P&S Mobile Genomics

Lecture 12: GenASM

Dr. Damla Senol Cali
ETH Zurich
Fall 2022
16 January 2023
Genome Sequencing

- **Genome sequencing:** Enables us to determine the order of the DNA sequence in an organism’s genome
  - Plays a pivotal role in:
    - Personalized medicine
    - Outbreak tracing
    - Understanding of evolution

- Modern genome sequencing machines extract smaller randomized fragments of the original DNA sequence, known as **reads**
  - **Short reads:** a few hundred base pairs, error rate of ~0.1%
  - **Long reads:** thousands to millions of base pairs, error rate of 10–15%
Genome Sequence Analysis

- **Read mapping**: *First key step* in genome sequence analysis (GSA)
  - Aligns reads to one or more possible locations within the reference genome, and
  - Finds the matches and differences between the read and the reference genome segment at that location

- Multiple steps of read mapping require *approximate string matching*
  - Approximate string matching (ASM) enables read mapping to account for sequencing errors and genetic variations in the reads

- Bottlenecked by the computational power and memory bandwidth limitations of existing systems
GenASM: ASM Framework for GSA

Our Goal:

Accelerate approximate string matching by designing a fast and flexible framework, which can accelerate multiple steps of genome sequence analysis

GenASM: First ASM acceleration framework for GSA

- Based upon the Bitap algorithm
  - Uses fast and simple bitwise operations to perform ASM

- Modified and extended ASM algorithm
  - Highly-parallel Bitap with long read support
  - Novel bitvector-based algorithm to perform traceback

- Co-design of our modified scalable and memory-efficient algorithms with low-power and area-efficient hardware accelerators
Use Cases & Key Results

(1) Read Alignment
- 116× speedup, 37× less power than Minimap2 (state-of-the-art SW)
- 111× speedup, 33× less power than BWA-MEM (state-of-the-art SW)
- 3.9× better throughput, 2.7× less power than Darwin (state-of-the-art HW)
- 1.9× better throughput, 82% less logic power than GenAx (state-of-the-art HW)

(2) Pre-Alignment Filtering
- 3.7× speedup, 1.7× less power than Shouji (state-of-the-art HW)

(3) Edit Distance Calculation
- 22–12501× speedup, 548–582× less power than Edlib (state-of-the-art SW)
- 9.3–400× speedup, 67× less power than ASAP (state-of-the-art HW)
Outline

- Introduction
- Background
  - Genome Sequencing & Genome Sequence Analysis
  - Approximate String Matching (ASM)
  - ASM with Bitap Algorithm
- GenASM: ASM Acceleration Framework
  - GenASM Algorithm
  - GenASM Hardware Design
  - Use Cases of GenASM
- Evaluation
- Conclusion
Genome Sequencing

- **Goal**: To determine the order of the DNA sequence (composed of A, C, G, Ts) in an organism’s genome.

- **Challenges**:
  - 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).
Genome Sequencing (cont’d.)

Large DNA molecule

Small DNA fragments

Reads

ACGTACCCCGT

ACGAGCGGGT

GATAACGTGTG

AAAAAATTTTTT

CTAGGGACCTT

ACGACG(TAGCT

AAAAAAA
Sequencing of COVID-19

Why genome sequencing and sequence data analysis are important?

- To detect the virus from a human sample
- To understand the sources and modes of transmission of the virus
- To sequence the genome of the virus itself, COVID-19, in order to track the mutations in the virus
- To explore the genes of infected patients
  - To understand why some people get more severe symptoms than others
  - To help with the development of new treatments
Future of Genome Sequencing & Analysis

MinION from ONT

SmidgION from ONT
1 Sequencing

Reference: TTTATCGCTTCCATGACGCAG
read1: ATCGC ATCC
read2: TATCG C ATC
read3: CATCCATGA
read4: CGCTTCCAT
read5: CCATGACGC
read6: TTCCATGAC

Genome Analysis

2 Read Mapping

Billions of Short Reads

3 Variant Calling

4 Scientific Discovery

Source: Prof. Onur Mutlu’s lecture slides
Source: Prof. Onur Mutlu’s lecture slides

Billions of Short Reads

Sequencing

Read Mapping

Bottlenecked in Mapping!!

Illumina HiSeq4000

300 M bases/min

on average

2 M bases/min

(0.6%)
Read Mapping

Reference genome → Indexing → Hash-table based index

Reads → Seeding → Potential mapping locations

Reference segment → Pre-Alignment Filtering → Remaining potential mapping locations

Query read → Pre-Alignment Filtering → Remaining potential mapping locations

Reads → Read Alignment → Optimal alignment
Approximate String Matching

- Sequenced genome may not exactly map to the reference genome due to genetic variations and sequencing errors

Reference: AAAAATGT TTAGTGCTACCTTG
Read: AAAAATGGTTTACTGCTACCTTG

- Approximate string matching (ASM):
  - Detect the differences and similarities between two sequences
  - In genomics, ASM is required to:
    - Find the minimum edit distance (i.e., total number of edits)
    - Find the optimal alignment with a traceback step
      - Sequence of matches, substitutions, insertions and deletions, along with their positions
      - 3M-1D-6M-1S-6M-1I-2M for the above example
  - Usually implemented as a dynamic programming (DP) based algorithm
Bitap Algorithm

- Bitap\textsuperscript{1,2} performs ASM with fast and simple bitwise operations
  - Amenable to efficient hardware acceleration
  - Computes the minimum edit distance between a text (e.g., reference genome) and a pattern (e.g., read) with a maximum of \( k \) errors

- Step 1: Pre-processing (per pattern)
  - Generate a pattern bitmask (PM) for each character in the alphabet (A, C, G, T)
  - Each PM indicates if character exists at each position of the pattern

- Step 2: Searching (Edit Distance Calculation)
  - Compare all characters of the text with the pattern by using:
    - Pattern bitmasks
    - Status bitvectors that hold the partial matches
    - Bitwise operations

---

Step 2: Edit Distance Calculation

For each character of the text (char):

Copy previous R bitvectors as oldR

For \( d = 1 \ldots k: \)

- deletion = \( \text{oldR}[d-1] \)
- substitution = \( \text{oldR}[d-1] \ll 1 \)
- insertion = \( \text{R}[d-1] \ll 1 \)
- match = \( (\text{oldR}[d] \ll 1) | \text{PM} \{\text{char}\} \)

\( \text{R}[d] = \text{deletion} \& \text{mismatch} \& \text{insertion} \& \text{match} \)

Check MSB of \( \text{R}[d] \):

- If 1, no match.
- If 0, match with \( d \) many errors.
Step 2: Edit Distance Calculation

For each character of the text (char):

- Copy previous R bitvectors as oldR
- R[0] = (oldR[0] << 1) | PM [char]

For d = 1...k:

- deletion = oldR[d-1]
- substitution = oldR[d-1] << 1
- insertion = R[d-1] << 1
- match = (oldR[d] << 1) | PM [char]

R[d] = deletion & mismatch & insertion & match

Check MSB of R[d]:

- If 1, no match.
- If 0, match with d many errors.
Bitap Algorithm (cont’d.)

- **Step 2: Edit Distance Calculation**

  For each character of the text (char):
  
  Copy previous R bitvectors as oldR
  
  \[
  R[0] = (\text{oldR}[0] \ll 1) \mid \text{PM}[\text{char}]
  \]
  
  For \( d = 1 \ldots k \):
  
  deletion \( = \text{oldR}[d-1] \)

  substitution \( = \text{oldR}[d-1] \ll 1 \)

  insertion \( = \text{R}[d-1] \ll 1 \)

  match \( = (\text{oldR}[d] \ll 1) \mid \text{PM}[\text{char}] \)

  \( R[d] = \text{deletion} \& \text{mismatch} \& \text{insertion} \& \text{match} \)

  Check MSB of \( R[d] \):

  - If 1, no match.
  - If 0, match with \( d \) many errors.

Data dependency between iterations (i.e., no parallelization)
Step 2: Edit Distance Calculation

For each character of the text (char):

- Copy previous R bitvectors as oldR
- \( R[0] = (\text{oldR}[0] << 1) \mid \text{PM}[\text{char}] \)

For \( d = 1 \ldots k: \)

- \( \text{deletion} = \text{oldR}[d-1] \)
- \( \text{substitution} = \text{oldR}[d-1] \ll 1 \)
- \( \text{insertion} = R[d-1] \ll 1 \)
- \( \text{match} = (\text{oldR}[d] \ll 1) \mid \text{PM}[\text{char}] \)

\( R[d] = \text{deletion} \& \text{mismatch} \& \text{insertion} \& \text{match} \)

Check MSB of \( R[d] \):

- If 1, no match.
- If 0, match with \( d \) many errors.

Does not store and process these intermediate bitvectors to find the optimal alignment (i.e., no traceback)
# Limitations of Bitap

1) **Data Dependency Between Iterations:**
   - Two-level data dependency forces the consecutive iterations to take place sequentially

2) **No Support for Traceback:**
   - Bitap does not include any support for optimal alignment identification

3) **No Support for Long Reads:**
   - Each bitvector has a length equal to the length of the pattern
   - Bitwise operations are performed on these bitvectors

4) **Limited Compute Parallelism:**
   - Text-level parallelism
   - Limited by the number of compute units in existing systems

5) **Limited Memory Bandwidth:**
   - High memory bandwidth required to read and write the computed bitvectors to memory
Outline

- Introduction
- Background
  - Genome Sequencing & Genome Sequence Analysis
  - Approximate String Matching (ASM)
  - ASM with Bitap Algorithm
- GenASM: ASM Acceleration Framework
  - GenASM Algorithm
  - GenASM Hardware Design
  - Use Cases of GenASM
- Evaluation
- Conclusion
GenASM: ASM Framework for GSA

- Approximate string matching (ASM) acceleration framework based on the Bitap algorithm
- First ASM acceleration framework for genome sequence analysis
- We overcome the five limitations that hinder Bitap’s use in genome sequence analysis:
  - Modified and extended ASM algorithm
    - Highly-parallel Bitap with long read support
    - Novel bitvector-based algorithm to perform traceback
  - Specialized, low-power and area-efficient hardware for both modified Bitap and novel traceback algorithms
GenASM Algorithm

- **GenASM-DC Algorithm:**
  - Modified Bitap for *Distance* Calculation
  - Extended for efficient *long read support*
  - Besides bit-parallelism that Bitap has, extended for parallelism:
    - Loop unrolling
    - Text-level parallelism

- **GenASM-TB Algorithm:**
  - Novel Bitap-compatible *TraceBack* algorithm
  - *Walks through the intermediate bitvectors* (match, deletion, substitution, insertion) generated by GenASM-DC
  - Follows a *divide-and-conquer approach* to decrease the memory footprint
GenASM Hardware Design

GenASM-DC: generates bitvectors and performs edit Distance Calculation

GenASM-TB: performs TraceBack and assembles the optimal alignment
GenASM Hardware Design

1. Host CPU
   - Reference & query locations

2. Main Memory
   - Reference text & query pattern

3. GenASM-DC Accelerator
   - Sub-text & sub-pattern

4. Generate bitvectors

5. Write bitvectors

6. Read bitvectors

7. Find the traceback output

GenASM-DC: generates bitvectors and performs edit distance calculation.

GenASM-TB: performs TraceBack and assembles the optimal alignment.
Our specialized compute units and on-chip SRAMs help us to:

→ Match the rate of computation with memory capacity and bandwidth

→ Achieve high performance and power efficiency

→ Scale linearly in performance with the number of parallel compute units that we add to the system
GenASM-DC: Hardware Design

- **Linear cyclic systolic array** based accelerator
  - Designed to **maximize parallelism and minimize memory bandwidth and memory footprint**

![Diagram of Processing Block (PB) and Processing Core (PC)]
GenASM-TB: Hardware Design

Very simple logic:

1. Reads the bitvectors from one of the TB-SRAMs using the computed address
2. Performs the required bitwise comparisons to find the traceback output for the current position
3. Computes the next TB-SRAM address to read the new set of bitvectors

Damla Senol Cali
Use Cases of GenASM

(1) **Read Alignment Step of Read Mapping**
   - Find the **optimal alignment** of how reads map to candidate reference regions

(2) **Pre-Alignment Filtering for Short Reads**
   - Quickly identify and **filter out the unlikely** candidate reference regions for each read

(3) **Edit Distance Calculation**
   - Measure the **similarity** or **distance** between two sequences

- We also discuss other possible use cases of GenASM in our paper:
  - Read-to-read overlap finding, hash-table based indexing, whole genome alignment, generic text search
Outline

- Introduction

- Background
  - Genome Sequencing & Genome Sequence Analysis
  - Approximate String Matching (ASM)
  - ASM with Bitap Algorithm

- GenASM: ASM Acceleration Framework
  - GenASM Algorithm
  - GenASM Hardware Design
  - Use Cases of GenASM

- Evaluation

- Conclusion
Evaluation Methodology

- We evaluate GenASM using:
  - Synthesized SystemVerilog models of the GenASM-DC and GenASM-TB accelerator datapaths
  - Detailed simulation-based performance modeling

- 16GB HMC-like 3D-stacked DRAM architecture
  - 32 vaults
  - 256GB/s of internal bandwidth, clock frequency of 1.25GHz
  - In order to achieve high parallelism and low power-consumption
  - Within each vault, the logic layer contains a GenASM-DC accelerator, its associated DC-SRAM, a GenASM-TB accelerator, and TB-SRAMs.
### Evaluation Methodology (cont’d.)

<table>
<thead>
<tr>
<th></th>
<th>SW Baselines</th>
<th>HW Baselines</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Read Alignment</strong></td>
<td>Minimap2&lt;sup&gt;1&lt;/sup&gt;</td>
<td>GACT (Darwin)&lt;sup&gt;3&lt;/sup&gt;</td>
</tr>
<tr>
<td></td>
<td>BWA-MEM&lt;sup&gt;2&lt;/sup&gt;</td>
<td>SillaX (GenAx)&lt;sup&gt;4&lt;/sup&gt;</td>
</tr>
<tr>
<td><strong>Pre-Alignment Filtering</strong></td>
<td>–</td>
<td>Shouji&lt;sup&gt;5&lt;/sup&gt;</td>
</tr>
<tr>
<td><strong>Edit Distance Calculation</strong></td>
<td>Edlib&lt;sup&gt;6&lt;/sup&gt;</td>
<td>ASAP&lt;sup&gt;7&lt;/sup&gt;</td>
</tr>
</tbody>
</table>

For Use Case 1: Read Alignment, we compare GenASM with:

- Minimap2 and BWA-MEM (state-of-the-art **SW**)
  - Running on Intel® Xeon® Gold 6126 CPU (12-core) operating @2.60GHz with 64GB DDR4 memory
  - Using two simulated datasets:
    - Long ONT and PacBio reads: 10Kbp reads, 10-15% error rate
    - Short Illumina reads: 100-250bp reads, 5% error rate

- GACT of Darwin and SillaX of GenAx (state-of-the-art **HW**)
  - Open-source RTL for GACT
  - Data reported by the original work for SillaX
  - GACT is best for long reads, SillaX is best for short reads
For Use Case 2: Pre-Alignment Filtering, we compare GenASM with:

- **Shouji** (state-of-the-art HW – FPGA-based filter)
  - Using two datasets provided as test cases:
    - 100bp reference-read pairs with an edit distance threshold of 5
    - 250bp reference-read pairs with an edit distance threshold of 15

For Use Case 3: Edit Distance Calculation, we compare GenASM with:

- **Edlib** (state-of-the-art SW)
  - Using two 100Kbp and 1Mbp sequences with similarity ranging between 60%-99%

- **ASAP** (state-of-the-art HW – FPGA-based accelerator)
  - Using data reported by the original work
Key Results – Area and Power

- Based on our synthesis of GenASM-DC and GenASM-TB accelerator datapaths using the Synopsys Design Compiler with a 28nm LP process:
  - Both GenASM-DC and GenASM-TB operate @ 1GHz

### Area (mm²)

- GenASM-DC (64 PEs): 0.049
- GenASM-TB: 0.016
- DC-SRAM (8 KB): 0.013
- TB-SRAMs (64 x 1.5 KB): 0.256

**Total (1 vault):** 0.334 mm²

**Total (32 vaults):** 10.69 mm²

% of a Xeon CPU core: 1%

### Power (W)

- GenASM-DC (64 PEs): 0.033
- GenASM-TB: 0.004
- DC-SRAM (8 KB): 0.009
- TB-SRAMs (64 x 1.5 KB): 0.055

**Total (1 vault):** 0.101 W

**Total (32 vaults):** 3.23 W

% of a Xeon CPU core: 1%
Based on our synthesis of GenASM-DC and GenASM-TB accelerator datapaths using the Synopsys Design Compiler with a 28nm LP process:

- Both GenASM-DC and GenASM-TB operate @ 1GHz

GenASM has **low area and power overheads**

### Key Results – Area and Power

- **Area (mm²):**
  - GenASM-DC (64 PEs): 0.049
  - GenASM-TB: 0.013
  - DC-SRAM (8 KB): 0.256
  - TB-SRAMs (64 x 1.5 KB): 0.055

- **Power (W):**
  - GenASM-DC (64 PEs): 0.033
  - GenASM-TB: 0.004
  - DC-SRAM (8 KB): 0.009
  - TB-SRAMs (64 x 1.5 KB): 0.009
Key Results – Use Case 1

(1) Read Alignment Step of Read Mapping
  o Find the optimal alignment of how reads map to candidate reference regions

(2) Pre-Alignment Filtering for Short Reads
  o Quickly identify and filter out the unlikely candidate reference regions for each read

(3) Edit Distance Calculation
  o Measure the similarity or distance between two sequences
GenASM achieves 648× and 116× speedup over 12-thread runs of BWA-MEM and Minimap2, while reducing power consumption by 34× and 37×
Key Results – Use Case 1 (Long Reads)

GenASM provides 3.9× better throughput, 6.6× the throughput per unit area, and 10.5× the throughput per unit power, compared to GACT of Darwin.
**Key Results – Use Case 1 (Short Reads)**

**SW**
GenASM achieves **111×** and **158×** speedup over 12-thread runs of BWA-MEM and Minimap2, while reducing power consumption by **33×** and **31×**

**HW**
GenASM provides **1.9× better throughput** and uses **63% less logic area** and **82% less logic power**, compared to SillaX of GenAx
Key Results – Use Case 2

(1) Read Alignment Step of Read Mapping
   - Find the optimal alignment of how reads map to candidate reference regions

(2) Pre-Alignment Filtering for Short Reads
   - Quickly identify and filter out the unlikely candidate reference regions for each read

(3) Edit Distance Calculation
   - Measure the similarity or distance between two sequences
Key Results – Use Case 2

- Compared to Shouji:
  - 3.7× speedup
  - 1.7× less power consumption
  - False accept rate of 0.02% for GenASM vs. 4% for Shouji
  - False reject rate of 0% for both GenASM and Shouji

GenASM is more efficient in terms of both speed and power consumption, while significantly improving the accuracy of pre-alignment filtering.
(1) Read Alignment Step of Read Mapping
   - Find the optimal alignment of how reads map to candidate reference regions

(2) Pre-Alignment Filtering for Short Reads
   - Quickly identify and filter out the unlikely candidate reference regions for each read

(3) Edit Distance Calculation
   - Measure the similarity or distance between two sequences
**Key Results – Use Case 3**

**SW**
GenASM provides 146 – 1458× and 627 – 12501× speedup, while reducing power consumption by 548× and 582× for 100Kbp and 1Mbp sequences, respectively, compared to Edlib.

**HW**
GenASM provides 9.3 – 400× speedup over ASAP, while consuming 67× less power.
Outline

- Introduction
- Background
  - Genome Sequencing & Genome Sequence Analysis
  - Approximate String Matching (ASM)
  - ASM with Bitap Algorithm
- GenASM: ASM Acceleration Framework
  - GenASM Algorithm
  - GenASM Hardware Design
  - Use Cases of GenASM
- Evaluation
- Conclusion
Additional Details in the Paper

- Details of the GenASM-DC and GenASM-TB algorithms
- Big-O analysis of the algorithms
- Detailed explanation of evaluated use cases
- Evaluation methodology details (datasets, baselines, performance model)
- Additional results for the three evaluated use cases
- Sources of improvements in GenASM (algorithm-level, hardware-level, technology-level)
- Discussion of four other potential use cases of GenASM
Conclusion

Problem:
- Genome sequence analysis is bottlenecked by the computational power and memory bandwidth limitations of existing systems
- This bottleneck is particularly an issue for approximate string matching

Key Contributions:
- GenASM: An approximate string matching (ASM) acceleration framework to accelerate multiple steps of genome sequence analysis
  - First to enhance and accelerate Bitap for ASM with genomic sequences
  - Co-design of our modified scalable and memory-efficient algorithms with low-power and area-efficient hardware accelerators
  - Evaluation of three different use cases: read alignment, pre-alignment filtering, edit distance calculation

Key Results: GenASM is significantly more efficient for all the three use cases (in terms of throughput and throughput per unit power) than state-of-the-art software and hardware baselines
Damla Senol Cali, Gurpreet S. Kalsi, Zulal Bingol, Can Firtina, Lavanya Subramanian, Jeremie S. Kim, Rachata Ausavarungnirun, Mohammed Alser, Juan Gomez-Luna, Amirali Boroumand, Anant Nori, Allison Scibisz, Sreenivas Subramoney, Can Alkan, Saugata Ghose, and Onur Mutlu,

“GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis”


GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis

Damla Senol Cali†, Gurpreet S. Kalsi‡, Zülal Bingöl♂, Can Firtina○, Lavanya Subramanian‡, Jeremie S. Kim♂, Rachata Ausavarungnirun♂, Mohammed Alser♂, Juan Gomez-Luna♂, Amirali Boroumand†, Anant Nori♂, Allison Scibisz†, Sreenivas Subramoney♂, Can Alkan♂, Saugata Ghose♂, Onur Mutlu♂,

†Carnegie Mellon University  ‡Processor Architecture Research Lab, Intel Labs  ○Bilkent University  ♂ETH Zürich
♂Facebook  ○King Mongkut’s University of Technology North Bangkok  *University of Illinois at Urbana–Champaign
Discussion

- GenASM for **generic text search**
  - Any other use cases?

- Most efficient **porting locations** of GenASM accelerators

- What about GenASM algorithms?
  - GPU mapping?
  - FPGA mapping?

- **Portable** sequencing devices + **low-power, memory-efficient** designs for sequence analysis

- **HW/SW co-design** for other emerging applications/domains
P&S Mobile Genomics
Lecture 12: GenASM

Dr. Damla Senol Cali
ETH Zurich
Fall 2022
16 January 2023
Backup Slides

(GenASM)
Genome Sequence Analysis

**Read Mapping**, method of aligning the reads against the reference genome in order to detect matches and variations.

**De novo Assembly**, method of merging the reads in order to construct the original sequence.
Read Mapping

- **Read mapping**: *First key step* in genome sequence analysis
  - Align reads to one or more possible locations within the reference genome and
  - Find the matches and differences between the read and the reference genome segment at that location
Approximate String Matching (ASM)

Approximate string matching algorithms:
- *Smith-Waterman (SW)* algorithm [Smith+, Advances in Applied Mathematics 1981]
  - Dynamic programming (DP) algorithm, with quadratic time and space complexity
  - Common algorithm used by read mappers

  - Transformed version of SW algorithm into bitvectors and bitwise operations

  - [Wu+, Communications of the ACM 1992] extended *Bitap* to perform approximate string matching
  - Bitvectors and bitwise operations

We have focused on the *Bitap* algorithm.
→ Reason: *Bitap* algorithm can perform ASM with **fast and simple bitwise operations**, which makes it amenable to efficient hardware acceleration.
Example for the Bitap Algorithm

PREPROCESSING

<table>
<thead>
<tr>
<th>Pattern Bitmasks:</th>
<th>CTGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>PM(A) = 1110</td>
<td></td>
</tr>
<tr>
<td>PM(C) = 0111</td>
<td></td>
</tr>
<tr>
<td>PM(G) = 1101</td>
<td></td>
</tr>
<tr>
<td>PM(T) = 1011</td>
<td></td>
</tr>
</tbody>
</table>

State Vectors:
- R0 = 1111
- R1 = 1111

TEXT

Text[0]: CGTGA
- oldR0 = 1111
- oldR1 = 1111

R0 = (oldR0 << 1) | PM(A) = 1110
- D : oldR0 = 1111
- S : oldR0 << 1 = 1110
- I : R0 << 1 = 1100
- M : (oldR1 << 1) | PM(A) = 1110
  = D & S & I & M = 1110

Text[1]: CGTGA
- oldR0 = 1011
- oldR1 = 0000

R0 = (oldR0 << 1) | PM(G) = 1111
- D : oldR0 = 1011
- S : oldR0 << 1 = 1010
- I : R0 << 1 = 0110
- M : (oldR1 << 1) | PM(T) = 1011
  = D & S & I & M = 0000

Alignment Found @ Location=2

Text[2]: CGTGA
- oldR0 = 1111
- oldR1 = 1000

R0 = (oldR0 << 1) | PM(T) = 1011
- D : oldR0 = 1101
- S : oldR0 << 1 = 1010
- I : R0 << 1 = 0110
- M : (oldR1 << 1) | PM(T) = 1011
  = D & S & I & M = 0000

Alignment Found @ Location=1

Text[3]: CGTGA
- oldR0 = 1110
- oldR1 = 1100

R0 = (oldR0 << 1) | PM(G) = 1110
- D : oldR0 = 1110
- S : oldR0 << 1 = 1110
- I : R0 << 1 = 1000
- M : (oldR1 << 1) | PM(A) = 1110
  = D & S & I & M = 1000

Alignment Found @ Location=0

Text[4]: CGTGA
- oldR0 = 1111
- oldR1 = 1111

R0 = (oldR0 << 1) | PM(A) = 1110
- D : oldR0 = 1111
- S : oldR0 << 1 = 1110
- I : R0 << 1 = 1100
- M : (oldR1 << 1) | PM(A) = 1110
  = D & S & I & M = 1100

Alignment Found @ Location=2

Text[5]: CGTGA
- oldR0 = 1111
- oldR1 = 0000

R0 = (oldR0 << 1) | PM(C) = 1111
- D : oldR0 = 1111
- S : oldR0 << 1 = 1110
- I : R0 << 1 = 1110
- M : (oldR1 << 1) | PM(T) = 1110
  = D & S & I & M = 1110

Alignment Found @ Location=0
## Loop Unrolling in GenASM-DC

<table>
<thead>
<tr>
<th>Cycle#</th>
<th>Thread1 Ro/1/2/…</th>
<th>Thread2 R0/4</th>
<th>Thread3 R1/5</th>
<th>Thread4 R2/6</th>
<th>Thread5 R3/7</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>T0-R0</td>
<td>T0-R0</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>T1-R1</td>
<td>T0-R2</td>
<td>T1-R3</td>
<td></td>
</tr>
<tr>
<td>#8</td>
<td>T0-R7</td>
<td>T2-R1</td>
<td>T2-R2</td>
<td>T2-R3</td>
<td></td>
</tr>
<tr>
<td>#9</td>
<td>T1-R0</td>
<td>T3-R0</td>
<td>T3-R2</td>
<td>T3-R3</td>
<td></td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>T2-R4</td>
<td>T0-R6</td>
<td>T0-R7</td>
<td></td>
</tr>
<tr>
<td>#16</td>
<td>T1-R7</td>
<td>T2-R5</td>
<td>T2-R6</td>
<td>T2-R7</td>
<td></td>
</tr>
<tr>
<td>#17</td>
<td>T2-R0</td>
<td>T3-R5</td>
<td>T3-R6</td>
<td>T3-R7</td>
<td></td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td></td>
</tr>
<tr>
<td>#32</td>
<td>T3-R7</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td></td>
</tr>
</tbody>
</table>

- Data written to memory
- Data read from memory

Target cell (R<sub>d</sub>): cells target cell depends on (oldR<sub>d</sub>, R<sub>d-1</sub>, oldR<sub>d-1</sub>)
# Traceback Example with GenASM-TB

## Deletion Example (Text Location=0)

<table>
<thead>
<tr>
<th>Text[0]: C</th>
<th>Text[1]: G</th>
<th>Text[2]: T</th>
<th>Text[3]: G</th>
<th>Text[4]: A</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0- :.....</td>
<td>R0- :.....</td>
<td>R0-M : 1011</td>
<td>R0-M : 1101</td>
<td>R0-M : 1110</td>
</tr>
<tr>
<td>R1-M : 0111</td>
<td>R1-D : 1011</td>
<td>R1- :.....</td>
<td>R1- :.....</td>
<td>R1- :.....</td>
</tr>
<tr>
<td>Match(C)</td>
<td>Del(-)</td>
<td>Match(T)</td>
<td>Match(G)</td>
<td>Match(A)</td>
</tr>
<tr>
<td>&lt;3,0,1&gt;</td>
<td>&lt;2,1,1&gt;</td>
<td>&lt;2,2,0&gt;</td>
<td>&lt;1,3,0&gt;</td>
<td>&lt;0,4,0&gt;</td>
</tr>
</tbody>
</table>

## Substitution Example (Text Location=1)

<table>
<thead>
<tr>
<th>Text[1]: G</th>
<th>Text[2]: T</th>
<th>Text[3]: G</th>
<th>Text[4]: A</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0- :.....</td>
<td>R0-M : 1011</td>
<td>R0-M : 1101</td>
<td>R0-M : 1110</td>
</tr>
<tr>
<td>R1-S : 0110</td>
<td>R1- :.....</td>
<td>R1- :.....</td>
<td>R1- :.....</td>
</tr>
<tr>
<td>Subs(C)</td>
<td>Match(T)</td>
<td>Match(G)</td>
<td>Match(A)</td>
</tr>
<tr>
<td>&lt;3,1,1&gt;</td>
<td>&lt;2,2,0&gt;</td>
<td>&lt;1,3,0&gt;</td>
<td>&lt;0,4,0&gt;</td>
</tr>
</tbody>
</table>

## Insertion Example (Text Location=2)

<table>
<thead>
<tr>
<th>Text[-]</th>
<th>Text[2]: T</th>
<th>Text[3]: G</th>
<th>Text[4]: A</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0- :.....</td>
<td>R0-M : 1011</td>
<td>R0-M : 1101</td>
<td>R0-M : 1110</td>
</tr>
<tr>
<td>R1-I : 0110</td>
<td>R1- :.....</td>
<td>R1- :.....</td>
<td>R1- :.....</td>
</tr>
<tr>
<td>Ins(C)</td>
<td>Match(T)</td>
<td>Match(G)</td>
<td>Match(A)</td>
</tr>
<tr>
<td>&lt;3,2,1&gt;</td>
<td>&lt;2,2,0&gt;</td>
<td>&lt;1,3,0&gt;</td>
<td>&lt;0,4,0&gt;</td>
</tr>
</tbody>
</table>
Backup Slides
(Sequencing)
Short Reads vs. Long Reads

➢ Short Reads
  ❑ Sequences with tens to hundreds of bases
  ❑ Highly accurate sequences
  ❑ Output of SRS technologies (e.g., Illumina, Ion Torrent)

➢ Long reads
  ❑ Sequences with thousands or millions of bases
  ❑ Sequences with high error rates
  ❑ Output of LRS technologies (e.g., Oxford Nanopore Technologies, PacBio)
Cost of Sequencing

*From NIH (https://www.genome.gov/about-genomics/fact-sheets/DNA-Sequencing-Costs-Data)
Cost of Sequencing (cont’d.)

*From NIH (https://www.genome.gov/about-genomics/fact-sheets/DNA-Sequencing-Costs-Data)
COVID-19 Sequencing with ONT

- From ONT (https://nanoporetech.com/covid-19/overview)
COVID-19 Sequencing with ONT (cont’d.)

How are scientists using nanopore sequencing to research COVID-19?

Samples are collected → Validated SARS-CoV-2 RT-PCR test performed → SARS-CoV-2 positive samples

- SARS-CoV-2 negative samples: used as negative controls

How can this be used?
- Genomic epidemiology: analyse variants & mutation rate, track spread of virus, identity clusters of transmission

What are the results?
- From RNA to full SARS-CoV-2 consensus sequence in ~7 hours
- Targeted amplification of SARS-CoV-2 genome + multiplexed, rapid nanopore sequencing
- RNA: data for RNA viruses (including SARS-CoV-2) + microbial transcripts
- DNA: data for bacteria + DNA viruses

How?Targeted SARS-CoV-2 nanopore sequencing

Metagenomic nanopore sequencing

- 1 x RNA metagenomic sequencing run
- 1 x DNA metagenomic sequencing run

What are the results?

- SARS-CoV-2 Direct RNA whole genome sequencing: assess viral genome in its native RNA form and the effect of base modifications
- Immune repertoire: assess response of the immune system to SARS-CoV-2 infection by sequencing of full-length immune cell receptor genes and transcripts
- Whole human genome sequencing: investigate what might cause different responses to the virus in different people based on their genome

How can this be used?
- Characterise co-infecting bacteria & viruses, identify any correlation of risk factors, research potential future treatment implications

What’s next?

Find out more at nanoporetech.com/covid19

- From ONT (https://nanoporetech.com/covid-19/overview)
Nanopore Genome Assembly Pipeline

- **Basecalling**
  - Raw signal data
  - DNA reads

- **Read-to-Read Overlap Finding**
  - DNA reads
  - Overlaps

- **Assembly**
  - Overlaps
  - Draft assembly

- **Read Mapping (Optional)**
  - Draft assembly
  - Mappings of reads against draft assembly

- **Polishing (Optional)**
  - Mappings of reads against draft assembly

- **Improved assembly**

---

Damla Senol Cali
Nanopore Sequencing Technology and Tools for Genome Assembly: Computational Analysis of the Current State, Bottlenecks and Future Directions

Damla Senol Cali\textsuperscript{1,*}, Jeremie S. Kim\textsuperscript{1,3}, Saugata Ghose\textsuperscript{1}, Can Alkan\textsuperscript{2,*} and Onur Mutlu\textsuperscript{3,1*}

\textsuperscript{1}Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, USA
\textsuperscript{2}Department of Computer Engineering, Bilkent University, Bilkent, Ankara, Turkey
\textsuperscript{3}Department of Computer Science, Systems Group, ETH Zürich, Zürich, Switzerland