Accelerating Genome Sequence Analysis via Efficient Hardware/Algorithm Co-Design

Damla Senol Cali
Ph.D. Thesis Defense - July 15, 2021
dsenol@andrew.cmu.edu

Committee:  Prof. Onur Mutlu (CMU, ETH Zurich)
            Prof. Saugata Ghose (CMU, UIUC)
            Prof. James C. Hoe (CMU)
            Prof. Can Alkan (Bilkent University)
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

- **Challenges:**
  - There is no sequencing machine that takes long DNA as an input, and gives the complete sequence as output
  - Sequencing machines extract *small randomized fragments* of the original DNA sequence
Sample Collection

Large DNA molecule

Chopped DNA fragments

Sequencing

Sequenced reads

Preparation

Genome Sequence Analysis
Sequencing Technologies

Short reads: a few hundred base pairs and error rate of ~0.1%
Long reads: thousands to millions of base pairs and error rate of 5–10%

Oxford Nanopore (ONT)
PacBio
Illumina
Current State of Sequencing
Current State of Sequencing (cont’d.)

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

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

Computation is a bottleneck!
Problem Statement

Rapid genome sequence analysis is currently bottlenecked by the computational power and memory bandwidth limitations of existing systems, as many of the steps in genome sequence analysis must process a large amount of data.
Our Goal & Approach

- **Our Goal:**
  Accelerating genome sequence analysis by **efficient** hardware/algorithm co-design

- **Our Approach:**
  1. Analyze the **multiple steps** and the **associated tools** in the genome sequence analysis pipeline,
  2. Expose the **tradeoffs** between accuracy, performance, memory usage and scalability, and
  3. Co-design **fast and efficient algorithms** along with scalable and energy-efficient customized hardware accelerators for the key bottleneck steps of the pipeline
Research Statement

Genome sequence analysis can be accelerated by co-designing fast and efficient algorithms along with scalable and energy-efficient customized hardware accelerators for the key bottleneck steps of the pipeline.
Bottleneck analysis of genome assembly pipeline for long reads

[Briefings in Bioinformatics, 2018]

GenASM: Approximate string matching framework for genome sequence analysis

[MICRO 2020]

BitMAc: FPGA-based near-memory acceleration of bitvector-based sequence alignment

[Will be submitted to Bioinformatics]

GenGraph: Hardware acceleration framework for sequence-to-graph mapping

[Will be submitted to HPCA 2022]
Bottleneck analysis of genome assembly pipeline for long reads

*Briefings in Bioinformatics, 2018*

GenASM: Approximate string matching framework for genome sequence analysis

*MICRO 2020*

BitMAc: FPGA-based near-memory acceleration of bitvector-based sequence alignment

*Will be submitted to Bioinformatics*

GenGraph: Hardware acceleration framework for sequence-to-graph mapping

*Will be submitted to HPCA 2022*
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.
With the emergence of long read sequencing technologies, de novo assembly becomes a promising way of constructing the original genome.

- **Basecalling** (Translates signal data into bases: A,C,G,T)
- **Read-to-Read Overlap Finding** (Finds pairwise read alignments for each pair of read)
- **Assembly** (Traverses the overlap graph & constructs the draft assembly)
- **Read Mapping** (Maps the reads to the draft assembly)
- **Polishing** (Polishes the draft assembly & increases the accuracy)
Our Contributions

- Analyze the tools in multiple dimensions: accuracy, performance, memory usage, and scalability
- Reveal new bottlenecks and trade-offs
- First study on bottleneck analysis of nanopore sequence analysis pipeline on real machines
- Provide guidelines for practitioners
- Provide guidelines for tool developers
Key Findings

- **Laptops** are becoming a popular platform for running genome assembly tools, as the **portability** of a laptop makes it a good fit for **in-field analysis**
  - Greater memory constraints
  - Lower computational power
  - Limited battery life

- **Memory usage** is an important factor that greatly affects the performance and the usability of the tool
  - Data structure choices that increase the memory requirements
  - Algorithms that are not cache-efficient
  - Not keeping memory usage in check with the number of threads

- **Scalability of the tool** with the number of cores is an important requirement. However, parallelizing the tool can **increase the memory usage**
  - Not dividing the input data into batches
  - Not limiting the memory usage of each thread
  - Dividing the dataset instead of the computation between simultaneous threads
Key Findings

Goal 1: High-performance and low-power

- **Memory usage** is an important factor that greatly affects the performance and the usability of the tool
  - Data structure choices that increase the memory requirements
  - Algorithms that are not cache-efficient
  - Not keeping memory usage in check with the number of threads

- **Scalability of the tool** with the number of cores is an important requirement. However, parallelizing the tool can increase the memory usage
  - Not dividing the input data into batches
  - Not limiting the memory usage of each thread
  - Dividing the dataset instead of the computation between simultaneous threads
Key Findings

Goal 1: High-performance and low-power

Goal 2: Memory-efficient

- Scalability of the tool with the number of cores is an important requirement. However, parallelizing the tool can increase the memory usage.
  - Not dividing the input data into batches
  - Not limiting the memory usage of each thread
  - Dividing the dataset instead of the computation between simultaneous threads
Key Findings

Goal 1: High-performance and low-power

Goal 2: Memory-efficient

Goal 3: Scalable/highly-parallel
Bottleneck analysis of genome assembly pipeline for long reads

[Briefings in Bioinformatics, 2018]

GenASM: Approximate string matching framework for genome sequence analysis

[MICRO 2020]

BitMAc: FPGA-based near-memory acceleration of bitvector-based sequence alignment

[Will be submitted to Bioinformatics]

GenGraph: Hardware acceleration framework for sequence-to-graph mapping

[Will be submitted to HPCA 2022]
**Recall: 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.

References:

- Genome Sequence Analysis

Images:

- DNA sequences: ACGTACCCCGT, ACGAGCGGGGT, GATACACTGTG, AAAAAAANAAA, TTTTTTTAATT, ACGACGCTAGCT, CTAGGGACCTT

Additional information:

- **ACGT** A DNA nucleotide sequence
- **AAAA** Another DNA nucleotide sequence
- **TTTTTT** Yet another DNA nucleotide sequence

**SAFARI**

(Visual and textual elements not transcribed here for simplicity.)
Read Mapping Pipeline

- **Indexing** (Pre-processing step to generate index of reference)
  - Hash-table based index

- **Seeding** (Query the index)
  - Potential mapping locations

- **Pre-Alignment Filtering** (Filter out dissimilar sequences)
  - Remaining potential mapping locations

- **Read Alignment** (Perform distance/score calculation & traceback)
  - Optimal alignment
GSA with Read Mapping

- **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

- Bottlenecksed 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
Approximate String Matching

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

  Reference: AAAATGTTTAGTGCTACTTG
  Read: AAAATGTTTAGTGCTACTTG

  deletion  substitution  insertion

- 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 differences)
    - Find the optimal alignment with a traceback step
      - Sequence of matches, substitutions, insertions and deletions, along with their positions
  - Usually implemented as a dynamic programming (DP) based algorithm
DP-based ASM

Commonly-used algorithm for ASM in genomics...

...with quadratic time and space complexity
Bitap Algorithm

- **Bitap**\(^1,2\) 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 \(\text{(A, C, G, T)}\)
  - 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

---

Limitations of Bitap

1) Data Dependency Between Iterations:
   - Two-level data dependency forces the consecutive iterations to take place sequentially
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 = oldR[d-1]
- substitution = oldR[d-1] \ll 1
- insertion = R[d-1] \ll 1
- match = (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.
Step 2: Edit Distance Calculation

For each character of the text (char):

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

- 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 \( = (\text{oldR}[d] \ll 1) | \text{PM}[\text{char}] \)

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

- Check MSB of \( R[d] \):
  
  - If 1, no match.
  
  - If 0, match with \( d \) many errors.

Data dependency between iterations (i.e., no parallelization)
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
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 \( d = 1 \ldots k \):
  
  \[
  \begin{align*}
  \text{deletion} & = \text{oldR}[d-1] \\
  \text{substitution} & = \text{oldR}[d-1] \ll 1 \\
  \text{insertion} & = \text{R}[d-1] \ll 1 \\
  \text{match} & = (\text{oldR}[d] \ll 1) \mid \text{PM} \text{[char]}
  \end{align*}
  \]
  
  \[ R[d] = \text{deletion} \& \text{mismatch} \& \text{insertion} \& \text{match} \]
  
  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:
   o Two-level data dependency forces the consecutive iterations to take place sequentially

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

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

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

5) Limited Memory Bandwidth:
   o High memory bandwidth required to read and write the computed bitvectors to memory
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 Hardware Design**

**GenASM-DC:**
- Generates bitvectors
- Performs edit distance calculation

**GenASM-TB:**
- Performs TraceBack
- Assembles the optimal alignment

**Diagram:**
- Main Memory
- Host CPU
- DC-SRAM
- GenASM-DC Accelerator
- GenASM-DC
- TB-SRAM\(_1\)
- TB-SRAM\(_2\)
- TB-SRAM\(_n\)
- GenASM-TB
- GenASM-TB Accelerator
GenASM Hardware Design

1. Reference and query locations
2. Reference text and query pattern
3. Generate bitvectors
4. Generate bitvectors and perform edit distance calculation
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)]
GenASMs-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
Use Cases of GenASM

(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

We also discuss other possible use cases of GenASM in our paper:
   o Read-to-read overlap finding, hash-table based indexing, whole genome alignment, generic text search
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
Evaluation Methodology (cont’d.)

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
Based on our synthesis of GenASM-DC and GenASM-TB accelerator datapaths using the Synopsys Design Compiler with a 28nm process:

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

### Key Results – Area and Power

<table>
<thead>
<tr>
<th></th>
<th>GenASM-DC (64 PEs)</th>
<th>GenASM-TB</th>
<th>DC-SRAM (8 KB)</th>
<th>TB-SRAMs (64 x 1.5 KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Area (mm²)</strong></td>
<td>0.049</td>
<td>0.016</td>
<td>0.013</td>
<td>0.055</td>
</tr>
<tr>
<td>Total (1 vault):</td>
<td>0.256</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Power (W)</strong></td>
<td>0.033</td>
<td>0.004</td>
<td>0.009</td>
<td>0.055</td>
</tr>
<tr>
<td>Total (32 vaults):</td>
<td>10.69 mm²</td>
<td></td>
<td></td>
<td>3.23 W</td>
</tr>
<tr>
<td>% of a Xeon CPU core:</td>
<td>1%</td>
<td></td>
<td></td>
<td>1%</td>
</tr>
</tbody>
</table>
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

GenASM has low area and power overheads
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 $648\times$ and $116\times$ speedup over 12-thread runs of BWA-MEM and Minimap2, while reducing power consumption by $34\times$ and $37\times$
GenASM provides 3.9x better throughput, 6.6x the throughput per unit area, and 10.5x 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

---

**HW**

GenASM is **more efficient in terms of both speed and power consumption**, while **significantly improving the accuracy of pre-alignment filtering**
Key Results – Use Case 3

(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

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.

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

**SW**

**HW**
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
Summary of GenASM

- **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*, and *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
Bottleneck analysis of genome assembly pipeline for long reads

[Briefings in Bioinformatics, 2018]

GenASM: Approximate string matching framework for genome sequence analysis

[MICRO 2020]

BitMAc: FPGA-based near-memory acceleration of bitvector-based sequence alignment

[Will be submitted to Bioinformatics]

GenGraph: Hardware acceleration framework for sequence-to-graph mapping

[Will be submitted to HPCA 2022]
BitMAc: FPGA-based GenASM

Our Goal:
Map GenASM accelerators to an FPGA with HBM2, where HBM2 offers high memory bandwidth and FPGA resources offer high parallelism by instantiating multiple copies of the GenASM accelerators.

- Re-modified GenASM algorithms for a better mapping to the FPGA resources.
- Intra-level parallelism by instantiating multiple processing elements (PEs) for the DC execution.
- Inter-level parallelism by running multiple independent GenASM executions in parallel.
Key Findings

- Based on the FPGA resources, the complete BitMAc design:
  - Each BitMAc accelerator contains a DC accelerator with 16 PEs, a TB accelerator, an FSM, and 13.2KB of M20Ks
  - 4 BitMAc accelerators connected to each pseudo-channel (128 in total)
  - Clocked at 200MHz

- BitMAc provides:
  - 136× – 761× speedup over the state-of-the-art CPU baselines
  - 6.8× – 19.4× speedup over the state-of-the-art GPU baseline
Key Findings (cont’d.)

- BitMAc has:
  - 64% logic utilization and 90% on-chip memory utilization
  - Total power consumption of 48.9W, where 59% accounts for the M20Ks

- Bottlenecked by the amount of on-chip memory (i.e., M20Ks)

- Cannot saturate the high bandwidth that multiple HBM2 stacks on the FPGA provide

- Need (1) algorithm-level modifications to decrease the amount of data that need to be stored in M20Ks, and (2) newer FPGA chips that provide a higher amount of on-chip memory capacity
Thesis Contributions

Bottleneck analysis of genome assembly pipeline for long reads
[Briefings in Bioinformatics, 2018]

GenASM: Approximate string matching framework for genome sequence analysis
[MICRO 2020]

BitMAc: FPGA-based near-memory acceleration of bitvector-based sequence alignment
[Will be submitted to Bioinformatics]

GenGraph: Hardware acceleration framework for sequence-to-graph mapping
[Will be submitted to HPCA 2022]
Recall: Read Mapping Pipeline

- **Indexing**
  - (Pre-processing step to generate index of reference)
  - Hash-table based index

- **Seeding**
  - (Query the index)
  - Potential mapping locations

- **Pre-Alignment Filtering**
  - (Filter out dissimilar sequences)
  - Remaining potential mapping locations

- **Read Alignment**
  - (Perform distance/score calculation & traceback)
  - Optimal alignment

- Reference genome
- Reads
- Reference segment
- Query read
- reference bias
Genome Graphs

Genome graphs:

- Include the reference genome together with genetic variations
- Provide a compact representation
- Enable us to move away from aligning with single reference genome (reference bias) and toward using the sequence diversity

Reference #1: ACGTACGT
Genome Graphs

Genome graphs:

- Include the reference genome together with genetic variations
- Provide a compact representation
- Enable us to move away from aligning with single reference genome (reference bias) and toward using the sequence diversity

Reference #1: ACGTACGT
Reference #2: ACGGACGT
Genome Graphs

Genome graphs:

- Include the reference genome together with genetic variations
- Provide a compact representation
- Enable us to move away from aligning with single reference genome (reference bias) and toward using the sequence diversity

Reference #1: ACGTACGT
Reference #2: ACGGACGT
Genome Graphs

Genome graphs:

- Include the reference genome together with genetic variations
- Provide a compact representation
- Enable us to move away from aligning with single reference genome (reference bias) and toward using the sequence diversity

Reference #1: ACGTACGT
Reference #2: ACGGACGT
Reference #3: ACGTTACGT
Genome Graphs

Genome graphs:
- Include the reference genome together with genetic variations
- Provide a compact representation
- Enable us to move away from aligning with single reference genome (reference bias) and toward using the sequence diversity

Reference #1: ACGTACGT
Reference #2: ACGGACGT
Reference #3: ACGTTACGT
Genome Graphs

Genome graphs:

- Include the **reference genome together with genetic variations**
- Provide a **compact representation**
- Enable us to move away from aligning with single reference genome (reference bias) and toward using the sequence diversity

**Reference #1**: ACGTACGT

**Reference #2**: ACGGACGT

**Reference #3**: ACGTTACGT

**Reference #4**: ACGACGT
Genome Graphs

Genome graphs:
- Include the **reference genome together with genetic variations**
- Provide a **compact representation**
- Enable us to **move away** from aligning with single reference genome (reference bias) and **toward using the sequence diversity**

Reference #1: ACGTACGT
Reference #2: ACGGACGT
Reference #3: ACGTTACGT
Reference #4: ACGACGT
GenGraph: First Graph Mapping Accelerator

Motivation:
- Traditional read mapping causes **reference bias**
- Aligning sequences to graphs is a newer field and practical tools only start to emerge
- **HW acceleration of sequence-to-graph mapping:** important but unexplored research problem

Goal: Design an accelerator framework for sequence-to-graph mapping, which provides high performance and high accuracy

Our Approach:
- **MinSeed:** The *first* minimizer-based seeding hardware
- **BitAlign:** The *first* sequence-to-graph alignment hardware based on modified GenASM algorithms and accelerators
Overview of GenGraph

Main Memory (graph-based reference & index)

3. Frequencies
4. Seed Locations
5. Graph Nodes

1. Read Scratchpad
2. Find Minimizers
3. Minimizer Scratchpad
4. Filter Frequencies
5. Seed Scratchpad
6. Calculate Seed Regions
7. DC-SRAM (Input Scratchpad)
8. Generate Bitvectors
9. Hop Queues
10. TB-SRAMs (Bitvector Scratchpad)
11. Perform Traceback

Hop Queues
query k-mers
minimizers

MinSeed (MS)

GenGraph

Host CPU
query read

GenGraph (graph-based reference & index)
Evaluation Methodology

- We evaluate GenGraph using:
  - Synthesized SystemVerilog models of the MinSeed and BitAlign accelerator datapaths
  - Simulation- and spreadsheet-based performance modeling

- 4 x 24GB HBM2E stacks, each with 8 independent channels
  - 1 MinSeed and 1 BitAlign HW per each channel (32 in total)

- Baseline tools:
  - GraphAligner for long reads and vg for short reads

- Simulated Datasets:
  - PacBio and ONT datasets (10 KBp reads with 5-10% error rate)
  - Illumina datasets (100-250 bp with 1% error rate)
Key Results – Area & Power

- Based on our *synthesis* of **MinSeed** and **BitAlign** accelerator datapaths using the Synopsys Design Compiler with a **28nm** process:
  - Both operates @ **1GHz**

<table>
<thead>
<tr>
<th>Component</th>
<th>Area (mm²)</th>
<th>Power (W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinSeed – Logic</td>
<td>0.013</td>
<td>0.008</td>
</tr>
<tr>
<td>BitAlign – DC Logic (64 PEs)</td>
<td>0.383</td>
<td>0.378</td>
</tr>
<tr>
<td>BitAlign – TB Logic</td>
<td>0.020</td>
<td>0.003</td>
</tr>
<tr>
<td><strong>Total – 1 x GenGraph – Logic</strong></td>
<td>0.416</td>
<td>0.389</td>
</tr>
<tr>
<td><strong>Total – 8 x GenGraph – Logic</strong></td>
<td>3.328</td>
<td>3.112</td>
</tr>
<tr>
<td><strong>Total – 32 x GenGraph – Logic</strong></td>
<td>13.312</td>
<td>12.448</td>
</tr>
</tbody>
</table>
GenGraph provides 3× throughput improvement over GraphAligner’s 12-thread execution, while reducing the power consumption by 6.7×
Key Results – Short Read Analysis

GenGraph provides 257× throughput improvement over vg’s 12-thread execution, while reducing the power consumption by 6.7×
Summary of GenGraph

- **Problem:**
  - Traditional read mapping causes reference bias
  - Aligning sequences to graphs is a newer field and practical tools only start to emerge
  - HW acceleration of sequence-to-graph mapping: important but unexplored research problem

- **Key Contributions:**
  - **GenGraph:** First acceleration framework for sequence-to-graph mapping
    - *First* minimizer-based seeding accelerator
    - *First* sequence-to-graph alignment accelerator based upon our new bitvector-based, highly-parallel algorithm
    - Evaluation for both short and long reads

- **Key Results:** GenGraph provides significant speedup compared to the baselines, while reducing the power consumption
# Thesis Contributions

<table>
<thead>
<tr>
<th>Contribution</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bottleneck analysis of genome assembly pipeline for long reads</td>
<td><em>Briefings in Bioinformatics, 2018</em></td>
</tr>
<tr>
<td>GenASM: Approximate string matching framework for genome sequence analysis</td>
<td><em>MICRO 2020</em></td>
</tr>
<tr>
<td>BitMAc: FPGA-based near-memory acceleration of bitvector-based sequence alignment</td>
<td><em>Will be submitted to Bioinformatics</em></td>
</tr>
<tr>
<td>GenGraph: Hardware acceleration framework for sequence-to-graph mapping</td>
<td><em>Will be submitted to HPCA 2022</em></td>
</tr>
</tbody>
</table>
Rapid genome sequence analysis is bottlenecked by the computational power and memory bandwidth limitations of existing systems, as many of the steps in genome sequence analysis must process a large amount of data.
Conclusion (cont’d.)

Genome sequence analysis can be accelerated by co-designing fast and efficient algorithms along with scalable and energy-efficient customized hardware accelerators for the key bottleneck steps of the pipeline.
Conclusion (cont’d.)

Bottleneck analysis of genome assembly pipeline for long reads

[Briefings in Bioinformatics, 2018]

GenASM: Approximate string matching framework for genome sequence analysis

[MICRO 2020]

BitMAc: FPGA-based near-memory acceleration of bitvector-based sequence alignment

[Will be submitted to Bioinformatics]

GenGraph: Hardware acceleration framework for sequence-to-graph mapping

[Will be submitted to HPCA 2022]
Other Publications @ SAFARI

**FPGA-based Near-Memory Acceleration of Modern Data-Intensive Applications (IEEE Micro, 2021)**
Gagandeep Singh, Mohammed Alser, Damla Senol Cali, Dionysios Diamantopoulos, Juan Gomez-Luna, Henk Corporaal, and Onur Mutlu.

**Accelerating Genome Analysis: A Primer on an Ongoing Journey (IEEE Micro, 2020)**
Mohammed Alser, Zulal Bingol, Damla Senol Cali, Jeremie S. Kim, Saugata Ghose, Can Alkan, and Onur Mutlu.

**Apollo: A Sequencing-Technology-Independent, Scalable, and Accurate Assembly Polishing Algorithm (Bioinformatics, 2020)**
Can Firtina, Jeremie S. Kim, Mohammed Alser, Damla Senol Cali, A. Ercument Cicek, Can Alkan, and Onur Mutlu.

**Demystifying Workload–DRAM Interactions: An Experimental Study (ACM SIGMETRICS, 2019)**
Saugata Ghose, Tianshi Li, Nastaran Hajinazar, Damla Senol Cali, and Onur Mutlu.

**GRIM-Filter: Fast Seed Location Filtering in DNA Read Mapping Using Processing-in-Memory Technologies (BMC Genomics, 2018)**
Jeremie S. Kim, Damla Senol Cali, Hongyi Xin, Donghyuk Lee, Saugata Ghose, Mohammed Alser, Hasan Hassan, Oguz Ergin, Can Alkan, and Onur Mutlu.
Acknowledgments

- Onur Mutlu and Saugata Ghose
- Can Alkan
- James C. Hoe
- Lavanya Subramanian and Rachata Ausavarungnirun
- Sree Subramoney and Gurpreet S. Kalsi
- Jeremie, Can, Zulal, Nastaran, Gagan, Giray, Mohammed, Nour, Amirali, Nika, Geraldo, Juan, Banu, Minh, Joel, Ziyi, and all other SAFARI, ARCANA, and Bilkent CompGen members
- My family and friends
- My parents, Mine and Sinan
- My sister, Irmak
- My husband, Tunca
Accelerating Genome Sequence Analysis via Efficient Hardware/Algorithm Co-Design

Damla Senol Cali
Ph.D. Thesis Defense - July 15, 2021
dsenol@andrew.cmu.edu

Committee: Prof. Onur Mutlu (CMU, ETH Zurich)
Prof. Saugata Ghose (CMU, UIUC)
Prof. James C. Hoe (CMU)
Prof. Can Alkan (Bilkent University)