **hash table** per **reference genome**.

- k-length sequences (k-mers)
  - AAAA
    - Location list where k-mer occurs in the reference genome:
      - @36: AAAAT
      - @434: AAAAAT
  - AAAAC
    - Location list:
      - @13: 609 788
  - AAAAT
    - Location list:
      - @36: 434
  - GGGGG
    - Location list:
      - @52: 67 334 634 851

We can query the table with substrings from reads to quickly find a list of possible mapping locations.
Hash Tables in Read Mapping

Read Sequence (100 bp)

99.9% of locations result in a mismatch

We want to **filter these out** so we do not waste time trying to align them
Location Filtering

- **Alignment** is expensive and requires the use of $O(n^2)$ dynamic programming algorithm
  - We need to align millions to billions of reads

- Modern read mappers reduce the time spent on alignment for increased performance. Can be done in two ways:
  1. Optimize the algorithm for alignment
  2. Reduce the number of alignments necessary by filtering out mismatches quickly

- Both methods are used by mappers today, but filtering has replaced alignment as the bottleneck [Xin+, BMC Genomics 2013]

Our goal is to accelerate read mapping by improving the **filtering** step
## GRIM-Filter Outline

1. Motivation and Goal

2. **Background: Read Mappers**
   - a. Hash Table Based
   - b. Hash Table Based with Filter

3. Our Proposal: GRIM-Filter

4. Mapping GRIM-Filter to 3D-Stacked Memory

5. Results

6. Conclusion
Hash Tables in Read Mapping

Read Sequence (100 bp)

Hash Table

Reference Genome

Filter

Match!

Mismatch

False Negative

SAFARI
# GRIM-Filter Outline

1. Motivation and Goal

2. Background: Read Mappers
   - a. Hash Table Based
   - b. Hash Table Based with Filter

3. Our Proposal: GRIM-Filter

4. Mapping GRIM-Filter to 3D-Stacked Memory

5. Results

6. Conclusion
Our Proposal: GRIM-Filter

1. **Data Structures: Bins & Bitvectors**

2. Checking a Bin

3. Integrating GRIM-Filter into a Mapper
GRIM-Filter: Bins

- We partition the genome into large sequences (**bins**).

<table>
<thead>
<tr>
<th>Bin x - 3</th>
<th>Bin x - 2</th>
<th>Bin x - 1</th>
<th>Bin x</th>
</tr>
</thead>
</table>

... GGAAATACGTTCAGTCAGTTGGAAATACGTTTTGGGCCTACTTCTCAGTACGTACAGTACAGTAAAAATGACAGTAAGAC ...

- Represent each bin with a **bitvector** that holds the occurrence of all permutations of a small string (**token**) in the bin.

- To account for matches that straddle bins, we employ overlapping bins.
  - A read will now always completely fall within a single bin.

### Bitvector

<table>
<thead>
<tr>
<th>Token</th>
<th>Bin x</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAAAA</td>
<td>1</td>
</tr>
<tr>
<td>AAAAC</td>
<td>0</td>
</tr>
<tr>
<td>AAAAT</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>CCCCC</td>
<td>1</td>
</tr>
<tr>
<td>CCCCT</td>
<td>0</td>
</tr>
<tr>
<td>CCCCCG</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>GGGGG</td>
<td>1</td>
</tr>
</tbody>
</table>

**AAAAA** exists in bin x

**CCCCT** doesn’t exist in bin x

**SAFARI**
GRIM-Filter: Bitvectors

Bin x

AAAAA
CGTGA
TGAGT
GAGTC
GTGAG
...

SAFARI
**GRIM-Filter: Bitvectors**

Storing all bitvectors requires $4^n \times t$ bits in memory, where $t = \text{number of bins}$.

For bin size $\sim 450$, and $n = 5$, memory footprint $\sim 3.8$ GB

<table>
<thead>
<tr>
<th>Reference Genome</th>
<th>$b_1$</th>
<th>$b_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAAAA</td>
<td>1</td>
<td>AAAAA</td>
</tr>
<tr>
<td>AAAAC</td>
<td>1</td>
<td>AAAAC</td>
</tr>
<tr>
<td>AAAAG</td>
<td>0</td>
<td>AAAAG</td>
</tr>
<tr>
<td>AAAAT</td>
<td>0</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>CCCCT</td>
<td>1</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>GACAG</td>
</tr>
<tr>
<td>.</td>
<td>GCATG</td>
<td>1</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>TTTTT</td>
<td>0</td>
</tr>
</tbody>
</table>
Our Proposal: GRIM-Filter

1. Data Structures: Bins & Bitvectors
2. Checking a Bin
3. Integrating GRIM-Filter into a Mapper
GRIM-Filter: Checking a Bin

How GRIM-Filter determines whether to discard potential match locations in a given bin prior to alignment

**INPUT:** Read Sequence \( r \)

\[
\text{GAACTTGGAGTCTA} \ldots \text{CGAG}
\]

1. Get tokens
2. Read bitvector for bin_num(\( x \))
3. Match tokens to bitvector
4. Sum
5. Compare

\[
\begin{array}{cccccc}
1 & 0 & 1 & \ldots & 0 & 1 \\
0 & 1 & 1 & \ldots & 1 & 0 \\
\end{array}
\]

\[
\geq \text{Threshold?}
\]

NO → Discard

YES → Send to Read Mapper for Sequence Alignment
Our Proposal: GRIM-Filter

1. Data Structures: Bins & Bitvectors
2. Checking a Bin
3. Integrating GRIM-Filter into a Mapper
GRIM-Filter: Error Tolerance

Threshold = \( \text{read_length} - (n-1) - n \times \left\lfloor \text{read_length} \times e \right\rfloor \)

- total number of tokens in a read
- maximum number of tokens that could contain errors
- number of errors allowed per read
- single substitution error
- tokens affected by the error
- one substitution error affects four tokens when \( n = 4 \)

GRIM-Filter can support different error tolerances by simply changing the threshold value

More details in the paper
Our Proposal: GRIM-Filter

1. Data Structures: Bins & Bitvectors
2. Checking a Bin
3. Integrating GRIM-Filter into a Mapper
Integrating GRIM-Filter into a Read Mapper

1. **GRIM-Filter: Filter Bitmask Generator**
   - **INPUT:** Read Sequence
     - GAACCTTGCGAG ... GTATT
   - **OUTPUT:** Seed Location Filter Bitmask
     - ...0001010 ...011010 ...

2. **GRIM-Filter: Seed Location Checker**
   - **INPUT:** All Potential Seed Locations
     - ...020128 ... 020131 ... 414415 ...
   - **Bitmask:** ...
   - **KEEP:** ...
   - **KEEP:** ...
   - **DISCARD:**
     - X

3. **Reference Segment Storage**
   - **reference segment @ 020131**
   - **reference segment @ 414415**

4. **Read Mapper: Sequence Alignment**
   - **INPUT:** Seed Location Filter Bitmask
   - **OUTPUT:** Correct Mappings

**SAFARI**
# GRIM-Filter Outline

1. Motivation and Goal

2. Background: Read Mappers
   - a. Hash Table Based
   - b. Hash Table Based with Filter

3. Our Proposal: GRIM-Filter

4. Mapping GRIM-Filter to 3D-Stacked Memory

5. Results

6. Conclusion
Key Properties of GRIM-Filter

1. **Simple Operations:**
   - To check a given bin, find the sum of all bits corresponding to each token in the read
   - **Compare** against threshold to determine whether to align

2. **Highly Parallel:** Each bin is operated on independently and there are many many bins

3. **Memory Bound:** Given the frequent accesses to the large bitvectors, we find that GRIM-Filter is memory bound

These properties together make GRIM-Filter a good algorithm to be run in 3D-Stacked DRAM

SAFARI
3D-Stacked Memory

- 3D-Stacked DRAM architecture has **extremely high bandwidth** as well as a stacked customizable logic layer
  - Logic Layer enables Processing-in-Memory, offloading computation to this layer and alleviating the memory bus
  - Embed GRIM-Filter operations into **DRAM logic layer** and appropriately distribute bitvectors throughout memory
3D-Stacked Memory

- 3D-Stacked DRAM architecture has **extremely high bandwidth** as well as a stacked customizable logic layer.
  - Logic Layer enables **Processing in Memory**, offloading computation to this layer and alleviating the memory bus.
  - Embed GRIM-Filter operations into DRAM logic layer and appropriately distribute bitvectors throughout memory.

3D-Stacked Memory

Micron’s HMC

Micron has working demonstration components


http://images.anandtech.com/doci/9266/HBMCar_678x452.jpg
GRIM-Filter in 3D-Stacked DRAM

- Each DRAM layer is organized as an array of banks
  - A bank is an array of cells with a row buffer to transfer data

- The layout of bitvectors in a bank enables filtering many bins in parallel
GRIM-Filter in 3D-Stacked DRAM

- Customized logic for accumulation and comparison per genome segment
  - Low area overhead, simple implementation
  - For HBM2, we use 4096 incremeneter LUTs, 7-bit counters, and comparators in logic layer
1. Motivation and Goal

2. Background: Read Mappers
   a. Hash Table Based
   b. Hash Table Based with Filter

3. Our Proposal: GRIM-Filter

4. Mapping GRIM-Filter to 3D-Stacked Memory

5. Results

6. Conclusion
Methodology

- Performance simulated using an in-house 3D-Stacked DRAM simulator

- Evaluate 10 real read data sets (From the 1000 Genomes Project)
  - Each data set consists of 4 million reads of length 100

- Evaluate two key metrics
  - Performance
  - False negative rate
    - The fraction of locations that pass the filter but result in a mismatch

- Compare against a state-of-the-art filter, FastHASH [Xin+, BMC Genomics 2013] when using mrFAST, but GRIM-Filter can be used with ANY read mapper
GRIM-Filter Performance

Benchmarks and their Execution Times

- FastHASH filter
- GRIM-Filter

**1.8x-3.7x performance benefit across real data sets**

**2.1x average performance benefit**

GRIM-Filter gets performance due to its hardware-software co-design
GRIM-Filter False Negative Rate

Benchmarks and their False Negative Rates

- FastHASH filter
- GRIM-Filter

Sequence Alignment Error Tolerance $e$

$e = 0.05$

5.6x-6.4x False Negative reduction across real data sets

6.0x average reduction in False Negative Rate

GRIM-Filter utilizes more information available in the read to filter
Other Results in the Paper

- Sensitivity of execution time and false negative rates to error tolerance of string matching
- Read mapper execution time breakdown
- Sensitivity studies on the filter
  - Token Size
  - Bin Size
  - Error Tolerance
<table>
<thead>
<tr>
<th>1. Motivation and Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. Background: Read Mappers</td>
</tr>
<tr>
<td>a. Hash Table Based</td>
</tr>
<tr>
<td>b. Hash Table Based with Filter</td>
</tr>
<tr>
<td>3. Our Proposal: GRIM-Filter</td>
</tr>
<tr>
<td>4. Mapping GRIM-Filter to 3D-Stacked Memory</td>
</tr>
<tr>
<td>5. Results</td>
</tr>
<tr>
<td>6. Conclusion</td>
</tr>
</tbody>
</table>
Conclusion

We propose an in-memory filtering algorithm to accelerate end-to-end read mapping by reducing the number of required alignments.

Key ideas:
- Introduce a new representation of coarse-grained segments of the reference genome.
- Use massively-parallel in-memory operations to identify read presence within each coarse-grained segment.

Key contributions and results:
- Customized filtering algorithm for 3D-Stacked DRAM.
- Compared to the previous best filter:
  - We observed 1.8x-3.7x read mapping speedup.
  - We observed 5.6x-6.4x fewer false negatives.

GRIM-Filter is a universal filter that can be applied to any read mapper.
General Outline

- Summary of the paper
- Strength
- Weaknesses
- Discussion
Strength

- We have a novel idea to improve DNA read mapping
- Could be used if PIM was possible today
- Sensitivity is changeable if needed (different applications possible)
- Good for high error
- Giving background information and future research ideas
- The evaluation of GRIM is clearly defined and explained
- open-source
General Outline

- Summary of the paper
- Strength
- Weaknesses
- Discussion
Weaknesses

- GRIM is evaluated on a Hashing-based read mapper
- How do we input/output data while computing
- In the runtime results we do not count the transformation from reference genome to bit-vector.
General Outline

- Summary of the paper
- Strength
- Weaknesses
- Discussion
Discussion

- Is there a possibility to have a smaller memory footprint?
Discussion

- Is there a possibility to have a smaller memory footprint?
  - Yes, we can take every $n$ token in the reference Genome and multiply the length of the bin span by $n$.
  - This move could impact our false positive rate of our filter. We can do this since the bit-vector density remains unchanged.
Discussion

- How could we mitigate the problems of slightly longer input parameters or inconsistent lengths?
Discussion

- How could we mitigate the problems of slightly longer input parameters or inconsistent lengths?
  - A small change in the length of the read would not impact Grim since we simply change the threshold.
  - Another possibility, if the read gets long, is to change the bit-vector span.
  - If the length of the read is too long for all other changes, we can cut it in half.
Discussion

- Is there a possibility of implementing GRIM efficiently in GPU/FPGA?
Discussion

- Is there a possibility of implementing GRIM efficiently in GPU/FPGA?
  - It is highly unlikely that GRIM would work in a reasonable amount of time since we need a high bandwidth which the FPGA / GPU cannot support.
  - Be 400 GB/s bandwidth (bus bandwidth not internal bandwidth, I assume the bandwidth similar and that we run on 4 Mherz)
    - =>838 bits per cycle
    - how do we store the bits efficiently in the GPU?
Discussion

- With long reads is Grim a good option in this configuration? If not could we change it?
Discussion

- With long reads is Grim a good option in this configuration? If not could we change it?
  - No it is not we would need around 70 Banks to hold 1 read (286 720 Banks required if we want to use the 4096 bandwidth).
  - What could we change? we could change the interconnection between bins, change the span of bins, and allow for other filtering possibilities.
Discussion

- Can we change the *fundamental working procedure* to accommodate for long reads?
Discussion

- Can we change the fundamental working procedure to accommodate for long reads?
  - We could change the filtering system of Grim, we know that our DNA has a specific order of garbage and useful DNA sequences.
    - If we know these in the reference Genome and our genome with the locations we found we could filter some of our locations out since their pattern of useful and garbage DNA is not the one of our read
    - In conjunction with the first 1,000 -10,000 bp we could find reasonable amount locations.
  - Another possibility is the cut the reads
    - save their location in relation to each other and use smaller reads combined with CKS/AF we could filter more thoroughly
    - Another possibility is to allow long reads to be cut in multiple pieces and to use Grim filter on those shorter reads. We could allow for independent or dependent thresholds.
    - => we would need 4 GRIM filters for a cut which results in 4 smaller read
Discussion

- Can we speedup GRIM?
Discussion

- Can we speedup GRIM?
  - Since GRIM speed is dependent on the bandwidth of 3D-stacked memory it would be possible to speed up Grim by increasing the bandwidth of the 3D-stacked memory
  - Another possibility could be to check every \( n \) token from the input \texttt{read(token)}
  - Another possibility would be the balancing of the runtimes of GRIM and of read alignment since currently the bottleneck is GRIM.