Accelerating Genome Analysis

A Primer on an Ongoing Journey

Onur Mutlu
omutlu@gmail.com
https://people.inf.ethz.ch/omutlu
16 February 2019
AACBBB Keynote Talk
Overview

- **System design for bioinformatics** is a critical problem
  - It has large scientific, medical, societal, personal implications

- This talk is about accelerating a key step in bioinformatics: genome sequence analysis
  - In particular, read mapping

- Many bottlenecks exist in accessing and manipulating huge amounts of genomic data during analysis

- We will cover various recent ideas to accelerate read mapping
  - My personal journey since September 2006
Our Dream (in 2007)

- An embedded device that can perform comprehensive genome analysis in real time (within a minute)
  - Which of these DNAs does this DNA segment match with?
  - What is the likely genetic disposition of this patient to this drug?
  - ...
Agenda

- The Problem: DNA Read Mapping
  - State-of-the-art Read Mapper Design

- Algorithmic Acceleration
  - Exploiting Structure of the Genome
  - Exploiting SIMD Instructions

- Hardware Acceleration
  - Specialized Architectures
  - Processing in Memory

- Future Opportunities: New Sequencing Technologies
What Is a Genome Made Of?

The chromosome is made up of genes

The genes consist of DNA

Chromosome - 23 pairs

Cell

Nucleus

Bases
- Adenine
- Cytosine
- Guanine
- Thymine

Nucleotide
- Base
- Sugar
- Phosphate

SAFARI The discovery of DNA’s double-helical structure (Watson+, 1953)
The Central Dogma of Molecular Biology

**DNA**
- Genotypes

**RNA**
- Transcription

**Protein**
- Phenotypes

**Replication**

**Translation**

Phenotypes

Genotypes
DNA Under Electron Microscope

human chromosome #12 from HeLa’s cell
DNA Sequencing

- **Goal:**
  - Find the complete sequence of A, C, G, T’s in DNA.

- **Challenge:**
  - There is no machine that takes long DNA as an input, and gives the complete sequence as output.
  - All sequencing machines chop DNA into pieces and identify relatively small pieces (but not how they fit together).
Untangling Yarn Balls & DNA Sequencing
Genome Sequencers

- Roche/454
- AB SOLiD
- Illumina HiSeq2000
- Pacific Biosciences RS
- SAFARI Ion Torrent PGM
- Ion Torrent Proton
- Illumina MiSeq
- Oxford Nanopore MinION
- Complete Genomics
- Illumina NovaSeq 6000
- Oxford Nanopore GridION

... and more! All produce data with different properties.
The Genomic Era

- 1990-2003: The Human Genome Project (HGP) provides a complete and accurate sequence of all DNA base pairs that make up the human genome and finds 20,000 to 25,000 human genes.

13 year-long $3,000,000,000 (in 1991 USD)
The Genomic Era (continued)

development of high-throughput sequencing (HTS) technologies

High-Throughput Sequencing (HTS)

= Second Generation
= Next Generation
= Massively Parallel Sequencing
= High Throughput Sequencing (HTS)
= Sequencing by Synthesis (Illumina)
High-Throughput Sequencing (HTS)

As a workaround, HTS technologies sequence random short DNA fragments (75-300 basepairs long) of copies of the original molecule.

The sequencer adds the molecule “T” to all bases near the flow cell surface and observes the chemical reaction via a CMOS sensor. If a reaction happens then the base is “A”.

Glass flow cell surface
High-Throughput Sequencing

- Massively parallel sequencing technology
  - Illumina, Roche 454, Ion Torrent, SOLID...

- Small DNA fragments are first amplified and then sequenced in parallel, leading to:
  - High throughput
  - High speed
  - Low cost
  - Short reads

- Sequencing is done by either reading optical signals as each base is added, or by detecting hydrogen ions instead of light, leading to:
  - Low error rates (relatively)
  - Reads lack information about their order and which part of genome they are originated from
1 Sequencing

Genome Analysis

2 Read Mapping

Billions of Short Reads

Sequencing Read Mapping

Variant Calling

Scientific Discovery

reference: TTTATCGCTTCCATGACGCAG
read1: ATCGCATCC
read2: TATCGCATC
read3: CATCCATGA
read4: CGCTTCCAT
read5: CCATGACGC
read6: TTCCATGAC
Example Question: If I give you a bunch of sequences, tell me where they are the same and where they are different.
The Genetic Similarity Between Species

- Human ~ Chimpanzee
  96%

- Human ~ Cat
  90%

- Human ~ Cow
  80%

- Human ~ Banana
  50-60%

Human ~ Human
99.9%
Question 2: Given a bunch of short sequences, Can you identify the approximate species cluster for genomically unknown organisms (bacteria)?

uncleaned de Bruijn graph

http://math.oregonstate.edu/~koslickd
Problem

Need to construct the entire genome from many reads
Billions of Short Reads

Sequencing

Read Mapping

Bottlenecked in Mapping!!

Illumina HiSeq4000

300 M bases/min

on average

2 M bases/min (0.6%)
The Read Mapping Bottleneck

300 Million bases/minute
Read Sequencing **

2 Million bases/minute
Read Mapping *

150x slower

* BWA-MEM
** HiSeqX10, MinION
Read Verification 93%

SAM printing 3%

Candidate alignment locations (CAL) 4%
Read Mapping

- Map many short DNA fragments (reads) to a known reference genome with some differences allowed.

Reference genome

Mapping short reads to reference genome is challenging (billions of 50-300 base pair reads)
Challenges in Read Mapping

- Need to find many mappings of each read
  - A short read may map to many locations, especially with High-Throughput DNA Sequencing technologies
  - How can we find all mappings efficiently?

- Need to tolerate small variances/errors in each read
  - Each individual is different: Subject’s DNA may slightly differ from the reference (Mismatches, insertions, deletions)
  - How can we efficiently map each read with up to $e$ errors present?

- Need to map each read very fast (i.e., performance is important)
  - Human DNA is 3.2 billion base pairs long → Millions to billions of reads (State-of-the-art mappers take weeks to map a human’s DNA)
  - How can we design a much higher performance read mapper?
**Read Alignment/Verification**

- **Edit distance** is defined as the minimum number of edits (i.e. insertions, deletions, or substitutions) needed to make the read exactly match the reference segment.

**NETHERLANDS** x **SWITZERLAND**

N E - T H E R L A N D S
S W I T Z E R L A N D -

- match
- deletion
- insertion
- mismatch
Why Is Read Alignment Slow?

- **Quadratic-time** dynamic-programming algorithm(s)

- **Data dependencies** limit the computation parallelism

- **Entire matrix** computed even though strings may be dissimilar.
Example: Dynamic Programming Table

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>E</th>
<th>T</th>
<th>H</th>
<th>E</th>
<th>R</th>
<th>L</th>
<th>A</th>
<th>N</th>
<th>D</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>W</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

immediate left, upper left, upper entries of its own
Example: Dynamic Programming Table

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>E</th>
<th>T</th>
<th>H</th>
<th>E</th>
<th>R</th>
<th>L</th>
<th>A</th>
<th>N</th>
<th>D</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>S</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>W</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>I</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>T</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>Z</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>E</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>R</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>L</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>A</td>
<td>9</td>
<td>9</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>N</td>
<td>10</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>D</td>
<td>11</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

- Matrix-filling is $O(mn)$ time and space.
- Backtrace is $O(m + n)$ time.
Example: Dynamic Programming

- **Quadratic-time** dynamic-programming algorithm

**WHY?!**

- NETHERLANDS x SWITZERLAND
- NETHERLANDS x S
- NETHERLANDS x SW
- NETHERLANDS x SWI
- NETERLANDS x SWIT
- NETHERLANDS x SWITZ
- NETHERLANDS x SWITZE
- NETHERLANDS x SWITZER
- NETHERLANDS x SWITZERL
- NETHERLANDS x SWITZERLAE
- NETHERLANDS x SWITZERLAN
- NETHERLANDS x SWITZERLAND
Agenda

- The Problem: DNA Read Mapping
  - State-of-the-art Read Mapper Design

- Algorithmic Acceleration
  - Exploiting Structure of the Genome
  - Exploiting SIMD Instructions

- Hardware Acceleration
  - Specialized Architectures
  - Processing in Memory

- Future Opportunities: New Sequencing Technologies
Read Mapping Algorithms: Two Styles

- **Hash based seed-and-extend** (hash table, suffix array, suffix tree)
  - Index the "k-mers" in the genome into a hash table (pre-processing)
  - When searching a read, find the location of a k-mer in the read; then extend through alignment
  - More sensitive (can find all mapping locations), but slow
  - Requires large memory; this can be reduced with cost to run time

- **Burrows-Wheeler Transform & Ferragina-Manzini Index based aligners**
  - BWT is a compression method used to compress the genome index
  - Perfect matches can be found very quickly, memory lookup costs increase for imperfect matches
  - Reduced sensitivity
Hash Table Based Read Mappers

- Key Idea
  - Preprocess the reference into a *Hash Table*
  - Use *Hash Table* to map reads
Hash Table-Based Mappers [Alkan+ Nature Gen’09]

- **k-mer or 12-mer (string of length k)**
  - AAAAAAAAAAAAAA
  - AAAAAAAAC
  - AAAAAAAAAT
  - ...
  - CCCCCCCCCCCCC
  - ...
  - ...
  - ...
  - TTTTTTTTTTTTT

- **Location list—where the k-mer occurs in reference genome**
  - 12  324  577  940
  - 13  421  412  765  889
  - NULL
  - 24  459  744  988  989
  - 36  535  123

- **Reference genome**

- **Once for a reference**
Hash Table Based Read Mappers

- **Key Idea**
  - Preprocess the reference into a *Hash Table*
  - Use *Hash Table* to map reads
Hash Table-Based Mappers [Alkan+ Nature Gen’09]

Hash Table (HT)

Verification/Local Alignment

Reference Genome

Valid mapping

read

k-mers

read

AAAAA

CC

TT

AAAA

CC

TT

12 324 557 940

24 459 744 988 989

36 535 823

***

..
Advantages of Hash Table Based Mappers

- Guaranteed to find all mappings → very sensitive
- Can tolerate up to $e$ errors

http://mrfast.sourceforge.net/

Personalized copy number and segmental duplication maps using next-generation sequencing

Can Alkan$^{1,2}$, Jeffrey M Kidd$^1$, Tomas Marques-Bonet$^{1,3}$, Gozde Aksay$^1$, Francesca Antonacci$^1$, Fereydoun Hormozdiari$^4$, Jacob O Kitzman$^1$, Carl Baker$^1$, Maika Malig$^1$, Onur Mutlu$^5$, S Cenk Sahinalp$^4$, Richard A Gibbs$^6$ & Evan E Eichler$^{1,2}$

Problem and Goal

- Poor performance of existing read mappers: Very slow
  - Verification/alignment takes too long to execute
  - Verification requires a memory access for reference genome + many base-pair-wise comparisons between the reference and the read (edit distance computation)

- Goal: Speed up the mapper by reducing the cost of verification
Overarching Key Idea

Filter fast before you align

Minimize costly edit distance computations
Agenda

- The Problem: DNA Read Mapping
  - State-of-the-art Read Mapper Design

- Algorithmic Acceleration
  - Exploiting Structure of the Genome
  - Exploiting SIMD Instructions

- Hardware Acceleration
  - Specialized Architectures
  - Processing in Memory

- Future Opportunities: New Sequencing Technologies
Reducing the Cost of Verification

- We observe that most verification (edit distance computation) calculations are unnecessary
  - 1 out of 1000 potential locations passes the verification process

- We observe that we can get rid of unnecessary verification calculations by
  - Detecting and rejecting early invalid mappings (filtering)
  - Reducing the number of potential mappings to examine
Key Observations [Xin+, BMC Genomics 2013]

- Observation 1
  - Adjacent k-mers in the read should also be adjacent in the reference genome
  - Read mapper can quickly reject mappings that do not satisfy this property

- Observation 2
  - Some k-mers are cheaper to verify than others because they have shorter location lists (they occur less frequently in the reference genome)
    - Mapper needs to examine only $e+1$ k-mers’ locations to tolerate $e$ errors
  - Read mapper can choose the cheapest $e+1$ k-mers and verify their locations
FastHASH Mechanisms [Xin+, BMC Genomics 2013]

- **Adjacency Filtering (AF):** Rejects obviously invalid mapping locations at early stage to avoid unnecessary verifications

- **Cheap K-mer Selection (CKS):** Reduces the absolute number of potential mapping locations to verify
Adjacency Filtering (AF)

- **Goal:** detect and filter out invalid mappings at early stage
- **Key Insight:** For a valid mapping, adjacent k-mers in the read are also adjacent in the reference genome

- **Key Idea:** search for adjacent locations in the k-mers’ location lists
  - If more than \( e \) k-mers fail $\rightarrow$ there must be more than \( e \) errors $\rightarrow$ invalid mapping
Adjacency Filtering (AF)

References

Hash Table (HT)

- Adjacency Filtering (AF)

k-mers

Hash Table (HT)

- Adjacency Filtering (AF)

Reference Genome

- Adjacency Filtering (AF)
Adjacency Filtering (AF): Rejects obviously invalid mapping locations at early stage to avoid unnecessary verifications

Cheap K-mer Selection (CKS): Reduces the absolute number of potential mapping locations to verify
Cheap K-mer Selection (CKS)

- **Goal:** Reduce the number of potential mappings to examine

- **Key insight:**
  - K-mers have different cost to examine: Some k-mers are cheaper as they have fewer locations than others (occur less frequently in reference genome)

- **Key idea:**
  - Sort the k-mers based on their number of locations
  - Select the k-mers with the fewest number locations to verify
Cheap K-mer Selection

- $e=2$ (examine 3 k-mers)

Previous work needs to verify:

- 3004 locations

FastHASH verifies only:

- 8 locations
Methodology

- Implemented **FastHASH** on top of state-of-the-art mapper: **mrFAST**
  - New version **mrFAST-2.5.0.0** over mrFAST-2.1.0.6

- Tested with real read sets generated from Illumina platform
  - 1M reads of a human (160 base pairs)
  - 500K reads of a chimpanzee (101 base pairs)
  - 500K reads of a orangutan (70 base pairs)

- Tested with simulated reads generated from reference genome
  - 1M simulated reads of human (180 base pairs)

- Evaluation system
  - Intel Core i7 Sandy Bridge machine
  - 16 GB of main memory
FastHASH Speedup: Entire Read Mapper

With FastHASH, new mrFAST obtains up to 19x speedup over previous version, without losing valid mappings.
Analysis

- Reduction of potential mappings with FastHASH

FastHASH filters out over 99% of the potential mappings without sacrificing any valid mappings
FastHASH Conclusion

- Problem: Existing read mappers perform poorly in mapping millions of short reads to the reference genome, in the presence of errors.

- Observation: Most of the verification calculations are unnecessary → filter them out.

- Key Idea: Exploit the structure of the genome to
  - Reject invalid mappings early (Adjacency Filtering)
  - Reduce the number of possible mappings to examine (Cheap K-mer Selection)

- Key Result: FastHASH obtains up to 19x speedup over the state-of-the-art mapper without losing valid mappings.
More on FastHASH

- Download source code and try for yourself
  - Download link to FastHASH

Xin et al. BMC Genomics 2013, 14(Suppl 1):S13
http://www.biomedcentral.com/1471-2164/14/S1/S13

Accelerating read mapping with FastHASH

Hongyi Xin¹, Donghyuk Lee¹, Farhad Hormozdiari², Samihan Yedkar¹, Onur Mutlu¹*, Can Alkan³*

From The Eleventh Asia Pacific Bioinformatics Conference (APBC 2013)
Vancouver, Canada. 21-24 January 2013

Agenda

- The Problem: DNA Read Mapping
  - State-of-the-art Read Mapper Design

- Algorithmic Acceleration
  - Exploiting Structure of the Genome
  - Exploiting SIMD Instructions

- Hardware Acceleration
  - Specialized Architectures
  - Processing in Memory

- Future Opportunities: New Sequencing Technologies
An Example: Shifted Hamming Distance

https://github.com/CMU-SAFARI/Shifted-Hamming-Distance

Shifted Hamming distance: a fast and accurate SIMD-friendly filter to accelerate alignment verification in read mapping

Hongyi Xin¹,* , John Greth², John Emmons², Gennady Pekhimenko¹, Carl Kingsford³, Can Alkan⁴,* and Onur Mutlu²,*

Shifted Hamming Distance

- **Key observation:**
  - If two strings differ by $E$ edits, then every bp match can be aligned in at most $2E$ shifts (of one of the strings).
  - Insight: Shifting a string by one “corrects” for one “error”

- **Key idea:**
  - Compute “Shifted Hamming Distance”: AND of $2E$ Hamming Distances of two strings, to filter out invalid mappings
  - Uses bit-parallel operations that nicely map to SIMD instructions

- **Key result:**
  - SHD is 3x faster than SeqAn (the best implementation of Gene Myers’ bit-vector algorithm), with only a 7% false positive rate
  - The fastest CPU-based filtering (pre-alignment) mechanism
Hamming Distance ($\sum \Theta$)

3 matches    5 mismatches

**Edit = 1 Deletion**

To cancel the effect of a deletion, we need to shift in the *right* direction.
Insight: Shifting a String Helps Similarity Search

3 matches      5 mismatches

To cancel the effect of the deletion, we need to shift in the right direction
Insight: Shifting a String Helps Similarity Search

7 matches 1 mismatches
Shifted Hamming Distance

7 matches

1 mismatches

*Edit = 1 Deletion*

**ISTANBUL**

**00011111**

**11100000**
We need to compute $2E+1$ vectors, $E=edit\ distance\ threshold$

$$dp[i][j] = \begin{cases} 0 & \text{if } X[i]=Y[j] \\ 1 & \text{if } X[i] \neq Y[j] \end{cases}$$

No data dependencies!
Key Idea of SHD Filtering

Generate 2E+1 masks

Amend random zeros:
101 → 111 & 1001 → 1111

AND all masks, ACCEPT iff number of ‘1’ ≤ Threshold

--- Masks after amendment ---

Hamming Mask
1-Deletion Mask
2-Deletion Mask
3-Deletion Mask
1-Insertion Mask
2-Insertion Mask
3-Insertion Mask

AND Mask

Needleman-Wunsch Alignment
Alignment vs. Pre-alignment (Filtering)

### Needleman-Wunsch

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>T</th>
<th>A</th>
<th>T</th>
<th>A</th>
<th>T</th>
<th>A</th>
<th>C</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### SHD

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>T</th>
<th>A</th>
<th>T</th>
<th>A</th>
<th>T</th>
<th>A</th>
<th>C</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Independent vectors can be processed in parallel using hardware technologies.
Sequencing generates many reads, each of which potentially mapping to many locations

Filtering (Pre-alignment) eliminates the need to verify/align read to invalid mapping locations

Alignment/verification (costly edit distance computation) is performed only on reads that pass the filter

- New bottleneck in read mapping becomes the “filtering (pre-alignment)” step
Agenda

- The Problem: DNA Read Mapping
  - State-of-the-art Read Mapper Design

- Algorithmic Acceleration
  - Exploiting Structure of the Genome
  - Exploiting SIMD Instructions

- Hardware Acceleration
  - Specialized Architectures
  - Processing in Memory

- Future Opportunities: New Sequencing Technologies
Location Filtering

- **Alignment** is expensive
  - 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
Ideal Filtering Algorithm

- Minimal False Accept Rate
- Maximal True Reject Rate
- Zero False Reject Rate
- Faster Than Mapper

Filter out all incorrect mappings

Do not filter out any correct mappings
Alignment vs. Pre-alignment (Filtering)

Needleman-Wunsch

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>T</th>
<th>A</th>
<th>T</th>
<th>A</th>
<th>A</th>
<th>T</th>
<th>A</th>
<th>C</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SHD

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>T</th>
<th>A</th>
<th>T</th>
<th>A</th>
<th>A</th>
<th>T</th>
<th>A</th>
<th>C</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Independent vectors can be processed in parallel using hardware technologies

DRAM Layers

Logic Layer
Our Solution: GateKeeper

1. High throughput DNA sequencing (HTS) technologies
2. Read Pre-Alignment Filtering Fast & Low False Positive Rate
3. Read Alignment Slow & Zero False Positives

FPGA-based Alignment Filter.

$10^{12}$ mappings

$10^3$ mappings
GateKeeper Walkthrough

Generate 2E+1 masks

Amend random zeros:
101 → 111 & 1001 → 1111

AND all masks, ACCEPT iff number of ‘1’ ≤ Threshold

--- Masks after amendment ---

Hamming Mask :
1-Deletion Mask :
2-Deletion Mask :
3-Deletion Mask :
1-Insertion Mask :
2-Insertion Mask :
3-Insertion Mask :
AND Mask :

Needleman-Wunsch Alignment :

```plaintext
AAAAAAAAAAAAAAGAGAGAGAGATATTTAGTGTTGCAGCACACAAACACAAAGAGGACCAACTTACGTGCTAAAAGGGGGAACATTGTTGGGCCGGA
AAAAAAAAAAAAAAGAGAGAGAGATAGTTAGTGTTGCAGCCACTACAACACAAAAGAGGACCAACTTACGTGCTAAAAGGGGAGACATTGTTGGGCCGG
00000000000000000000000000100000000000011111110111100011111011111111111001000001111011010010101
00000100111110011111111111001001010010101010111111111111011011010111111111101001011010110111110
000000010111110011111111111111111110011100110111111111111111111111111111111111111111111111111111110
0000000000000000000000000000100000111111111111111111111111111111111111111111111111111111111111111110
```
GateKeeper Walkthrough (cont’d)

Generate 2E+1 masks

Amend random zeros:
101 → 111 & 1001 → 1111

AND all masks, ACCEPT iff number of ‘1’ ≤ Threshold

- E right-shift registers (length=ReadLength)
- E left-shift registers (length=ReadLength)
- (2E+1) * (ReadLength) 2-XOR operations.

- (2E)*(ReadLength) 2-AND operations.
- (ReadLength/4) 5-input LUT.
- \( \log_2 \text{ReadLength} \)-bit counter.

Hamming mask before amending:

0 1 0 0 1 0 0 0 1 1 0 1 0 0 0 1 0 1 0 1 1 0 0 0 1 1 1 1 0 0 0 1 0

Hamming mask after amending:

0 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 0

5-input LUT

(2E+1)*(ReadLength) 5-input LUT.
**GateKeeper Accelerator Architecture**

- **Maximum data throughput** = ~13.3 billion bases/sec
- Can examine 8 (300 bp) or 16 (100 bp) mappings concurrently at 250 MHz
- **Occupies 50%** (100 bp) to **91%** (300 bp) of the FPGA slice LUTs and registers

---

[Diagram showing the flow of data and processing through GateKeeper.]
## GateKeeper vs. SHD

<table>
<thead>
<tr>
<th>GateKeeper</th>
<th>SHD</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPGA (Xilinx VC709)</td>
<td>Intel SIMD</td>
</tr>
<tr>
<td>Multi-core (parallel)</td>
<td>Single-core (sequential)</td>
</tr>
<tr>
<td>Examines a single mapping @ 125 MHz</td>
<td>Examines a single mapping @ ~2MHz</td>
</tr>
<tr>
<td>Limited to PCIe Gen3(4x) transfer rate (128 bits @ 250MHz)</td>
<td>Limited to a read length of 128 bp (SSE register size)</td>
</tr>
<tr>
<td>Amending requires:</td>
<td></td>
</tr>
<tr>
<td>(2E+1) 5-input LUT.</td>
<td>Amending requires:</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4(2E+1) bitwise OR.</td>
</tr>
<tr>
<td></td>
<td>4(2E+1) packed shuffle.</td>
</tr>
<tr>
<td></td>
<td>3(2E+1) shift.</td>
</tr>
</tbody>
</table>
GateKeeper: Speed & Accuracy Results

90x-130x faster filter
than SHD (Xin et al., 2015) and the Adjacency Filter (Xin et al., 2013)

4x lower false accept rate
than the Adjacency Filter (Xin et al., 2013)

10x speedup in read mapping
with the addition of GateKeeper to the mrFAST mapper (Alkan et al., 2009)

Freely available online
github.com/BilkentCompGen/GateKeeper
Conclusions

- FPGA-based pre-alignment greatly speeds up read mapping
  - 10x speedup of a state-of-the-art mapper (mrFAST)

- FPGA-based pre-alignment can be integrated with the sequencer
  - It can help to hide the complexity and details of the FPGA
  - Enables real-time filtering while sequencing
More on GateKeeper

- Download and test for yourself
  https://github.com/BilkentCompGen/GateKeeper

Alser+, "GateKeeper: A New Hardware Architecture for Accelerating Pre-Alignment in DNA Short Read Mapping", Bioinformatics, 2017.

---

Sequence analysis

GateKeeper: A New Hardware Architecture for Accelerating Pre-Alignment in DNA Short Read Mapping

Mohammed Alser¹,*, Hasan Hassan², Hongyi Xin³, Oğuz Ergin², Onur Mutlu⁴,*, and Can Alkan¹,*
**MAGNET** (AACBB 2018, TIR 2017)

- **Key observation:** the use of **AND operation** to check if a zero (match) exists in a column introduces filtering inaccuracy.
- **Key Idea:** count the **consecutive zeros** in each mask and select the longest in a divide-and-conquer approach.
- **MAGNET** is **17x to 105x more accurate** than GateKeeper and SHD.

---

### Hamming Mask

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
</table>

### Deletion Mask

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
</table>

### Insertion Mask

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
</table>

### Needleman-Wunsch Alignment

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
</table>

---

Refer to the original document for detailed explanations and more examples.
MAGNET Walkthrough

Find the longest segment of consecutive zeros
Exclude the errors from the search space
Divide the problem into two subproblems and repeat

"MAGNET: understanding and improving the accuracy of genome pre-alignment filtering", arXiv preprint 2017
MAGNET Accelerator

Short Reads Repository

FPGA Board

Host

RIFFA Driver

RIFFA RX Engine

RIFFA TX Engine

FIFO

Read Controller

FIFO

Mapping Controller

MAGNET #1

BSD

LME

RLEE

MAGNET #2

MAGNET #3

MAGNET #4

MAGNET #5

MAGNET #6

MAGNET #7

MAGNET #8

MAGNET #9

MAGNET #N

MAGNET #N-1

MAGNET #N-2

3 filtering stages
Agenda

- The Problem: DNA Read Mapping
  - State-of-the-art Read Mapper Design

- Algorithmic Acceleration
  - Exploiting Structure of the Genome
  - Exploiting SIMD Instructions

- Hardware Acceleration
  - Specialized Architectures
  - Processing in Memory

- Future Opportunities: New Sequencing Technologies
Read Mapping & Filtering

- **Problem:** Heavily bottlenecked by Data Movement

- GateKeeper performance limited by DRAM bandwidth [Alser+, Bioinformatics 2017]

- Ditto for SHD [Xin+, Bioinformatics 2015]

- **Solution:** Processing-in-memory can alleviate the bottleneck

- However, we need to design mapping & filtering algorithms to fit processing-in-memory
Hash Tables in Read Mapping

Read Sequence (100 bp)

Hash Table

Reference Genome

Filter

Match!

37 140
894 1203
1564

Mismatch

False Negative
We need to design mapping & filtering algorithms that fit processing-in-memory.
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).

![Diagram showing bin partitioning and bitvector representation.]

- 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.
GRIM-Filter: Bitvectors

Bin x

<table>
<thead>
<tr>
<th>Bin x Bitvector</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAAAA</td>
</tr>
<tr>
<td>CGTGA</td>
</tr>
<tr>
<td>TGAGT</td>
</tr>
<tr>
<td>GAGTC</td>
</tr>
<tr>
<td>GTGAG</td>
</tr>
</tbody>
</table>

...
GRIM-Filter: Bitvectors

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

For bin size $\sim 200$, and $n = 5$, memory footprint $\sim 3.8$ GB
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 $r$

GAACTTGGAGTCTA ... CGAG

1. **Get tokens**

2. **Read bitvector for bin_num(x)**

3. **Match tokens to bitvector**

4. **Sum**

5. **Compare**

- $\geq$ Threshold?
  - NO: Discard
  - YES: Send to Read Mapper for Sequence Alignment

SAFARI
Our Proposal: GRIM-Filter

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

**INPUT**: Read Sequence

```
GAAC TTGC GAG ... GTATT
```

**GRIM-Filter**: Filter Bitmask Generator

```
...0001 010...011010...
```

**Seed Location Filter Bitmask**

**INPUT**: All Potential Seed Locations

```
020128 020131 ... 414415 ...
```

**GRIM-Filter**: Seed Location Checker

```
...0001 010...011010...
```

**KEEP**

... **KEEP**

**DISCARD**

**Reference Segment Storage**

```
reference segment @ 020131
... 
reference segment @ 414415
```

**Read Mapper**: Sequence Alignment

**EDIT-DISTANCE CALCULATION**

**OUTPUT**: Correct Mappings
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
Opportunity: 3D-Stacked Logic + Memory

Other “True 3D” technologies under development
## DRAM Landscape (circa 2015)

<table>
<thead>
<tr>
<th>Segment</th>
<th>DRAM Standards &amp; Architectures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commodity</td>
<td>DDR3 (2007) [14]; DDR4 (2012) [18]</td>
</tr>
<tr>
<td>Performance</td>
<td>eDRAM [28], [32]; RLDAM3 (2011) [29]</td>
</tr>
</tbody>
</table>

Table 1. Landscape of DRAM-based 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, via high-bandwidth low-latency access to DRAM layers
  - 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 incrementer LUTs, 7-bit counters, and comparators in logic layer

Details are in [Kim+, BMC Genomics 2018]
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 \cite{Xin} 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
More on GRIM-Filter


Proceedings of the 16th Asia Pacific Bioinformatics Conference (APBC), Yokohama, Japan, January 2018.
arxiv.org Version (pdf)

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

Jeremie S. Kim¹,⁶*, Damla Senol Cali¹, Hongyi Xin², Donghyuk Lee³, Saugata Ghose¹, Mohammed Alser⁴, Hasan Hassan⁶, Oguz Ergin⁵, Can Alkan⁴* and Onur Mutlu⁶,¹*

*From* The Sixteenth Asia Pacific Bioinformatics Conference 2018
Yokohama, Japan. 15-17 January 2018
Aside: In-Memory Graph Processing

- Large graphs are everywhere (circa 2015)
  - 36 Million Wikipedia Pages
  - 1.4 Billion Facebook Users
  - 300 Million Twitter Users
  - 30 Billion Instagram Photos

- Scalable large-scale graph processing is challenging

![Bar chart showing speedup with 32 Cores and 128 Cores, with a +42% increase in Speedup]
Key Bottlenecks in Graph Processing

```java
for (v: graph.vertices) {
    for (w: v.successors) {
        w.next_rank += weight * v.rank;
    }
}
```

1. Frequent random memory accesses
2. Little amount of computation
Tesseract: System for Graph Processing

Interconnected set of 3D-stacked memory+logic chips with simple cores

- Host Processor
- Memory-Mapped Accelerator Interface (Noncacheable, Physically Addressed)
- Crossbar Network
- In-Order Core
- DRAM Controller
- LP
- PF Buffer
- MTP
- Message Queue
- NI

SAFARI: Ahn+, "A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing" ISCA 2015.
Tesseract System for Graph Processing

Host Processor
Memory-Mapped Accelerator Interface
(Noncacheable, Physically Addressed)

Crossbar Network

Memory

Logic

In-Order Core

Communications via Remote Function Calls

Message Queue
for (v: graph.vertices) {
    for (w: v.successors) {
        w.next_rank += weight * v.rank;
    }
}
for (v: graph.vertices) {
    for (w: v.successors) {
        w.next_rank += weight * v.rank;
    }
}

Vault #1

Vault #2

v

&w

w
Communications In Tesseract (III)

```java
for (v: graph.vertices) {
    for (w: v.successors) {
        put(w.id, function() { w.next_rank += weight * v.rank; });
    }
}
barrier();
```

Non-blocking Remote Function Call

Can be delayed until the nearest barrier
Remote Function Call (Non-Blocking)

1. Send function address & args to the remote core
2. Store the incoming message to the message queue
3. Flush the message queue when it is full or a synchronization barrier is reached

```javascript
put(w.id, function() { w.next_rank += value; })
```
Tesseract System for Graph Processing

Host Processor
Memory-Mapped Accelerator Interface
Noncacheable, Physically Addressed)

Memory

Logic

Crossbar Network

Prefetching

LP
PF Buffer
MTP
Message Queue

DRAM Controller
NI

SAFARI
Evaluated Systems

**DDR3-OoO**

- 8 OoO 4GHz
- 8 OoO 4GHz
- 8 OoO 4GHz

102.4GB/s

**HMC-OoO**

- 8 OoO 4GHz
- 8 OoO 4GHz
- 8 OoO 4GHz

640GB/s

**HMC-MC**

- 128 In-Order 2GHz
- 128 In-Order 2GHz
- 128 In-Order 2GHz

640GB/s

**Tesseract**

- 32 Tesseract Cores
- 8TB/s

---

SAFARI Ahn+, “A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing” ISCA 2015.
Tesseract Graph Processing Performance

>13X Performance Improvement

On five graph processing algorithms

- DDR3-OoO: +56%
- HMC-OoO: +25%
- HMC-MC: 9.0x
- Tesseract LP: 11.6x
- Tesseract LP-MTP: 13.8x

SAFARI Ahn+, “A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing” ISCA 2015.
Tesseract Graph Processing Performance

Memory Bandwidth Consumption

- DDR3-OoO: 80GB/s
- HMC-OoO: 190GB/s
- HMC-MC: 243GB/s
- Tesseract: 1.3TB/s
- Tesseract-LP: 2.2TB/s
- Tesseract-LP-MTP: 2.9TB/s
Effect of Bandwidth & Programming Model

- **HMC-MC Bandwidth (640GB/s)**
- **Tesseract Bandwidth (8TB/s)**

<table>
<thead>
<tr>
<th>Programming Model</th>
<th>Bandwidth</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMC-MC</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>HMC-MC + PIM BW</td>
<td></td>
<td>2.3x</td>
</tr>
<tr>
<td>Tesseract + Conventional BW</td>
<td></td>
<td>3.0x</td>
</tr>
<tr>
<td>Tesseract (No Prefetching)</td>
<td></td>
<td>6.5x</td>
</tr>
</tbody>
</table>
Tesseract Graph Processing System Energy

Ahn+, “A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing” ISCA 2015.
More on Tesseract

- Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi,

"A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing"


[Slides (pdf)] [Lightning Session Slides (pdf)]

A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing

Junwhan Ahn  Sungpack Hong§  Sungjoo Yoo  Onur Mutlu†  Kiyoung Choi
junwhan@snu.ac.kr, sungpack.hong@oracle.com, sungjoo.yoo@gmail.com, onur@cmu.edu, kchoi@snu.ac.kr
Seoul National University  §Oracle Labs  †Carnegie Mellon University
Agenda

- **The Problem: DNA Read Mapping**
  - State-of-the-art Read Mapper Design

- **Algorithmic Acceleration**
  - Exploiting Structure of the Genome
  - Exploiting SIMD Instructions

- **Hardware Acceleration**
  - Specialized Architectures
  - Processing in Memory

- **Future Opportunities: New Sequencing Technologies**
Recall: High-Throughput Sequencing

- Massively parallel sequencing technology
  - Illumina, Roche 454, Ion Torrent, SOLID...

- Small DNA fragments are first amplified and then sequenced in parallel, leading to
  - High throughput
  - High speed
  - Low cost
  - Short reads
    - Amplification step limits the read length since too short or too long fragments are not amplified well.

- Sequencing is done by either reading optical signals as each base is added, or by detecting hydrogen ions instead of light, leading to:
  - Low error rates (relatively)
  - Reads lack information about their order and which part of genome they are originated from
Nanopore Sequencing Technology

- **Nanopore sequencing** is an emerging and a promising single-molecule DNA sequencing technology.

- First nanopore sequencing device, **MinION**, made commercially available by **Oxford Nanopore Technologies (ONT)** in **May 2014**.
  - Inexpensive
  - Long read length (> 882K bp)
  - Portable: Pocket-sized
  - Produces data in real-time
Nanopore Sequencing Technology

Nanopore sequencing is an emerging and a promising single-molecule DNA sequencing technology.

- **No amplification** → **Less limit on read length** → **Longer read length**

First nanopore sequencing device, **MinION**, made commercially available by **Oxford Nanopore Technologies (ONT)** in **May 2014**.

- Inexpensive
- Long read length (> 882K bp)
- Portable: Pocket-sized
- Produces data in real-time
Nanopore Sequencing

- Nanopore is a nano-scale hole
- In nanopore sequencers, an ionic current passes through the nanopores
- When the DNA strand passes through the nanopore, the sequencer measures the change in current
- This change is used to identify the bases in the strand with the help of different electrochemical structures of the different bases
Advantages of Nanopore Sequencing

Nanopores:

- Do *not* require any labeling of the DNA or nucleotide for detection during sequencing
- Rely on the electronic or chemical structure of the different nucleotides for identification
- Allow sequencing very long reads, and
- Provide portability, low cost, and high throughput.
Challenges of Nanopore Sequencing

- One major drawback: high error rates

- Nanopore sequence analysis tools have a critical role to:
  - overcome high error rates
  - take better advantage of the technology

- Faster tools are critically needed to:
  - Take better advantage of the real-time data production capability of MinION
  - Enable fast, real-time data analysis
Nanopore Genome Assembly Pipeline

**Basecalling**
- Tools: Metrichor, Nanonet, Scrappie, Nanocall, DeepNano

**Read-to-Read Overlap Finding**
- Tools: GraphMap, Minimap

**Assembly**
- Tools: Canu, Miniasm

**Read Mapping**
- Tools: BWA-MEM, Minimap, (GraphMap)

**Polishing**
- Tools: Nanopolish, Racon

**Figure 1.** The analyzed genome assembly pipeline using nanopore sequence data, with its five steps and the associated tools for each step.

### Nanopore Genome Assembly Tools (I)

Table 12. Accuracy analysis results for the full pipeline with a focus on the last two steps.

<table>
<thead>
<tr>
<th></th>
<th>Number of Bases</th>
<th>Number of Contigs</th>
<th>Identity (%)</th>
<th>Coverage (%)</th>
<th>Number of Mismatches</th>
<th>Number of Indels</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4,683,072</td>
<td>1</td>
<td>99.48</td>
<td>99.93</td>
<td>8,198</td>
<td>15,581</td>
</tr>
<tr>
<td>2</td>
<td>4,540,352</td>
<td>1</td>
<td>92.33</td>
<td>96.31</td>
<td>162,884</td>
<td>182,965</td>
</tr>
<tr>
<td>3</td>
<td>4,637,916</td>
<td>2</td>
<td>92.38</td>
<td>95.80</td>
<td>159,206</td>
<td>180,603</td>
</tr>
<tr>
<td>4</td>
<td>4,650,502</td>
<td>1</td>
<td>98.46</td>
<td>100.00</td>
<td>18,036</td>
<td>51,842</td>
</tr>
<tr>
<td>5</td>
<td>4,648,710</td>
<td>1</td>
<td>98.45</td>
<td>100.00</td>
<td>17,906</td>
<td>52,168</td>
</tr>
<tr>
<td>6</td>
<td>4,598,267</td>
<td>1</td>
<td>97.70</td>
<td>99.91</td>
<td>24,014</td>
<td>82,906</td>
</tr>
<tr>
<td>7</td>
<td>4,600,109</td>
<td>1</td>
<td>97.78</td>
<td>100.00</td>
<td>23,339</td>
<td>79,721</td>
</tr>
<tr>
<td>8</td>
<td>4,622,285</td>
<td>1</td>
<td>98.48</td>
<td>100.00</td>
<td>16,872</td>
<td>52,509</td>
</tr>
<tr>
<td>9</td>
<td>4,620,597</td>
<td>1</td>
<td>98.49</td>
<td>100.00</td>
<td>16,874</td>
<td>52,232</td>
</tr>
<tr>
<td>10</td>
<td>4,593,402</td>
<td>1</td>
<td>98.01</td>
<td>99.97</td>
<td>20,322</td>
<td>72,284</td>
</tr>
<tr>
<td>11</td>
<td>4,592,907</td>
<td>1</td>
<td>98.04</td>
<td>100.00</td>
<td>20,170</td>
<td>70,705</td>
</tr>
<tr>
<td>12</td>
<td>4,673,871</td>
<td>1</td>
<td>98.40</td>
<td>99.98</td>
<td>13,583</td>
<td>60,612</td>
</tr>
<tr>
<td>13</td>
<td>4,673,606</td>
<td>1</td>
<td>98.40</td>
<td>99.98</td>
<td>13,798</td>
<td>60,423</td>
</tr>
<tr>
<td>14</td>
<td>5,157,041</td>
<td>8</td>
<td>97.87</td>
<td>99.80</td>
<td>18,085</td>
<td>78,492</td>
</tr>
<tr>
<td>15</td>
<td>5,156,375</td>
<td>8</td>
<td>97.87</td>
<td>99.94</td>
<td>17,922</td>
<td>77,807</td>
</tr>
<tr>
<td>16</td>
<td>1,383,851</td>
<td>86</td>
<td>93.49</td>
<td>28.82</td>
<td>19,057</td>
<td>65,244</td>
</tr>
<tr>
<td>17</td>
<td>1,367,834</td>
<td>86</td>
<td>94.43</td>
<td>28.74</td>
<td>15,610</td>
<td>55,275</td>
</tr>
<tr>
<td>18</td>
<td>4,707,961</td>
<td>5</td>
<td>90.75</td>
<td>97.11</td>
<td>91,502</td>
<td>347,005</td>
</tr>
<tr>
<td>19</td>
<td>4,673,069</td>
<td>5</td>
<td>92.23</td>
<td>97.10</td>
<td>72,646</td>
<td>291,918</td>
</tr>
<tr>
<td>20</td>
<td>7,429,290</td>
<td>106</td>
<td>96.46</td>
<td>99.24</td>
<td>27,811</td>
<td>102,682</td>
</tr>
<tr>
<td>21</td>
<td>7,404,454</td>
<td>106</td>
<td>96.03</td>
<td>99.21</td>
<td>34,023</td>
<td>110,640</td>
</tr>
<tr>
<td>22</td>
<td>4,566,253</td>
<td>1</td>
<td>96.76</td>
<td>99.86</td>
<td>25,791</td>
<td>125,386</td>
</tr>
<tr>
<td>23</td>
<td>4,571,810</td>
<td>1</td>
<td>96.90</td>
<td>99.97</td>
<td>24,994</td>
<td>119,519</td>
</tr>
</tbody>
</table>

## Nanopore Genome Assembly Tools (II)

### Table 13. Performance analysis results for the full pipeline with a focus on the last two steps.

<table>
<thead>
<tr>
<th>Step 4: Read Mapper</th>
<th>Step 5: Polisher</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wall Clock Time (h:m:s)</td>
<td>CPU Time (h:m:s)</td>
</tr>
<tr>
<td>2</td>
<td>Metrichor + Minimap + Miniasm + BWA-MEM + Nanopolish</td>
</tr>
<tr>
<td>3</td>
<td>Metrichor + GraphMap + Miniasm + BWA-MEM + Nanopolish</td>
</tr>
<tr>
<td>5</td>
<td>Metrichor + — + Canu + Minimap + Racon</td>
</tr>
<tr>
<td>7</td>
<td>Metrichor + Minimap + Miniasm + BWA-MEM + Racon</td>
</tr>
<tr>
<td>8</td>
<td>Nanonet + — + Canu + BWA-MEM + Racon</td>
</tr>
<tr>
<td>9</td>
<td>Nanonet + — + Canu + Minimap + Racon</td>
</tr>
<tr>
<td>10</td>
<td>Nanonet + Minimap + Miniasm + BWA-MEM + Racon</td>
</tr>
<tr>
<td>11</td>
<td>Nanonet + Minimap + Miniasm + BWA-MEM + Racon</td>
</tr>
<tr>
<td>13</td>
<td>Scrappie + — + Canu + Minimap + Racon</td>
</tr>
<tr>
<td>15</td>
<td>Scrappie + Minimap + Miniasm + Minimap + Racon</td>
</tr>
<tr>
<td>16</td>
<td>Nanocall + — + Canu + BWA-MEM + Racon</td>
</tr>
<tr>
<td>17</td>
<td>Nanocall + — + Canu + Minimap + Racon</td>
</tr>
<tr>
<td>18</td>
<td>Nanocall + Minimap + Miniasm + BWA-MEM + Racon</td>
</tr>
<tr>
<td>19</td>
<td>Nanocall + Minimap + Miniasm + Minimap + Racon</td>
</tr>
<tr>
<td>21</td>
<td>DeepNano + — + Canu + Minimap + Racon</td>
</tr>
<tr>
<td>23</td>
<td>DeepNano + Minimap + Miniasm + Minimap + Racon</td>
</tr>
</tbody>
</table>
Nanopore sequencing technology and tools for genome assembly: computational analysis of the current state, bottlenecks and future directions

Damla Senol Cali+, Jeremie S Kim, Saugata Ghose, Can Alkan, Onur Mutlu

Briefings in Bioinformatics, bby017, https://doi.org/10.1093/bib/bby017
Published: 02 April 2018  Article history ▼

[Preliminary arxiv.org version]
Agenda

- The Problem: DNA Read Mapping
  - State-of-the-art Read Mapper Design

- Algorithmic Acceleration
  - Exploiting Structure of the Genome
  - Exploiting SIMD Instructions

- Hardware Acceleration
  - Specialized Architectures
  - Processing in Memory

- Future Opportunities: New Sequencing Technologies
Conclusion

- **System design for bioinformatics** is a critical problem
  - It has large scientific, medical, societal, personal implications

- This talk is about accelerating **a key step in bioinformatics**: genome sequence analysis
  - In particular, **read mapping**

- We covered various **recent ideas to accelerate read mapping**
  - My personal journey since September 2006

- **Many future opportunities exist**
  - Especially with new sequencing technologies
  - Especially with new applications and use cases
Acknowledgments

- Can Alkan, Bilkent University

- Many students at ETH, CMU, Bilkent
  - Mohammed Alser, Damla Senol Cali, Jeremie Kim, Hasan Hassan, Donghyuk Lee, Hongyi Xin, ...

- Funders:
  - NIH and Industrial Partners (Alibaba, AMD, Google, Facebook, HP Labs, Huawei, IBM, Intel, Microsoft, Nvidia, Oracle, Qualcomm, Rambus, Samsung, Seagate, VMware)

- All papers, source code, and more are at:
  - https://people.inf.ethz.ch/omutlu/projects.htm
Accelerating Genome Analysis
A Primer on an Ongoing Journey

Onur Mutlu
omutlu@gmail.com
https://people.inf.ethz.ch/omutlu
16 February 2019
AACBB Keynote Talk