GenAx: A Genome Sequencing Accelerator

Daichi Fujiki*  Arun Subramaniyan*  Tianjun Zhang*  Yu Zeng
Reetuparna Das  David Blaauw  Satish Narayanasamy

* Equally contributed to the paper
Genomics is set to transform medicine

Population-based treatment

Personalized treatment
Genome sequencing costs have plummeted

“Illumina says it can deliver a $100 genome — soon”
Portable sequencers are becoming commonplace
Sequence analysis has several computational steps

- **Reference genome**
  - ATCGTGAGTT
  - CGTAAGAG
  - TTTT

- **Reads**
  - ATCGTGAGTT
  - GTGCATCTAC
  - CAATCACTAC
  - ATCGTGCTAC

- **Aligned reads**
  - ATCGTGAGTT
  - CGTAAGAG

- **Variant Calling**
  - ATCGTGAGTT
  - CGTAAGAG

- **Diagnosis**

---

**Human Genome**

- 3 G bases
- Sequenced reads (~billions)

**Secondary Analysis**

- (350 genomes/week) per machine
- (5.6 genomes/week) per server
Read alignment is a major bottleneck in sequence analysis.

Seeding – a Filtration Step

Reference Genome

Seed

Read

AATA

0 52 103 512
Seed Extension

Reference Genome

Seed

Read

Candidate Reference Strings

Score
Seed Extension

Reference Genome

Seed

Read

Candidate Reference Strings | Score
---|---
GAATA-CTA-AATTTAT | 15
G--AATA-C---TTTAT | 11
AAATACCTAAAATTTAT | 17

Read | AAATACCTAAAATTTAT
Seed extension as approximate string matching

**Levenshtein (edit) distance**: minimum number of edits (insertions, substitutions, deletions) required to perfectly match the read (or query string Q) and reference string R.
Genome Sequencing – Alignment Methodology

Smith Waterman Matrix

Initialize the scoring matrix

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

Substitution matrix:

\[
S(a_i, b_j) = \begin{cases} 
+3, & a_i = b_j \\
-3, & a_i \neq b_j 
\end{cases}
\]

Gap penalty:

\[
W_k = kW_1 \\
W_1 = 2
\]

©Wikimedia commons
Genome Sequencing – Alignment Methodology

Smith Waterman Matrix
Genome Sequencing – Alignment Methodology

Smith Waterman Matrix
Genome Sequencing – Alignment Methodology

Banded Smith Waterman Matrix

$O(kn)$
Genome Sequencing – Alignment Methodology

Banded Smith Waterman Matrix

Levenshtein Automata
Genome Sequencing – Alignment Methodology

Banded Smith Waterman Matrix

Levenshtein Automata
Genome Sequencing – Alignment Methodology

Banded Smith Waterman Matrix

Levenshtein Automata
Genome Sequencing – Alignment Methodology

Banded Smith Waterman Matrix

New Automaton Algorithm

O(kn)

\[ O(k^2) \]

String independent

Silla: String Independent Local Levenshtein Automata

\[ \Delta_{i,d} = R[c-i] \ XNOR \ Q[c-d] \]
**Algorithm Contribution**

- **Silla**: String Independent Local Levenshtein Automata

  \[ \Delta_{1,d} = R[c-i] \ XOR \ Q[c-d] \]

  \[ D_{i,d} = R[c-i] \ XOR \ Q[c-d] \]

**Hardware Implementation**

- **SillaX**: Silla Accelerator for Genome Sequencing

- **SMEM + Hash based seeding algorithm**

  - Index Table
  - Position Table
  - 512 entry CAMs

- **Binary search**
Algorithm Contribution

**Silla**: String Independent Local Levenshtein Automata

\[ \Delta_{i,d} = R[c-i] \text{ XNOR } Q[c-d] \]

- **K = 1**
  - del
  - match
  - ins

- **K = 2**
  - del
  - ins

**Algorithm**: SMEM + Hash based seeding algorithm

Hardware Implementation

**SillaX**: Silla Accelerator for Genome Sequencing

- Index Table
- Position Table
- Segmenting

**Hardware Optimization**

SMEM + Hash based Seeding Accelerator

512 entry CAM
512 entry CAM
512 entry CAM
512 entry CAM
512 entry CAM
**Indel Silla**

Reference: \( \text{AGTA} \text{TGCCATT} \)

Query: \( \text{AGTA} \text{TACCATT} \)

**Cycle c**

**Indel Silla**

**Silla**: String Independent Local Levenshtein Automaton
Indel Silla

Silla: String Independent Local Levenshtein Automaton
String Independent

Exact match

Reference

Query

Cycle c

Indel Silla

Silla: String Independent Local Levenshtein Automaton
Silla: String Independent Local Levenshtein Automaton
**Silla: String Independent Local Levenshtein Automaton**
**String Independent**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGTAAGCCTTA</td>
<td>AGTATTGCCATTTA</td>
</tr>
</tbody>
</table>

Key observation:
3D Silla $\cong$ 2 layer Silla

Silla: String Independent Local Levenshtein Automaton

$O(k^3)$
Can we merge these nodes?

**Key observation (2)**

$\Delta_{1,1|0}$ will see the same characters as $\Delta_{0,0|2}$ in the future.

**3D Silla $\cong$ 2 layer Silla**

**Collapsing 3D Silla**
Can we merge these nodes?

**Key observation (2)**

$\Delta_{1,1|0}$ will see the same characters as $\Delta_{0,0|2}$ in the future

Collapsing 3D Silla

**Silla:** String Independent Local Levenshtein Automaton
Can we merge these nodes?

**Key observation (2)**

\[ \Delta_{1,1|0} = \Delta_{0,0|1} = \Delta_{0,0|2} \]

→ Insert Wait node

Collapsing 3D Silla

**Silla:** String Independent Local Levenshtein Automaton
Can we merge these nodes?

**Key observation (2)**

\[ \Delta_{1,1|0} \] will see the same characters as \[ \Delta_{0,0|2} \] in the future.

→ Insert Wait node

**Collapsing 3D Silla**

**Silla:** String Independent Local Levenshtein Automaton
Can we merge these nodes?

**Key observation (2)**

\[ \Delta_{1,1|0} = \Delta_{0,0|1} = \Delta_{0,0|2} \]

- \( \Delta_{1,1|0} \) will see the same characters as \( \Delta_{0,0|2} \) in the future

→ Insert Wait node: \( i, d \)

**Collapsing 3D Silla**

**Silla: String Independent Local Levenshtein Automaton**
Silla: String Independent Local Levenshtein Automaton
**Problem statement**
Some nodes require long wires form comparators / other nodes

**Silla:** String Independent Local Levenshtein Automaton
Local

Reference: $\text{AGTATTACATTA}$

Query: $\text{AGTATTACATTA}$

Cycle $c$

- $\Delta_{0,0}$

- $\Delta_{1,0}$

- $\Delta_{0,1}$

- $\Delta_{1,1}$

$\Delta_{0,0}(t=c) = \Delta_{1,1}(t=c+1) = \Delta_{2,2}(t=c+2) = \ldots$

Comparators

Pipelined Data paths

Silla: String Independent Local Levenshtein Automaton
Local

Reference: AGTA AGCCATTA

Query: AGTA TGCATTA

Cycle c

Reference: AGTA AGCCATTA

Query: AGTA TGCATTA

Cycle c+1

$\Delta_{0,0}(t=c) = \Delta_{1,1}(t=c+1) = \Delta_{2,2}(t=c+2) = \ldots$

Pipelined Data paths

Comparators

Silla: String Independent Local Levenshtein Automaton
Local

Reference  
AGTA  AGCCATTA

Query  
AGTA  TGCATT

Δ₀,₀

Cycle c

Reference  
AGTA  AGCCATTA

Query  
AGTA  TGCATT

Δ₁,₁

Δ₀,₀

Cycle c+1

Δ₀,₀(t=c) = Δ₁,₁(t=c+1) = Δ₂,₂(t=c+2) = ...

Comparators

Pipelined Data paths

Silla: String Independent Local Levenshtein Automaton
\[ \Delta_{0,0}(t=c) = \Delta_{1,1}(t=c+1) = \Delta_{2,2}(t=c+2) = \ldots \]
GenAx: A Genome Sequencing Accelerator

Hardware Implementation
### In-place Traceback

<table>
<thead>
<tr>
<th></th>
<th>T</th>
<th>G</th>
<th>T</th>
<th>T</th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>9</td>
<td>7</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>9</td>
<td>7</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>4</td>
<td>7</td>
<td>6</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>10</td>
<td>8</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>8</td>
<td>13</td>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>7</td>
<td>9</td>
<td>8</td>
</tr>
</tbody>
</table>

### Traceback Recap

![Traceback Recap Diagram](image)

### Hardware Implementation
In-place Traceback

No External Traceback Memory
Traceback Information is stored in nodes
Greedy approach

-string Matching Phase
  Best score is propagated
  Traceback Pointer is kept in nodes
  Traceback Pointer is updated with better score

Traceback Phase
  Pointer Trailing from the node with best score

Traceback Machine

Hardware Implementation
In-place Traceback

No External Traceback Memory
- Traceback Information is stored in nodes
- Greedy approach

String Matching Phase
- Best score is propagated
- Traceback Pointer is kept in nodes
- Traceback Pointer is updated with better score

Traceback Phase
- Pointer Trailing from the node with best score

Traceback Machine

Hardware Implementation
In-place Traceback

No External Traceback Memory
- Traceback Information is stored in nodes
- Greedy approach

String Matching Phase
- Best score is propagated
- Traceback Pointer is kept in nodes
- Traceback Pointer is updated with better score

Traceback Phase
- Pointer Trailing from the node with best score

Traceback Machine

Hardware Implementation
In-place Traceback

Broken Pointer Trail

- Previous best score is updated and breaks the path
- Rerun the machine till the cycle
- 7.6% of reads require rerun

Hardware Implementation
In-place Traceback

Broken Pointer Trail

- Previous best score is updated and breaks the path
- Rerun the machine till the cycle
- 7.6% of reads require rerun

Hardware Implementation
In-place Traceback

Broken Pointer Trail

- Previous best score is updated and breaks the path
- Rerun the machine till the cycle
- 7.6% of reads require rerun

Hardware Implementation
In-place Traceback

Broken Pointer Trail

- Previous best score is updated and breaks the path
- Rerun the machine till the cycle
- 7.6% of reads require rerun

Hardware Implementation
GenAx: A Genome Sequencing Accelerator
Affine gap Scoring

<table>
<thead>
<tr>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Match</td>
</tr>
<tr>
<td>Mismatch</td>
</tr>
<tr>
<td>Gap Opening</td>
</tr>
<tr>
<td>Gap Extension</td>
</tr>
</tbody>
</table>

Trace 1: M M M I I M I I I M M
Edit distance = 5
Score = -11
BAD

Trace 2: M M M M I I I I I M M
Edit distance = 5
Score = -5
GOOD

Hardware Implementation
Affine gap Scoring

<table>
<thead>
<tr>
<th></th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Match</td>
<td>+1</td>
</tr>
<tr>
<td>Mismatch</td>
<td>-4</td>
</tr>
<tr>
<td>Gap Opening</td>
<td>-7</td>
</tr>
<tr>
<td>Gap Extension</td>
<td>-1</td>
</tr>
</tbody>
</table>

Trace 1

M M M I I M I I I M M

Edit distance = 5
Score = -11

Trace 2

M M M M I I I I I M M

Edit distance = 5
Score = -5

Hardware Implementation
Affine gap Scoring

<table>
<thead>
<tr>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Match</td>
</tr>
<tr>
<td>Mismatch</td>
</tr>
<tr>
<td>Gap Opening</td>
</tr>
<tr>
<td>Gap Extension</td>
</tr>
</tbody>
</table>

**Trace 1:** 
Edit distance = 5
Score = -11
**BAD**

**Trace 2:** 
Edit distance = 5
Score = -5
**GOOD**

Composability

Composable to large edit distances

Hardware Implementation
Hardware Implementation

GenAx: A Genome Sequencing Accelerator

Seeding machine
- Index Table
- Position Table
- SMEM + Hash based Seeding Accelerator

SillaX
- In-place Traceback
- Affine gap Scoring
- Composability

512 entry CAM
512 entry CAM
512 entry CAM
512 entry CAM
512 entry CAM

Query
Reference
match
del
ins
0,0
1,0
2,0
0,1
1,1
0,2
Identifying super-maximal exact matches for a read

H1 = \{ 10, 100, 390 \}

CAM
- 10
- 100
- 390

k-mer-1 hits

Reference

AGTA

Read

0

Index Table

Position Table

10
100
390

ATGCCATG

H1 = \{ 10, 100, 390 \}
Identifying super-maximal exact matches for a read

H1 = { 10, 100, 390 } ∩ H2 = { 26, 100, 390 }

k-mer-2 hits: 390, 100, 26
k-mer-1 hits: 390, 100
Identifying super-maximal exact matches for a read

\[ H_1 = \{ 10, 100, 390 \} \]
\[ H_2 = \{ 26, 100, 390 \} \]
\[ H_3 = \{ 26, 100 \} \]

\[ \cap \]
\[ \cap \]

k-mer-3 hits

k-mer-1 hits

\[ \text{SMEM} = \{ 100 \} \]
Seeding implementation: Key ideas

1. Binary search-based intersection for frequent k-mers

Intersecting m hits of k-mer-1 with n hits of k-mer-2

\[ O(m \log n) \] steps
Seeding implementation: Key ideas

2 Probing: Intersect from k-mer with minimum number of hits

Read AAAAAAAAAAAAAAAGTAATGCCATGATGCCGTATGAATGCAAGT

# Hits

Intersection:

1. Intersect from k-mer with minimum number of hits

2. Intersect from k-mer with minimum number of hits

Hit counts:

1. 100
2. 251
3. 100

# Hits

1. 1
2. 2

Correct solution:

- The correct solution has 2 hits.

Incorrect solution:

- The incorrect solution has only 1 hit.
Methodology

- Input
  
  Reference Genome: GRCh38 (Human Genome) from UCSC genome browser
  Input Reads: Illumina Platinum Genomes (50x, 787M reads, 101bp)

- Baseline
  
  CPU: BWA-MEM on Intel Xeon E5-2697 (2.6GHz, 56threads) + 128GB DDR4
  GPU: CUSHAW2-GPU on NVIDIA Titan Xp (1.58GHz, 3840cores) + 12GB GDDR5X

- SillaX configuration
  
  Synthesis: Synopsis Design Compiler, 28nm process, 2GHz → 5.64mm², 6.6W
  Bandwidth (K): 40

- Seeding machine configuration
  
  K-mer size: 12
  Segmenting: 512 segments, 48MB index table and 18MB position table
Performance (Throughput / Power)

**GenAx**

- Throughput (Kreads/sec)
  - BWA-MEM: 56K
  - CUSHAW2 GPU: 4,058K
  - GenAx: 4,058K

- Avg. Power (W)
  - BWA-MEM: 128K
  - CUSHAW2 GPU: 12x
  - GenAx: 12x

**SillaX**

- Throughput (Khits/s)
  - Illumina 100: 63x over CPU
  - 5000x over GPU

Baseline
- CPU: SeqAn Library
- GPU: SW#
Conclusion

Contributions

- **Silla** – a novel automaton algorithm for approximate string matching
  - $O(k^2)$ complexity
  - Naturally maps to systolic array / automaton accelerator

- **SillaX** – a seed extension accelerator
  - Affine gap + Traceback + Composable

- **GenAx** – a read alignment accelerator
  - Drop-in replacement of BWA-MEM

Results

- **31.7x** speedup over BWA-MEM on 56-thread Xeon processor
- **12x** power reduction
- **5.6x** area reduction
“Discover the genetic, lifestyle and environmental factors that influence a population’s health and provides personalized solutions that allow individuals to improve their health and wellness.”
GenAx: A Genome Sequencing Accelerator

Daichi Fujiki  Arun Subramaniyan  Tianjun Zhang  Yu Zeng
Reetuparna Das  David Blaauw  Satish Narayanasamy

Thank you.
Any questions?