HiPEAC 2024

Paper

ApHMM

Accelerating Profile Hidden Markov Models for Fast and Energy-Efficient Genome Analysis

Can Firtina

canfirtina@gmail.com

https://cfirtina.com

Kamlesh Pillai, Gurpreet S. Kalsi, Bharathwaj Suresh, Damla Senol Cali, Jeremie S. Kim, Taha Shahroodi, Meryem Banu Cavlak, Joël Lindegger, Mohammed Alser, Juan Gómez Luna, Sreenivas Subramoney, Onur Mutlu

SAFARI ETH zürich

intel TU Delft Carnegie Mellon

Executive Summary

Motivation: Graph structures such as **profile Hidden Markov Models (pHMMs)** are commonly used to accurately analyze biological sequences

Problem: The parameters used in pHMMs are mainly trained and used with a **computationally intensive Baum-Welch algorithm**, causing major performance and energy overhead for many genomics workloads

Goal: Enable rapid, power-efficient, and flexible use of pHMMs for genomics workloads

ApHMM: the first flexible and hardware-software accelerator for pHMMs that can

1) Substantially reduce unnecessary data storage, data movement, and computations by effectively co-designing hardware and software together

2) Provide a flexible design to support several genomics workloads that use pHMMs

Key Results: Our ASIC implementation compared to CPU, GPU, and FPGA baselines across 3 workloads

- **15.55×–260.03×, 1.83×–5.34×, and 27.97× better performance**
- **Up to 2622.94×** reduction in **energy consumption**

https://github.com/CMU-SAFARI/ApHMM-GPU

Outline

Background & Problem

ApHMM

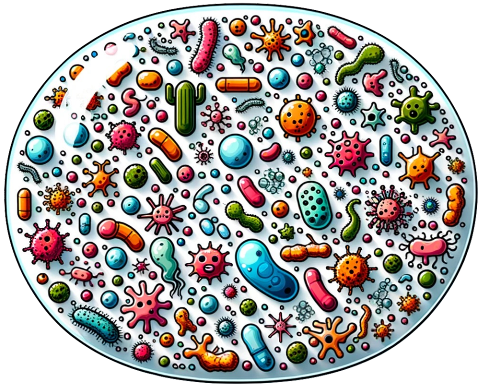
Evaluation

Conclusion

Genome Analysis – Why?

- **Fast and accurate** genome analysis is important for:

Understanding **genetic variations, species, and evolution**



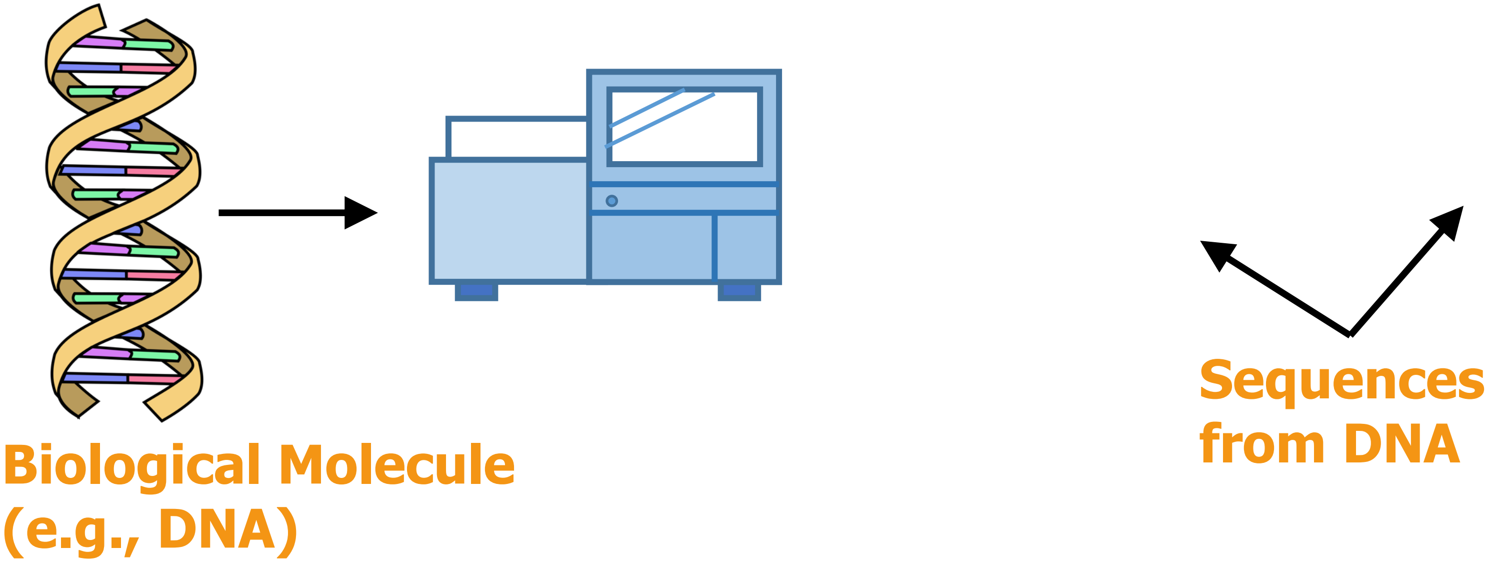
Predicting the **presence of pathogens** in an environment

Surveillance of **disease outbreaks**

Personalized medicine

Background: Genome Analysis – How?

- **Genome sequencing machines** can quickly convert biological molecules
  - Into sequences of characters for analysis



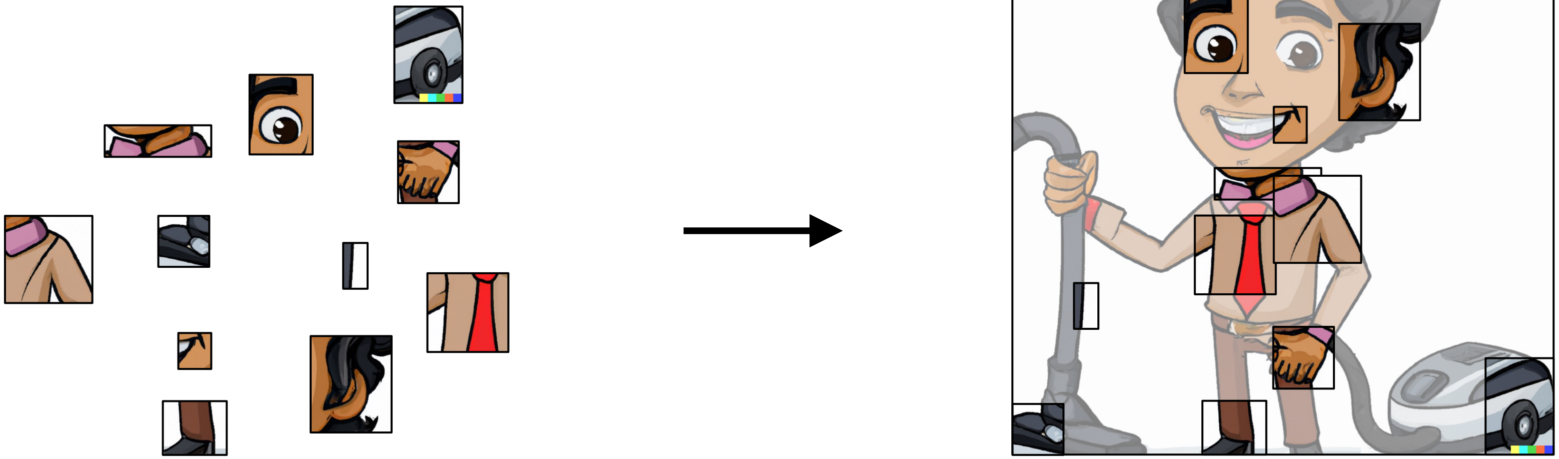
Biological Molecule (e.g., DNA)

Sequences from DNA

Sequence Comparison is Essential

- Analyze sequences by accurately and quickly **comparing** them
  - To **each other**
  - To a **template sequence** representative of a species, a certain group…

Biological Sequences (e.g., DNA, proteins)

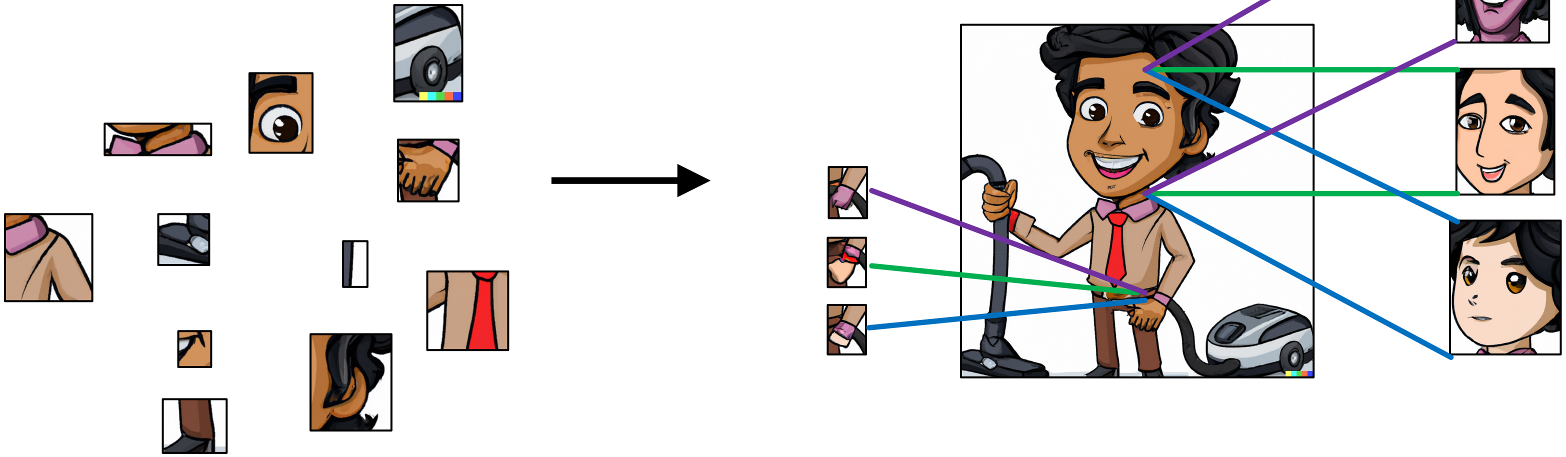


- Essential to understand functionality of a sequence, mutations, diseases…

Graphs for Sequence Comparisons

- **Graphs are commonly used** in sequence comparisons
  - **Can avoid redundant** comparisons and storage
  - Provides **rich information** on **expected variations** between sequences

Biological Sequences (e.g., DNA, proteins)

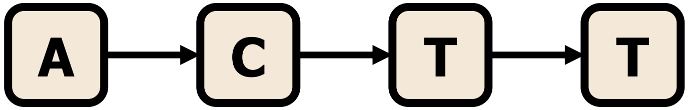


Profile Hidden Markov Models

- Profile Hidden Markov Models (pHMMs) are powerful and common **graph structures** for sequence comparison
  - **Goal:** Identify variations between sequences **probabilistically**
  - Each **state** outputs a biological character (**emission**) when visited
  - States are visited via **transitions** (edges) based on **observed variations**
  - **Variations:** No variation

Expected sequence: ACTT

Observed Sequence #1: ACTT
(No variation)

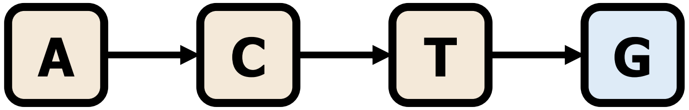


Profile Hidden Markov Models

- Profile Hidden Markov Models (pHMMs) are powerful and common **graph structures** for sequence comparison
  - **Goal:** Identify variations between sequences **probabilistically**
  - Each **state** outputs a biological character (**emission**) when visited
  - States are visited via **transitions** (edges) based on **observed variations**
  - **Variations:** No variation, Substitutions

Expected sequence: ACTT

Observed Sequence #2: ACTG (Substitutions)

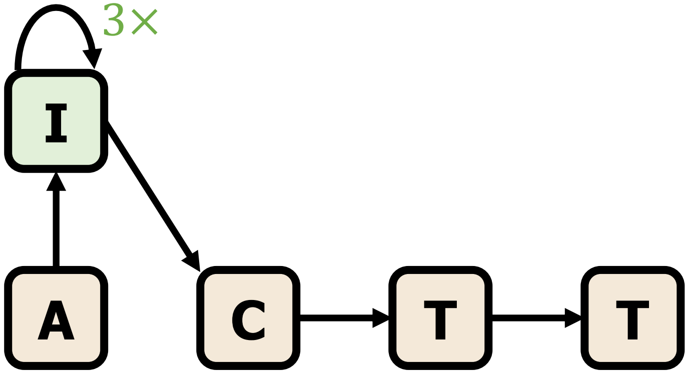


Profile Hidden Markov Models

- Profile Hidden Markov Models (pHMMs) are powerful and common **graph structures** for sequence comparison
  - **Goal:** Identify variations between sequences **probabilistically**
  - Each **state** outputs a biological character (**emission**) when visited
  - States are visited via **transitions** (edges) based on **observed variations**
  - **Variations:** No variation, Substitutions, Insertions

Expected sequence: ACTT

Observed Sequence #3: AGGGCTT
(I: Insertions)

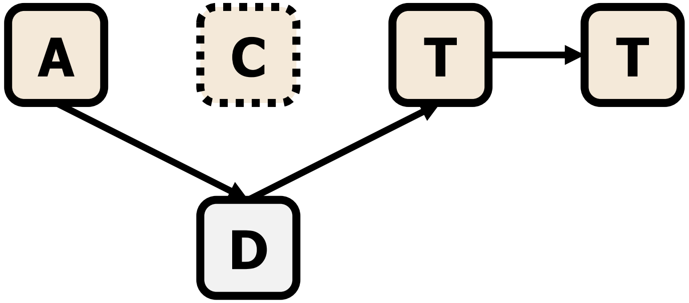


Profile Hidden Markov Models

- Profile Hidden Markov Models (pHMMs) are powerful and common **graph structures** for sequence comparison
  - **Goal:** Identify variations between sequences **probabilistically**
  - Each **state** outputs a biological character (**emission**) when visited
  - States are visited via **transitions** (edges) based on **observed variations**
  - **Variations:** No variation, Substitutions, Insertions, Deletions

Expected sequence: ACTT

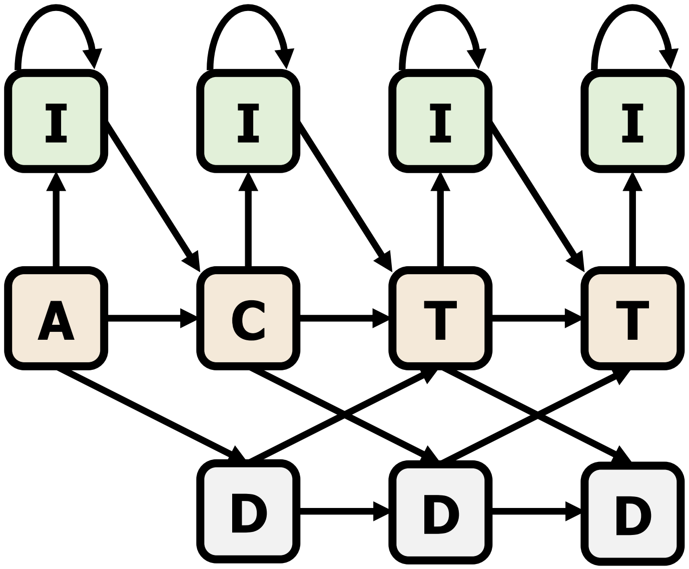
Observed Sequence #4: ATT
(D: Deletions)



Profile Hidden Markov Models

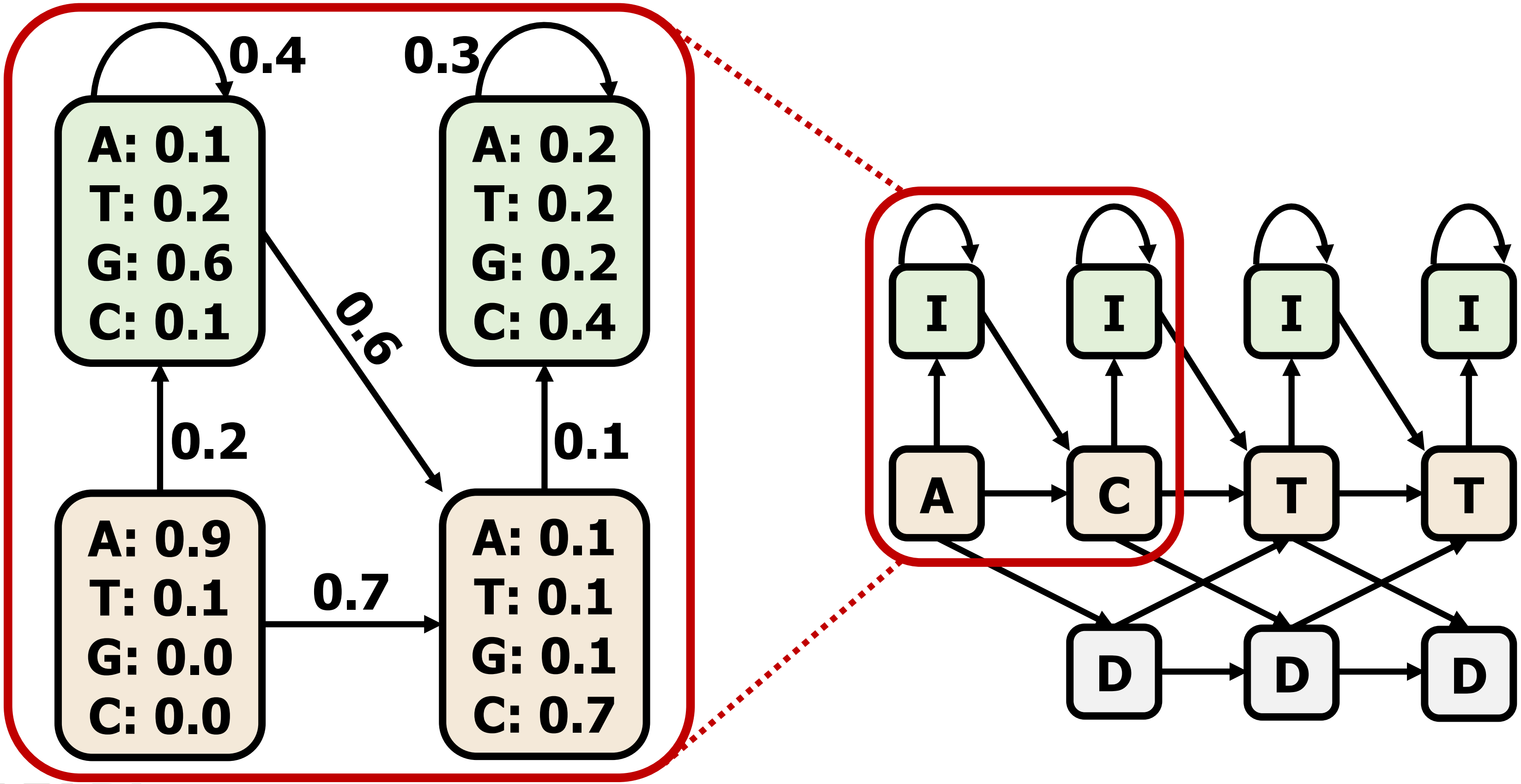
- Profile Hidden Markov Models (pHMMs) are powerful and common **graph structures** for sequence comparison
  - **Goal:** Identify variations between sequences **probabilistically**
  - Each **state** outputs a biological character (**emission**) when visited
  - States are visited via **transitions** (edges) based on **observed variations**
  - **Variations:** No variation, Substitutions, Insertions, Deletions

Observed Sequence #1: ACTT
Observed Sequence #2: ACTG
Observed Sequence #3: AGGGCTT
Observed Sequence #4: ATT
…



Probabilities in pHMMs

- Profile Hidden Markov Models (pHMMs) are powerful and common **graph structures** for sequence comparison
  - **Goal:** Identify variations between sequences **probabilistically**



Utilizing Probabilities in pHMMs

- **The Baum-Welch algorithm** is commonly used with pHMMs
  - For both **inference and training** by effectively utilizing the probabilities

- **Inference:** Identifying the variations between sequences

- **Training:** Maximizing parameters to observe certain variations

Forward Calculations

$$\boxed{F_t(i)} = \sum_{j \in V} \boxed{F_{t-1}(j)} \alpha_{ji} e_{S[t]}(v_i)$$

Backward Calculations

$$\boxed{B_t(i)} = \sum_{j \in V} \boxed{B_{t+1}(j)} \alpha_{ij} e_{S[t+1]}(v_j)$$

Updating Transition Probabilities

$$\alpha^*_{ij} = \frac{\sum\limits_{t=1}^{n_S-1} \alpha_{ij} e_{S[t+1]}(v_j) F_t(i) B_{t+1}(j)}{\sum\limits_{t=1}^{n_S-1} \sum\limits_{x \in V} \alpha_{ix} e_{S[t+1]}(v_x) F_t(i) B_{t+1}(x)}$$

Updating Emission Probabilities

$$e^*_X(v_i) = \frac{\sum\limits_{t=1}^{n_S} F_t(i) B_t(i) [S[t] = X]}{\sum\limits_{t=1}^{n_S} F_t(i) B_t(i)}$$

Utilizing Probabilities in pHMMs

- **The Baum-Welch algorithm** is commonly used with pHMMs
  - For both **inference and training** by effectively utilizing the probabilities

- **Inference:** Identifying the variations between sequences

- **Training:** Maximizing parameters to observe certain variations

Forward Calculations

$$\boxed{F_t(i)} = \sum_{j \in V} F_{t-1}(j)\alpha_{ji} e_{S[t]}(v_i)$$

Backward Calculations

$$\boxed{B_t(i)} = \sum_{j \in V} B_{t+1}(j)\alpha_{ij} e_{S[t+1]}(v_j)$$

Updating Transition Probabilities

$$\alpha^*_{ij} = \frac{\sum\limits_{t=1}^{n_S-1} \alpha_{ij} e_{S[t+1]}(v_j)\boxed{F_t(i)B_{t+1}(j)}}{\sum\limits_{t=1}^{n_S-1} \sum\limits_{x \in V} \alpha_{ix} e_{S[t+1]}(v_x)\boxed{F_t(i)B_{t+1}(x)}}$$

Updating Emission Probabilities

$$e^*_X(v_i) = \frac{\sum\limits_{t=1}^{n_S} \boxed{F_t(i)B_t(i)}[S[t] = X]}{\sum\limits_{t=1}^{n_S} \boxed{F_t(i)B_t(i)}}$$

Utilizing Probabilities in pHMMs

- **The Baum-Welch algorithm** is commonly used with pHMMs
  - For both **inference and training** by effectively utilizing the probabilities

- **Inference:** Identifying the variations between sequences

- **Training:** Maximizing parameters to observe certain variations

Forward Calculations

$$F_t(i) = \sum_{j \in V} F_{t-1}(j)\alpha_{ji} e_{S[t]}(v_i)$$

Backward Calculations

$$B_t(i) = \sum_{j \in V} B_{t+1}(j)\alpha_{ij} e_{S[t+1]}(v_j)$$

Training Step

Updating Transition Probabilities

$$\alpha^*_{ij} = \frac{\sum\limits_{t=1}^{n_S-1} \alpha_{ij} e_{S[t+1]}(v_j) F_t(i) B_{t+1}(j)}{\sum\limits_{t=1}^{n_S-1} \sum\limits_{x \in V} \alpha_{ix} e_{S[t+1]}(v_x) F_t(i) B_{t+1}(x)}$$

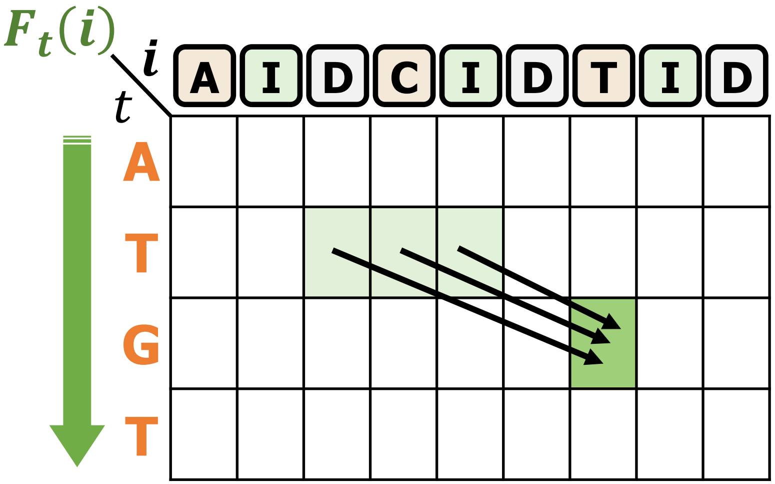
Updating Emission Probabilities

$$e^*_X(v_i) = \frac{\sum\limits_{t=1}^{n_S} F_t(i) B_t(i) [S[t] = X]}{\sum\limits_{t=1}^{n_S} F_t(i) B_t(i)}$$

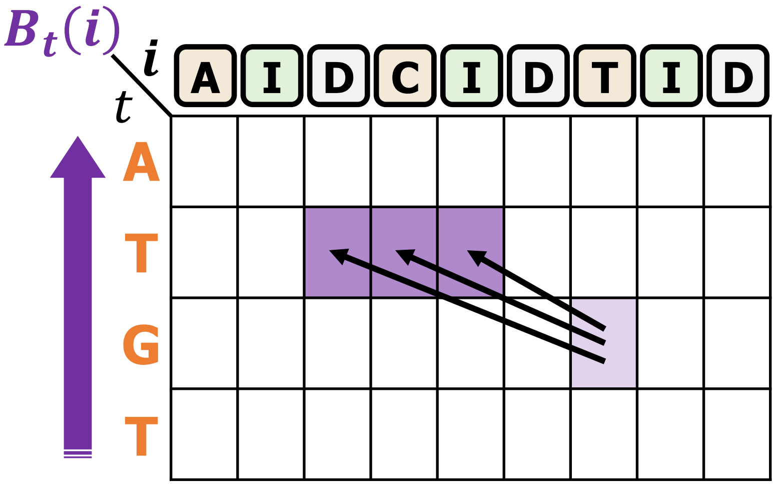
Forward & Backward Calculations

- **A dynamic programming** approach
  - Calculate the ‘possibility’ of visiting each state in a pHMM
  - Given an observed sequence (from both directions of the sequence)

Observed Sequence: ATGT



Forward Calculations

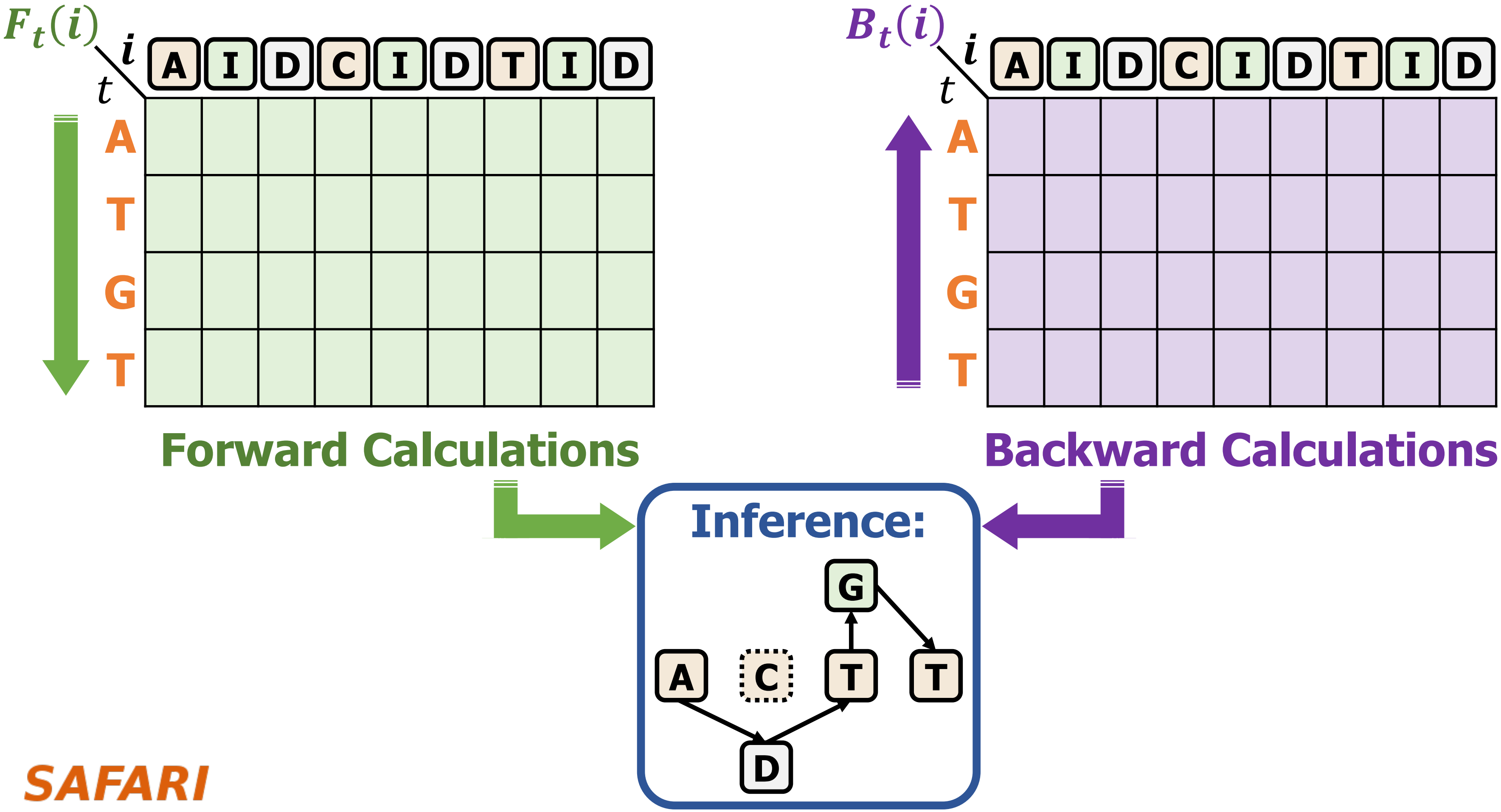


Backward Calculations

Inference using pHMMs

- **Goal:** Identifying the variations between sequences
  - **Inference** by using decoding algorithms (e.g., the Viterbi Algorithm)

Observed Sequence: ATGT



Forward Calculations

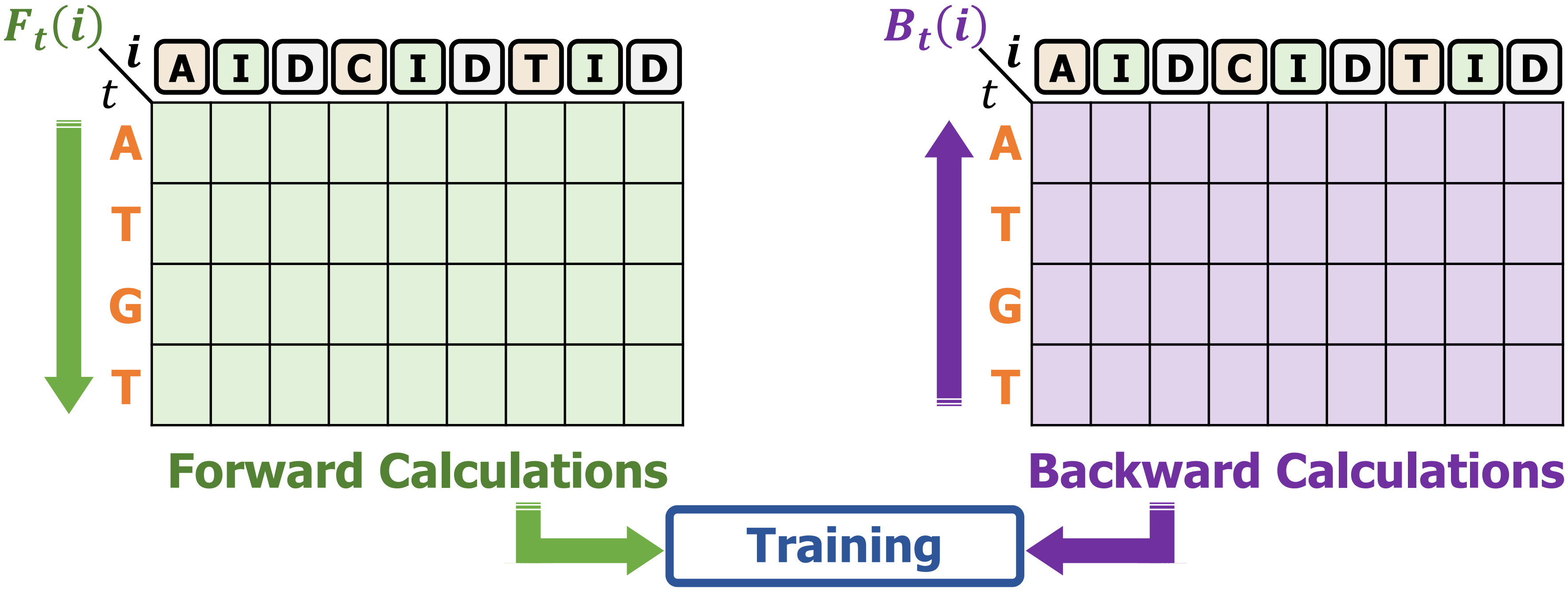
Backward Calculations

Inference:

Training using pHMMs

- **Goal:** Maximizing parameters to observe certain variations
  - **Training** using the parameter updating steps in the Baum-Welch algorithm

Observed Sequence: ATGT



$F_t(i)$

i A I D C I D T I D

t A T G T

Forward Calculations

$B_t(i)$

i A I D C I D T I D

t A T G T

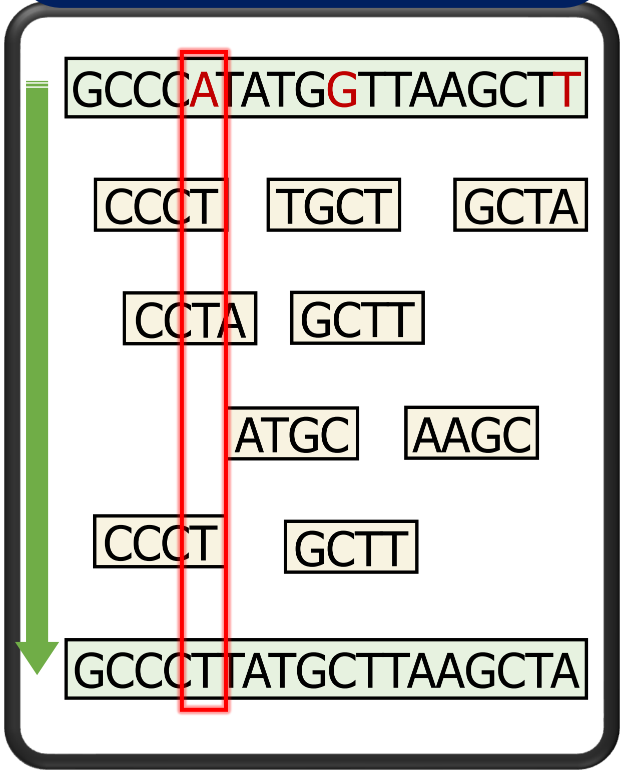
Backward Calculations

Training

pHMMs in Genomics Workloads

- **pHMMs** are commonly used in many genomics applications

1. Error Correction



GCCCATATGGTTAAGCTT

CCCT TGCT GCTA

CCTA GCTT

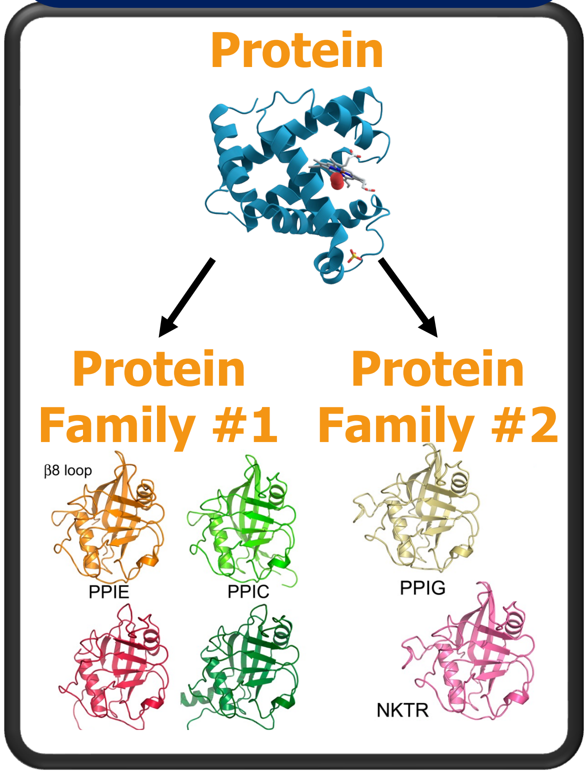
ATGC AAGC

CCCT GCTT

GCCCTTATGCTTAAGCTA

2. Protein Family Search

Protein



Protein Family #1

Protein Family #2

3. Multiple Sequence Alignment

GCCC-TATGGTTAAGCTT

GCCCATATGATTAAGCTT

GCCCATATGGTTAAGCTT

GCCCGTATGGTT---GCTT

GCCCATATGCTTAAGCTT

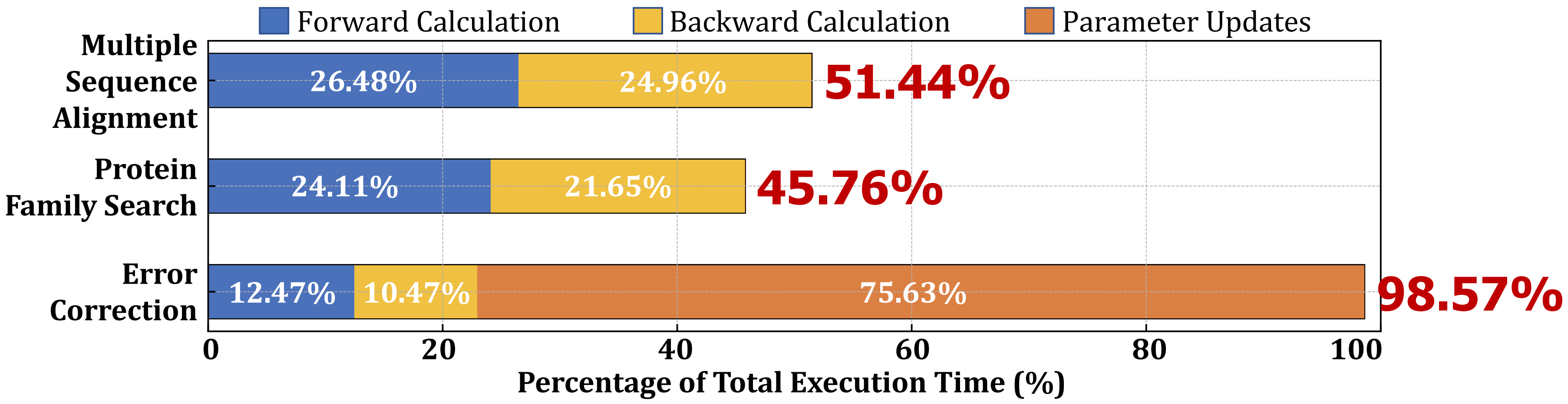
GCCC---TGGTTAAGCT--T

GCCCATATCCTTAAGCTT

GCCCATATGGTTAAGCTT

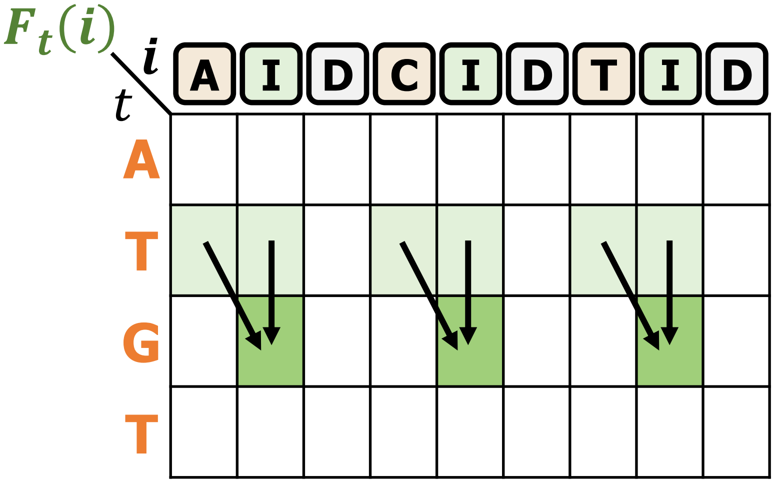
The Baum-Welch Algorithm is Costly

- The Baum-Welch algorithm causes a **major computational overhead** in genomics workloads
  - Taking up from **46% to 99% of the overall execution time**
  - Computationally complex dynamic programming calculations
  - Compute intensive many floating-point operations

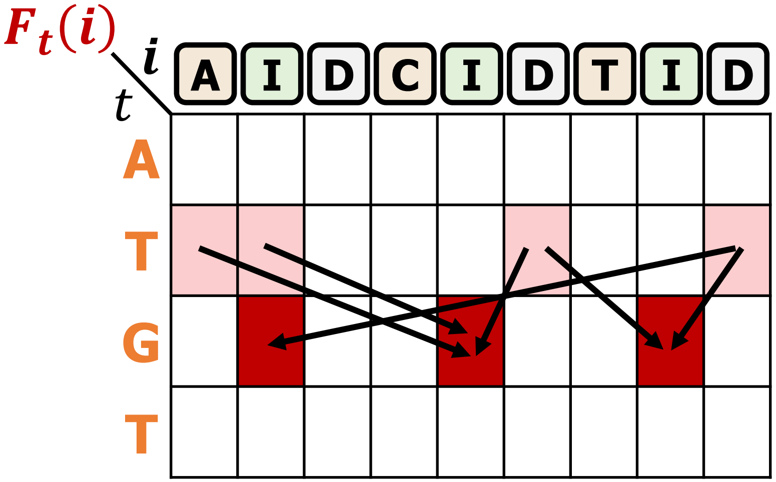


Existing Solutions are Ineffective

- pHMMs are specialized version of **Hidden Markov Models (HMMs)** with **fixed patterns** on states and transitions



Forward Calculations in pHMMs

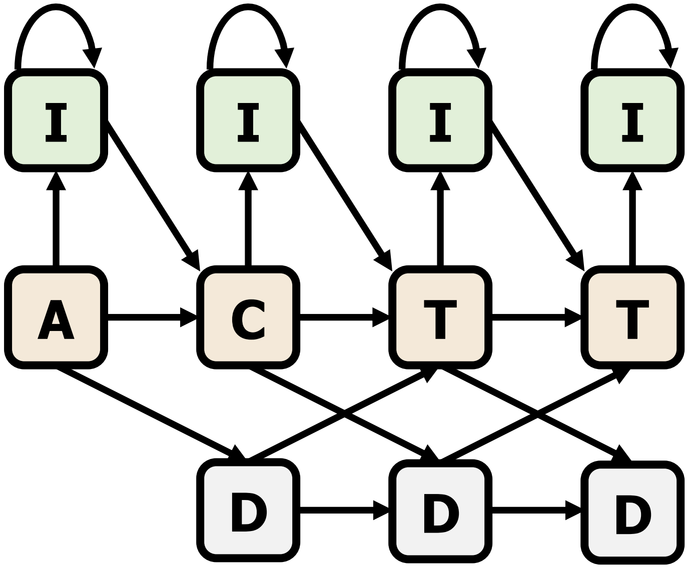


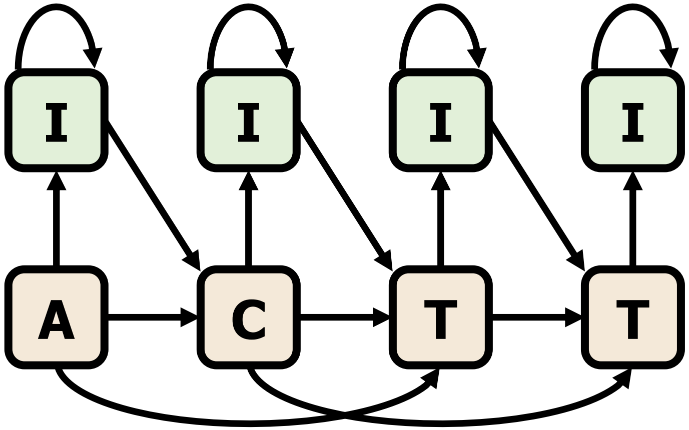
Forward Calculations in HMMs

Generic HMM accelerators **cannot exploit the fixed data dependency pattern** of pHMMs

Existing Solutions are Inflexible

- pHMM requirements can change based on the application
  - **Different pHMM designs:**





  - **Different alphabet sizes**: DNA (4 letters), protein (20 letters)

Lack of **flexible mechanisms**
to handle different design choices

SAFARI

Existing Solutions are Inefficient

- Suboptimal vectorization of SIMD-based solutions on CPUs and GPUs
  - High warp divergence, branching, low port utilization…

- A significant portion of the floating-point operations in dynamic programming is redundant
  - Same multiplications results can redundantly be computed during training
  - Unnecessary data movements

Existing solutions provide suboptimal solutions due to
inefficient hardware or software design

The Problem

The Baum-Welch algorithm causes major performance overhead in important genomics applications

- Same multiplications appear repeatedly due to constant values during

Hardware- or software-only solutions are not sufficient for effectively accelerating pHMMs

Outline

Background & Problem

ApHMM

Evaluation

Conclusion

Goal

Enable **rapid, power-efficient, and flexible** use of pHMMs when using the Baum-Welch algorithm

ApHMM

The first flexible hardware-software co-designed acceleration framework that can significantly reduce the computational overhead of the Baum-Welch algorithm for pHMMs

ApHMM-GPU: The first GPU implementation of the Baum-Welch algorithm for pHMMs

Key Software & Hardware Optimizations

- **Minimize redundant data storage** by efficient pipelining
- **Reduce unnecessary computations** with quick filtering
- **Avoid repeated operations** by utilizing lookup tables

SW

- **Reduce data movement** by exploiting fixed data pattern
- **Flexible and efficient** control logic and hardware design

HW

Key Software & Hardware Optimizations

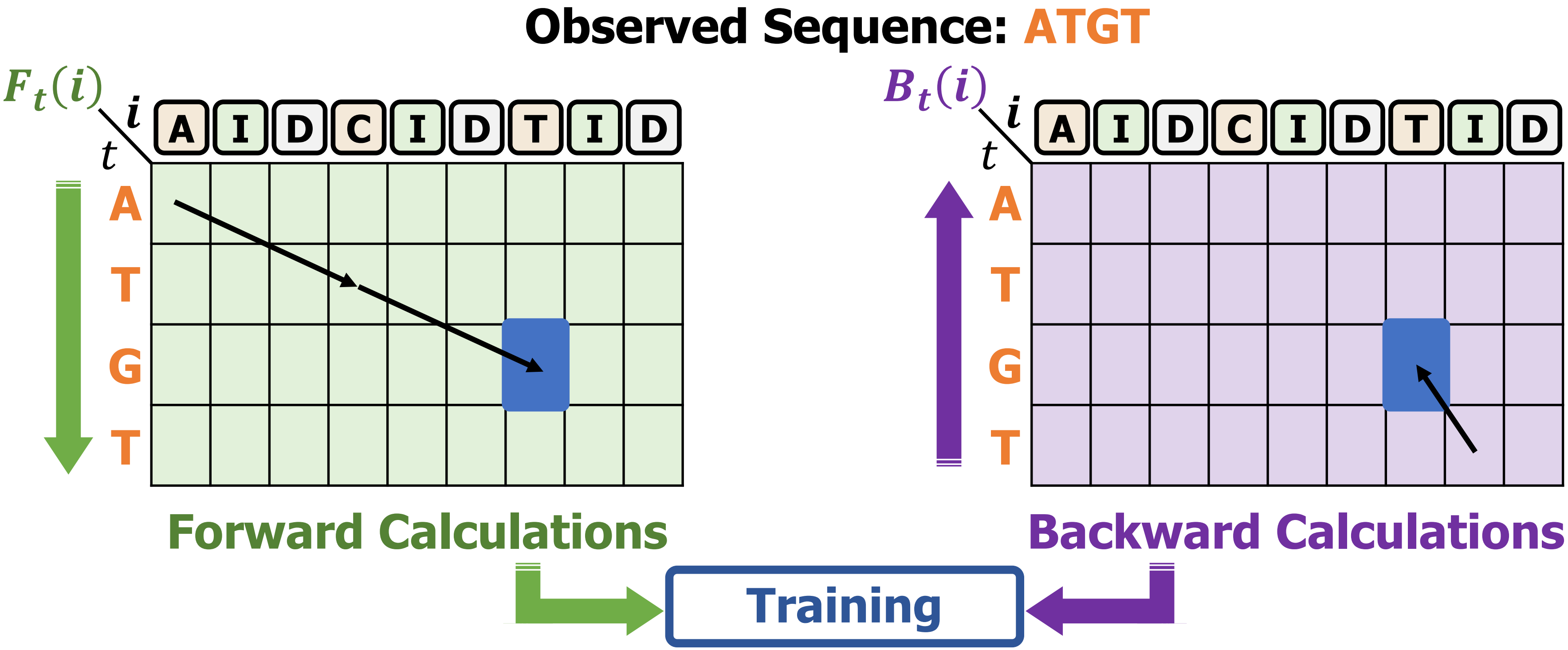
- **Minimize redundant data storage** by efficient pipelining
- **Reduce unnecessary computations** with quick filtering

SW

- **Avoid repeated operations** by utilizing lookup tables

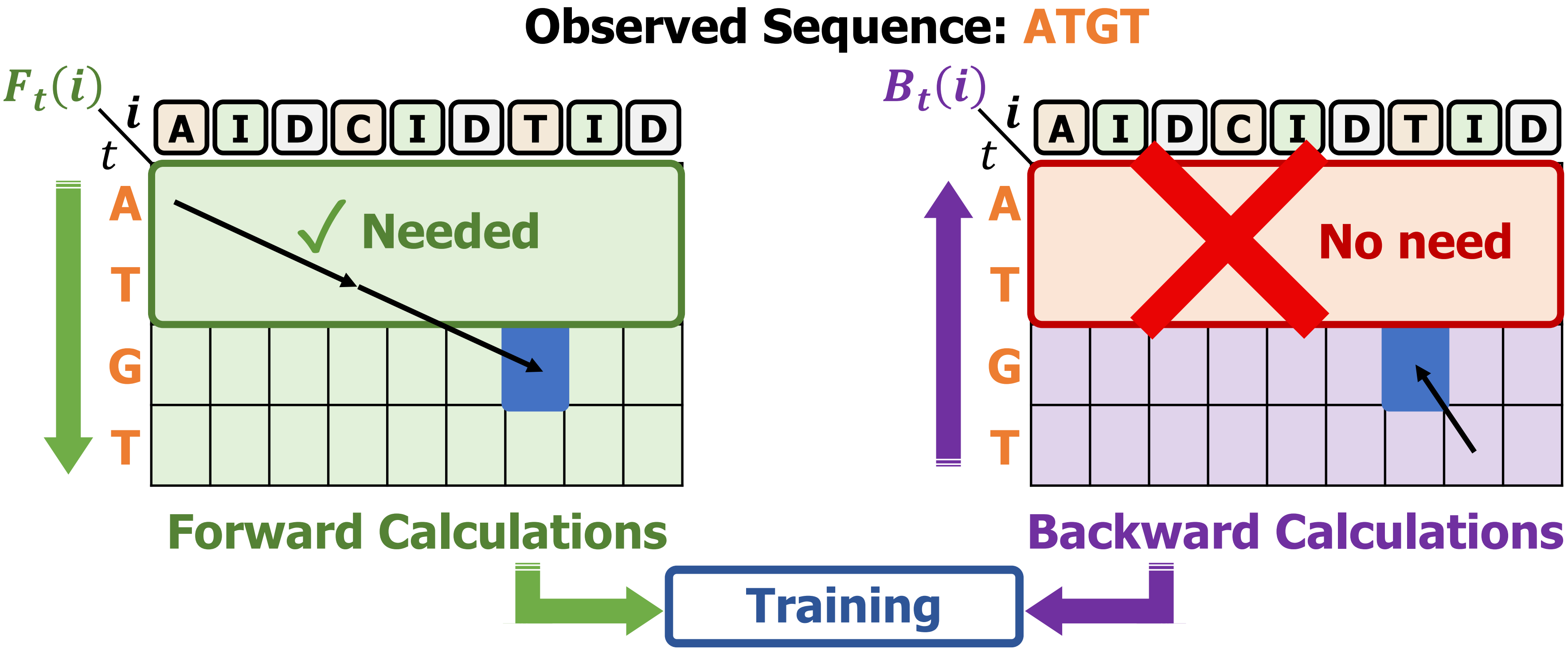
SW: Minimizing Redundant Storage

- **Observation:** Filling the entire Backward table is unnecessary
  - **Pipelining opportunities** to directly consume a Backward value



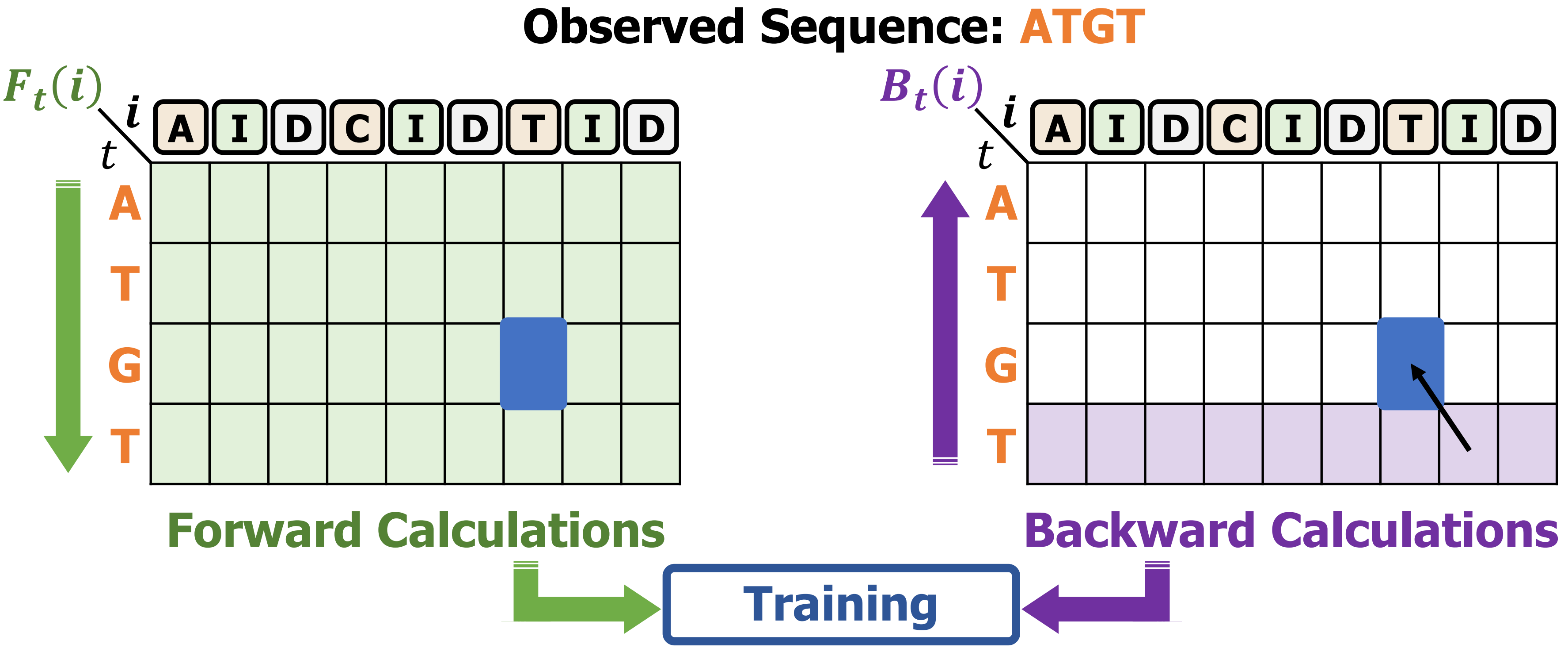
SW: Minimizing Redundant Storage

- **Observation:** Filling the entire Backward table is unnecessary
  - **Pipelining opportunities** to directly consume a Backward value



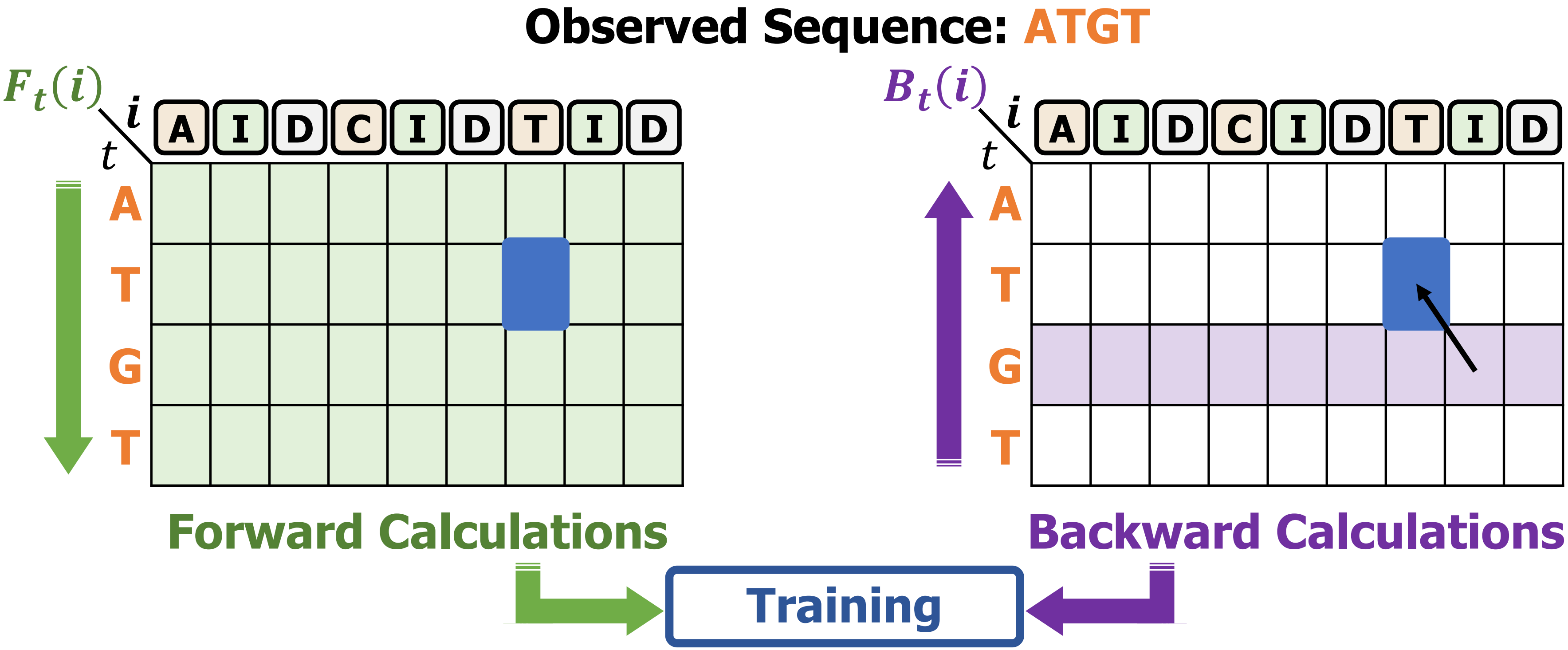
SW: Minimizing Redundant Storage

- **Observation:** Filling the entire Backward table is unnecessary
  - **Pipelining opportunities** to directly consume a Backward value
  - **Partial compute approach:** Only a single row should be **fully stored**



SW: Minimizing Redundant Storage

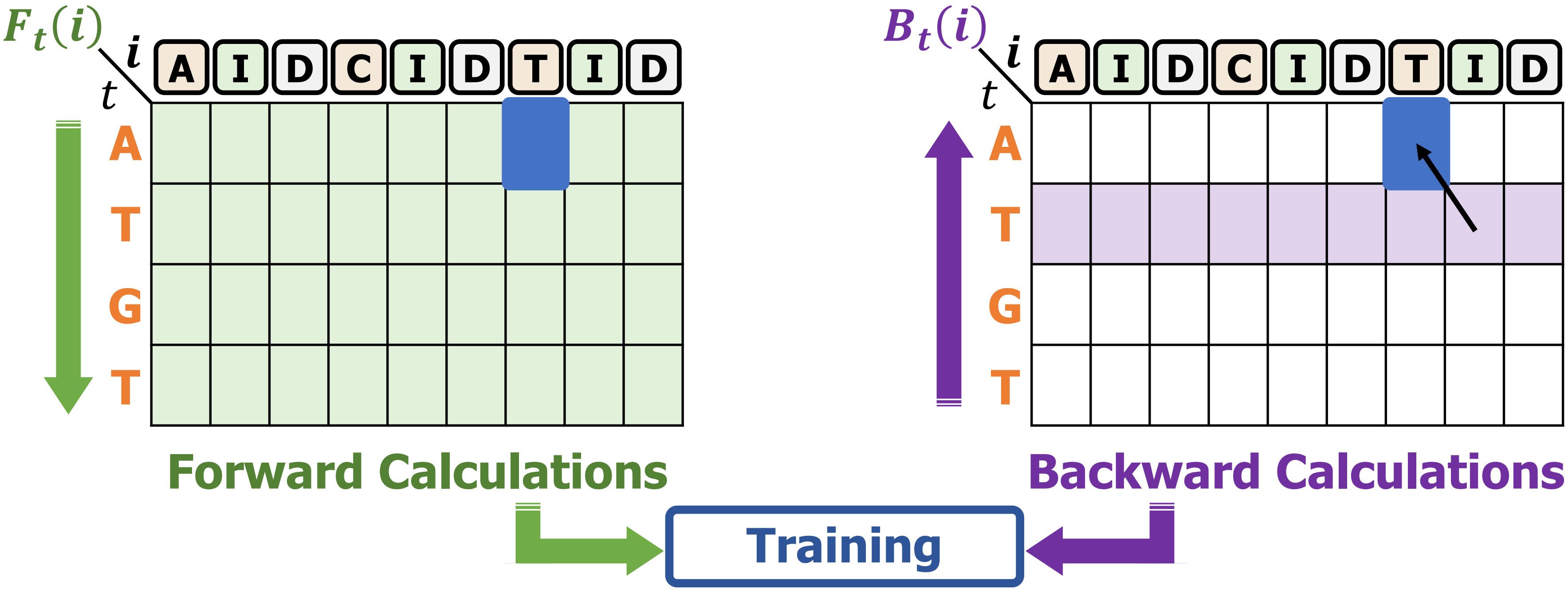
- **Observation:** Filling the entire Backward table is unnecessary
  - **Pipelining opportunities** to directly consume a Backward value
  - **Partial compute approach:** Only a single row should be **fully stored**



SW: Minimizing Redundant Storage

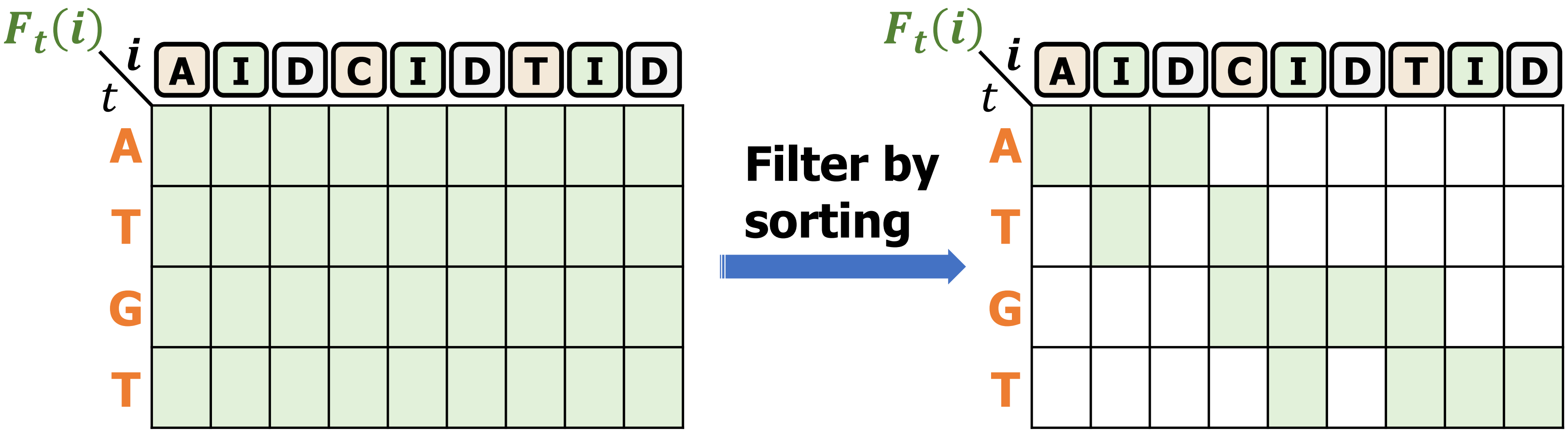
- **Observation:** Filling the entire Backward table is unnecessary
  - **Pipelining opportunities** to directly consume a Backward value
  - **Partial compute approach:** Only a single row should be **fully stored**
  - **Reduces the storage requirements** during training

Observed Sequence: ATGT



SW: Reducing Unnecessary Computations

- **Observation:** 'Negligible' cells can be ignored without significantly reducing overall accuracy
  - **Filtering:** Non-negligible states are identified by sorting
  - **Sorting** to find **exactly** n states with **largest** Forward or Backward values



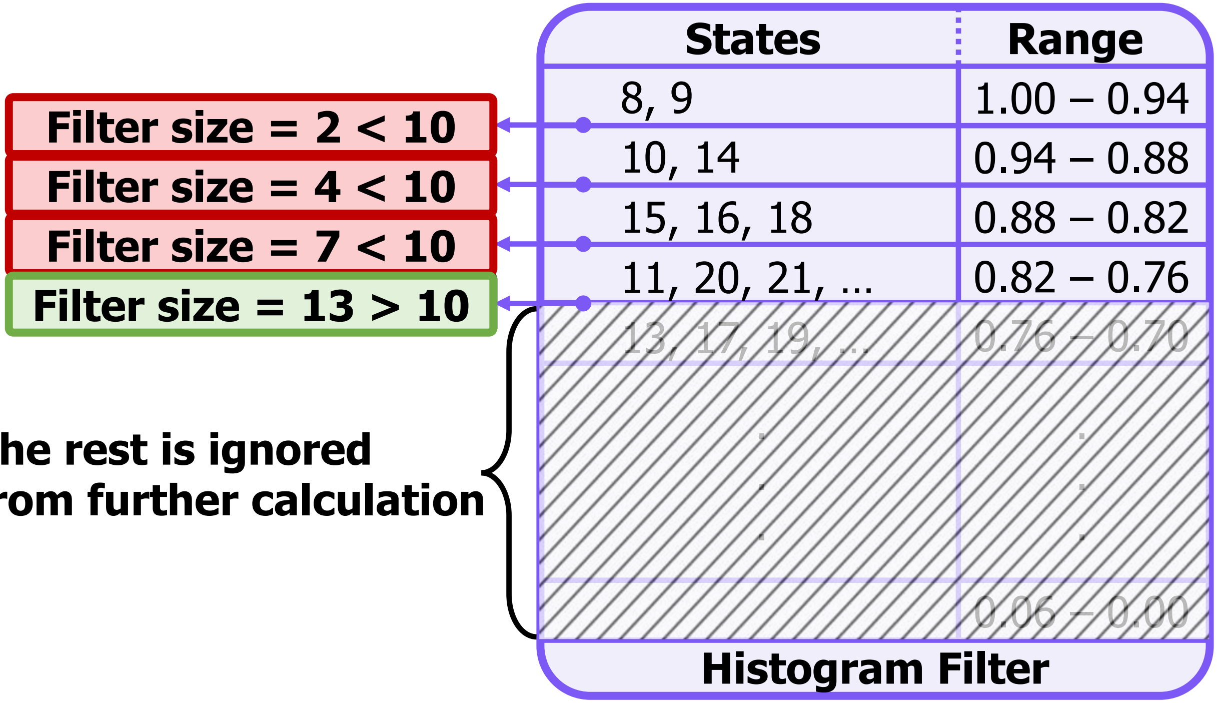
Forward Calculations

Filtered Forward Calculations

- **Sorting is complex** to implement in hardware (and costly)
  - Can we filter without sorting?

SW: Reducing Unnecessary Computations

- **Observation:** ‘Negligible’ cells can be ignored without significantly reducing overall accuracy
  - **Goal:** Find **at least** n states with largest Forward and Backward values
  - **Histogram-based filtering:** Placing the states into buckets corresponding to a range of values
  - Filter is full as soon we find **at least** n **states (e.g.,** $n = 10$**)**



Filter size = 2 < 10

Filter size = 4 < 10

Filter size = 7 < 10

Filter size = 13 > 10

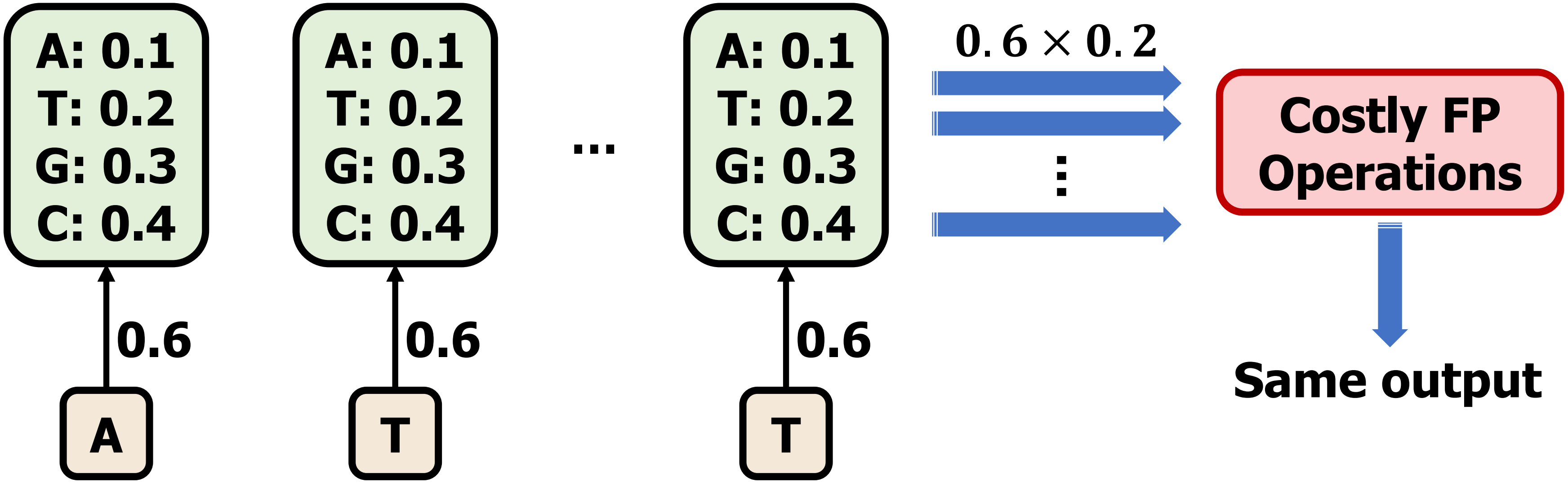
The rest is ignored from further calculation

States	Range
8, 9	1.00 – 0.94
10, 14	0.94 – 0.88
15, 16, 18	0.88 – 0.82
11, 20, 21, …	0.82 – 0.76
13, 17, 19, …	0.76 – 0.70
⋮	⋮
	0.06 – 0.00

Histogram Filter

SW: Avoiding Repeated Operations

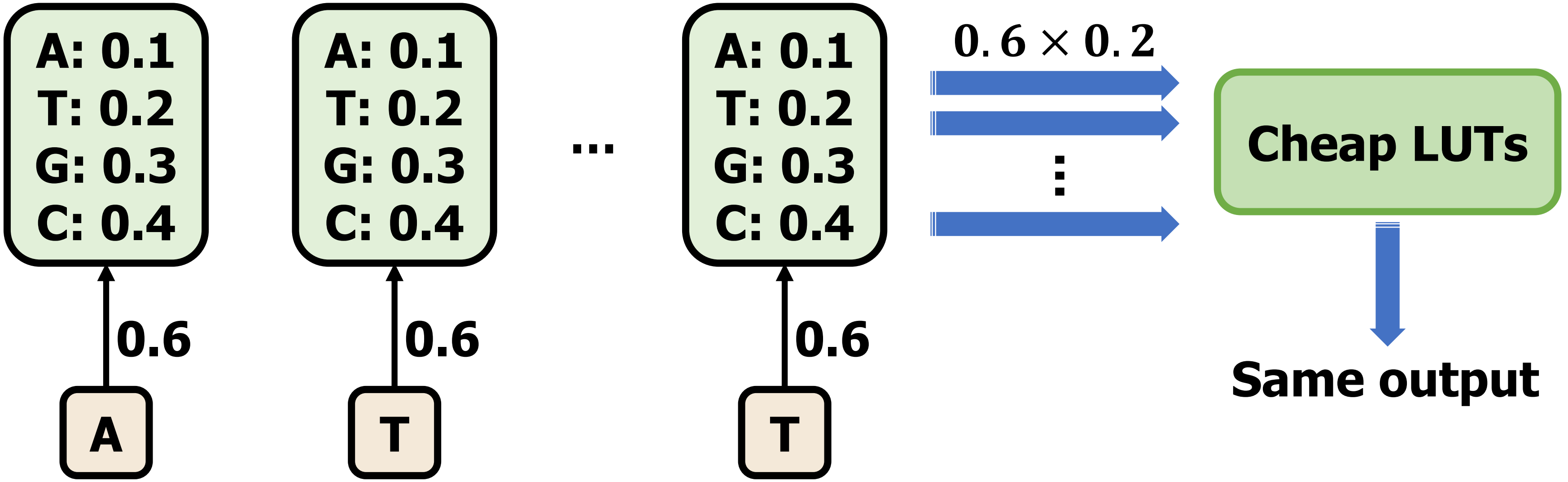
- **Observation:** Same multiplications are redundantly performed
  - **Same default values are used** for each possible connection in pHMMs
  - **Fixed connection patterns** generate a fixed set of multiplication results



- **Goal:** Avoid redundant computations
  - By enabling efficient reuse of the common multiplications results

SW: Avoiding Repeated Operations

- **Observation:** Same multiplications are redundantly performed
  - **Same default values are used** for each possible connection in pHMMs
  - **Fixed connection patterns** generate a fixed set of multiplication results



- **Goal:** Avoid redundant computations
  - By enabling efficient reuse of the common multiplications results
  - **Lookup tables (LUTs)** to efficiently store and use these common results

Key Software & Hardware Optimizations

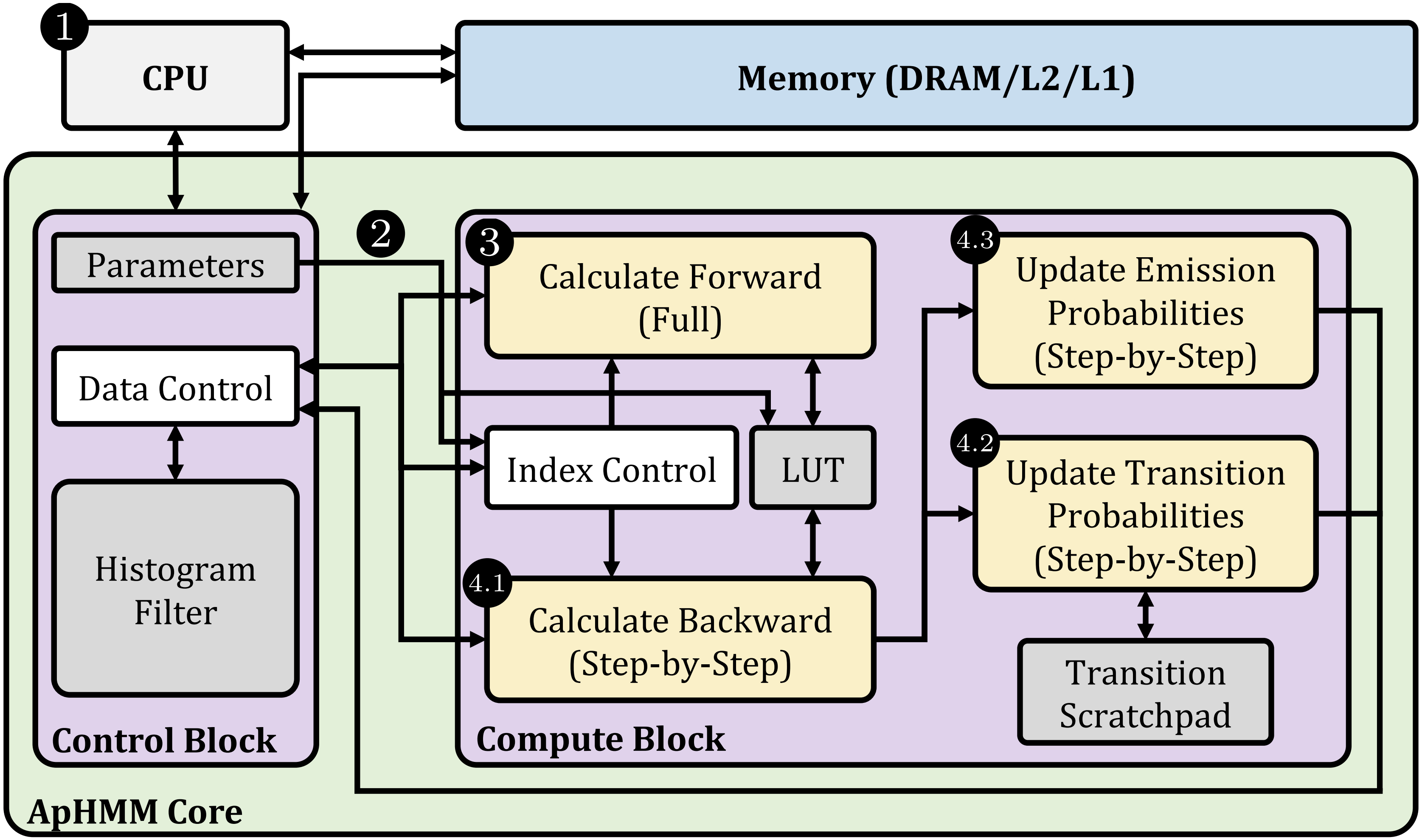
- **Minimize redundant data storage** by efficient pipelining
- **Reduce unnecessary computations** with quick filtering
- **Avoid repeated operations** by utilizing lookup tables

SW

- **Reduce data movement** by exploiting fixed data pattern
- **Flexible and efficient** control logic and hardware design

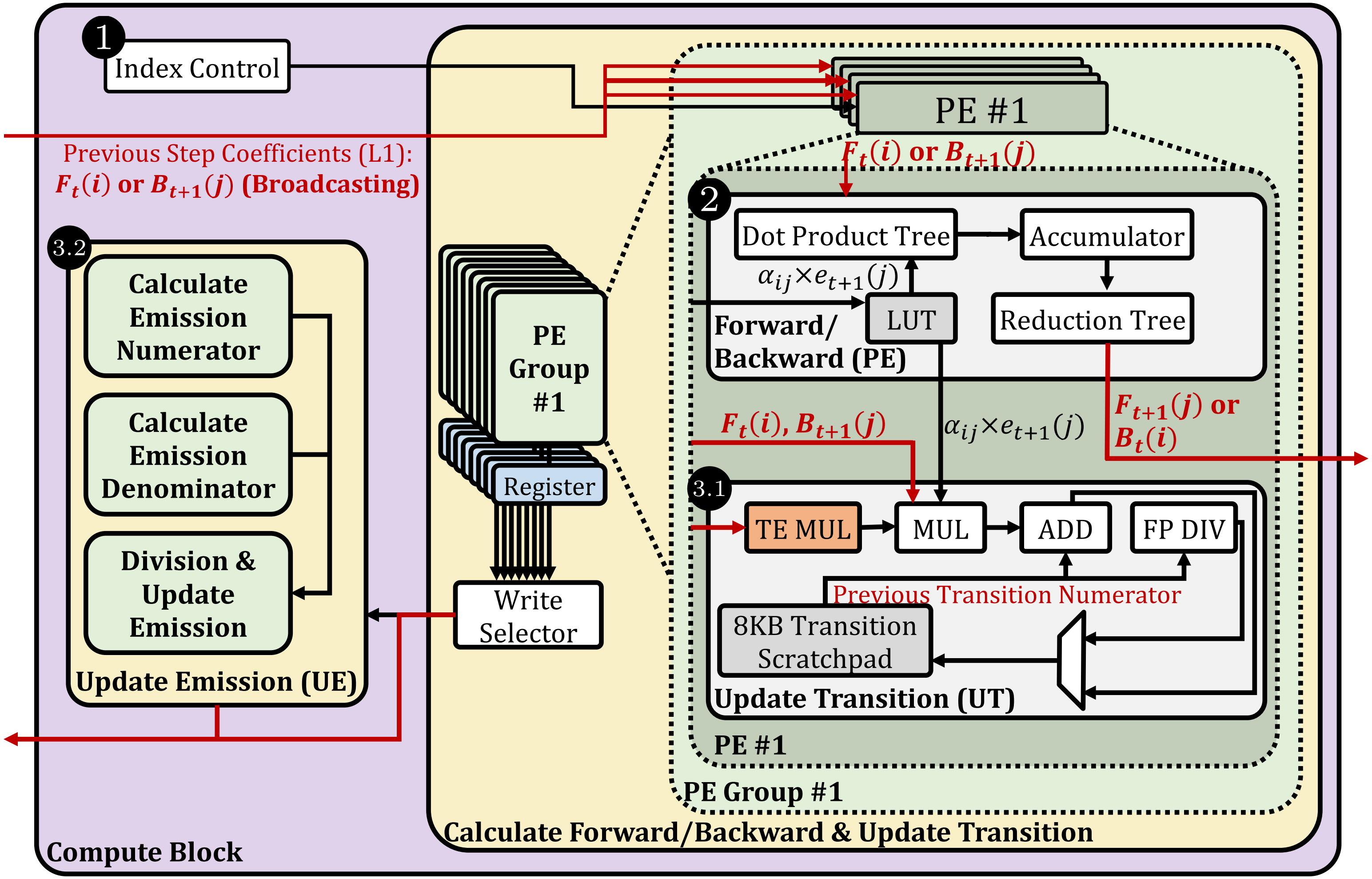
HW

Overview of ApHMM Design



✓ **Flexible and efficient control logic and hardware** design enables opting out from heuristics and supporting different pHMM designs

Computing the Baum-Welch in ApHMM


1
Index Control
PE #1
Previous Step Coefficients (L1):
$F_t(i)$ or $B_{t+1}(j)$ (Broadcasting)
$F_t(i)$ or $B_{t+1}(j)$
2
Dot Product Tree
Accumulator
$\alpha_{ij} \times e_{t+1}(j)$
LUT
Reduction Tree
Forward/ Backward (PE)
3.2
Calculate Emission Numerator
Calculate Emission Denominator
Division & Update Emission
Update Emission (UE)
PE Group #1
Register
Write Selector
$F_t(i), B_{t+1}(j)$
$\alpha_{ij} \times e_{t+1}(j)$
$F_{t+1}(j)$ or $B_t(i)$
3.1
TE MUL
MUL
ADD
FP DIV
Previous Transition Numerator
8KB Transition Scratchpad
Update Transition (UT)
PE #1
PE Group #1
Calculate Forward/Backward & Update Transition
Compute Block

✓ Efficiently exploiting data locality, broadcasting, memoization, streaming, and pipelining with our SW optimizations for an effective HW-SW co-design

Outline

Background & Problem

ApHMM

Evaluation

Conclusion

Evaluation Methodology

- **Performance, Area, and Power Analysis:**
  - Synthesized SystemVerilog Model in a 28nm process @1GHz
  - **CPU baseline:** AMD EPYC 7742 @2.26GHz (1, 12, 32 threads)
  - **GPU baselines:** Titan V & A100
  - **FPGA baseline:** FPGA D&C

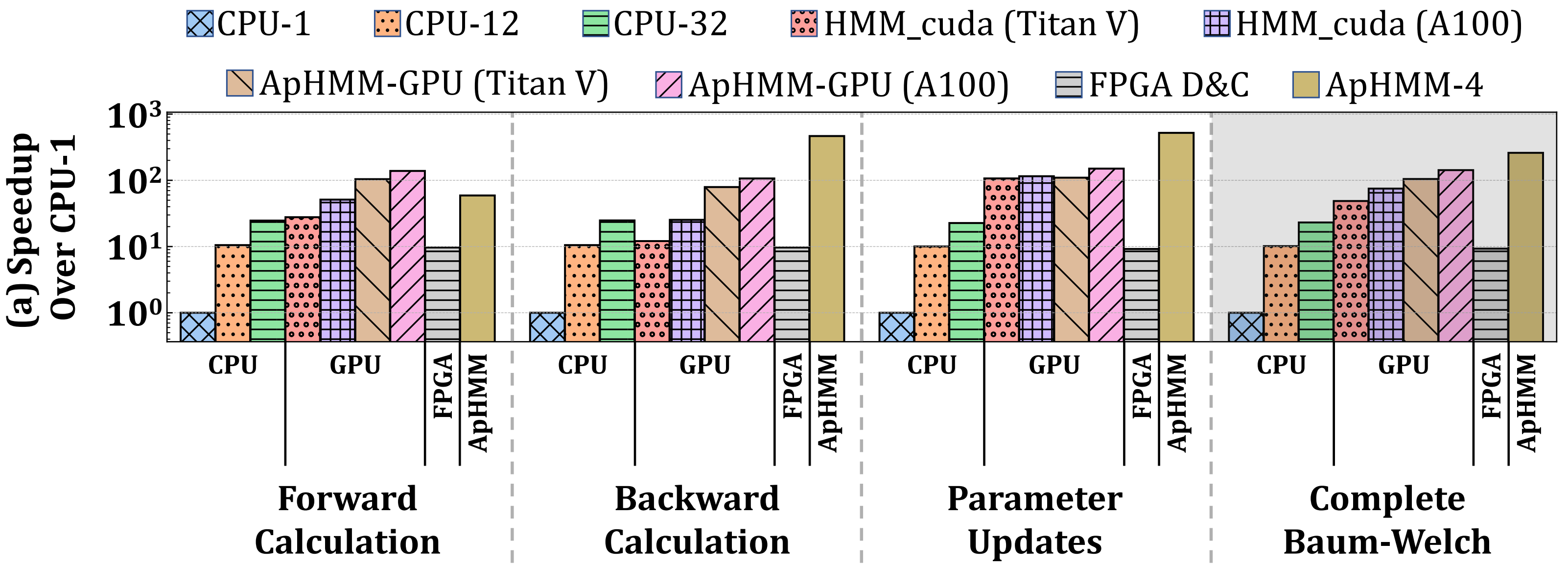
- **Use cases** and their software baseline:
  1. Error Correction – Apollo
  2. Protein Family Search – HMMER
  3. Multiple Sequence Alignment – HMMER

Evaluation Methodology

- **Comparison Points**
  - CPU: Apollo, HMMER
  - GPU: ApHMM-GPU, HMM_cuda
  - FPGA: FPGA D&C

- **Datasets**
  - Error correction: **Real 10,000 DNA sequences** from Escherichia coli (*E. coli*) with average 5,128 read length
  - Protein family search: Entire Pfam database (**19,632 pHMMs**) and **real 214,393 protein sequences** from Mitochondrial carrier
  - Multiple sequence alignment: Aligning over **~1 million protein sequences** from Pfam database

Performance: The Baum-Welch Algorithm

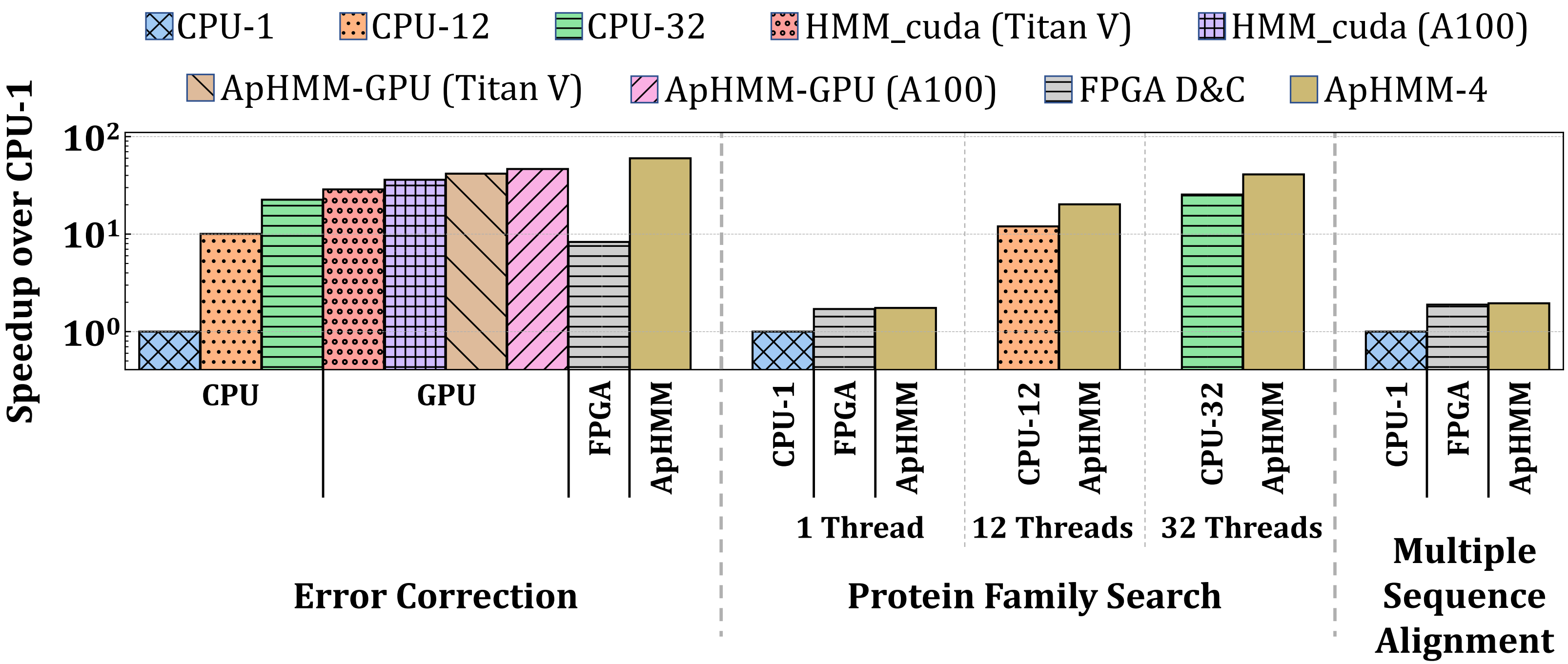


15.55×–260.03×, 1.83×–5.34×, and 27.97× faster than the CPU, GPU, and FPGA implementations of the Baum-Welch algorithm

GPUs provide **better performance for Forward calculations** due to frequent off-chip memory accesses in ApHMM during Forward calculation

SAFARI

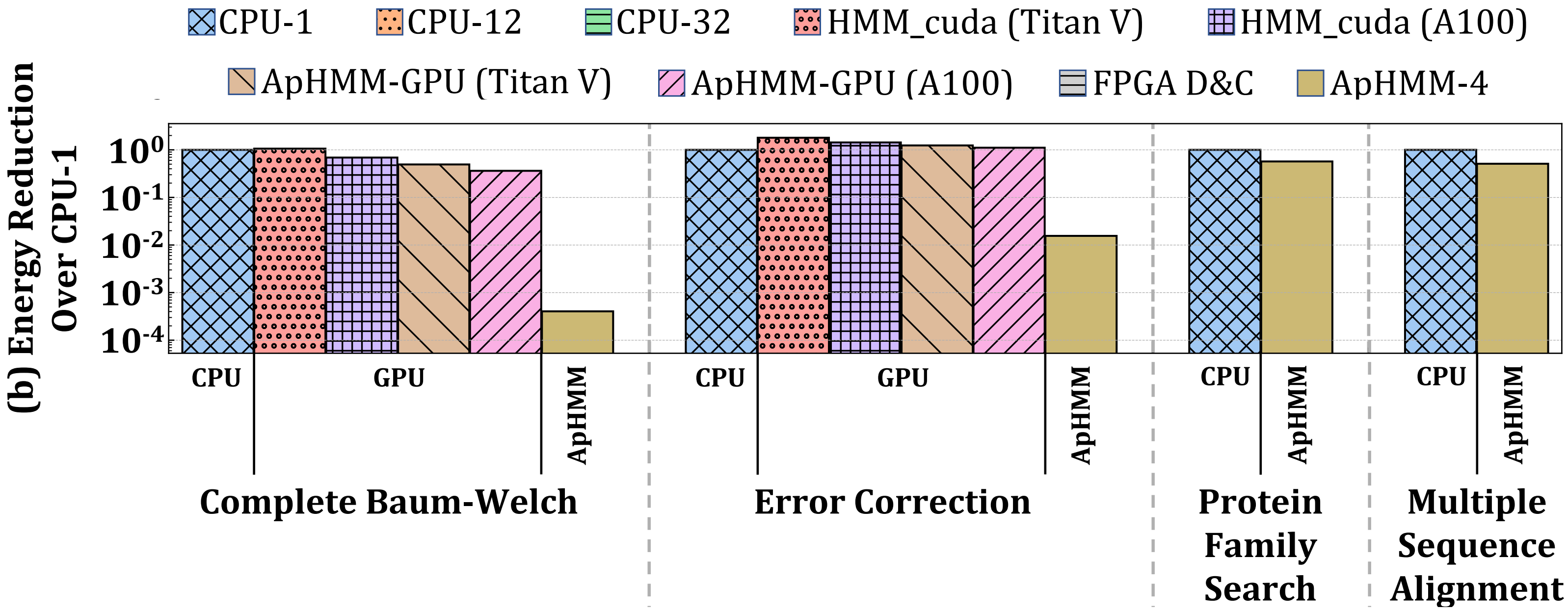
Performance: Workload Acceleration



1.29×–59.94×, 1.03×–1.75×, and 1.03×–1.95× better performance compared to the CPU, GPU, and FPGA baselines

Error correction benefits most from the acceleration due to **frequent and costly training**

Energy: Overall Comparisons



For the Baum-Welch algorithm: **2474.09× and 896.70×–2622.94×** reduction in energy consumption compared to CPU-1 and GPU implementations

For the workloads: **64.24×, 1.75×, and 1.96×** reduction compared to CPU-1

SAFARI

Speedup of Each Optimization

- We analyze the speedup that each optimization provides over the CPU baseline

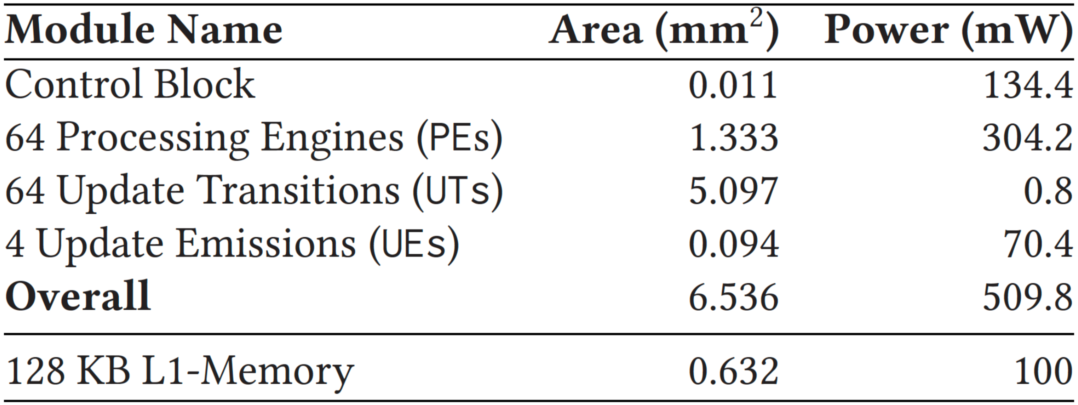
Optimization	Speedup (×)
Histogram Filter	1.07
LUTs	2.48
Broadcasting and Partial Compute	3.39
Memoization	1.69
Overall	15.20

Broadcasting and partial compute together is only possible

with an efficient HW-SW co-design

Area and Power

- We analyze the **area and power for ApHMM-4** using the Synopsys Design Compiler with a 28nm process @1GHz:



Module Name	**Area (mm^2)**	**Power (mW)**
Control Block	0.011	134.4
64 Processing Engines (PEs)	1.333	304.2
64 Update Transitions (UTs)	5.097	0.8
4 Update Emissions (UEs)	0.094	70.4
Overall	6.536	509.8
128 KB L1-Memory	0.632	100

UTs require the largest area due to several complex units such as multiplexer, division pipeline, and local memory

ApHMM can significantly accelerate pHMMs with relatively small area and power requirements

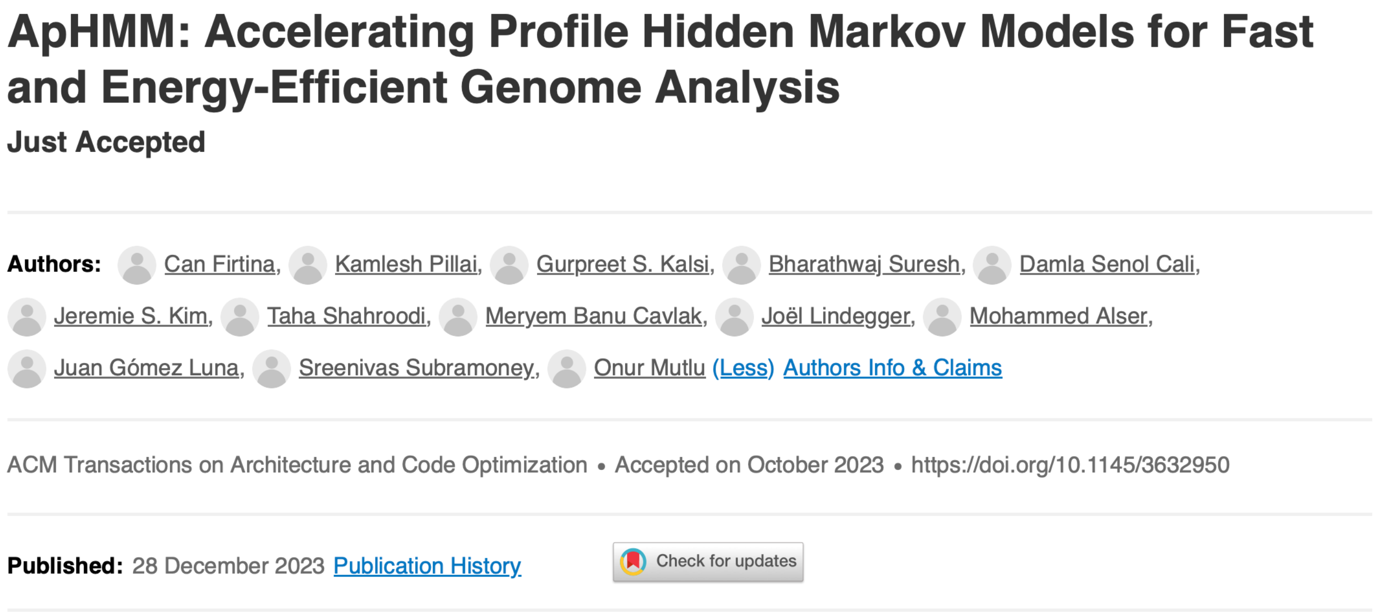
More in the Paper

- **More Results**
  - Detailed discussion on the results generated per use case
  - Justification of the dataset and baseline choices

- **Details of all mechanisms and configurations**
  - Details of our design space exploration
  - Data distribution and memory layout
  - Control and execution flow of ApHMM cores
  - Related work discussion (e.g., Pair HMMs vs pHMMs)
  - Detailed background on the equations and algorithms

ApHMM

- Can Firtina, Kamlesh Pillai, Gurpreet S. Kalsi, Bharathwaj Suresh, Damla Senol Cali, Jeremie S. Kim, Taha Shahroodi, Meryem Banu Cavlak, Joël Lindegger, Mohammed Alser, Juan Gómez Luna, Sreenivas Subramoney, and Onur Mutlu,
  "ApHMM: Accelerating Profile Hidden Markov Models for Fast and Energy-Efficient Genome Analysis"
  ACM TACO, Dec 2023.
  [Online link at ACM TACO]
  [arXiv preprint]
  [ApHMM Source Code]



ApHMM: Accelerating Profile Hidden Markov Models for Fast and Energy-Efficient Genome Analysis

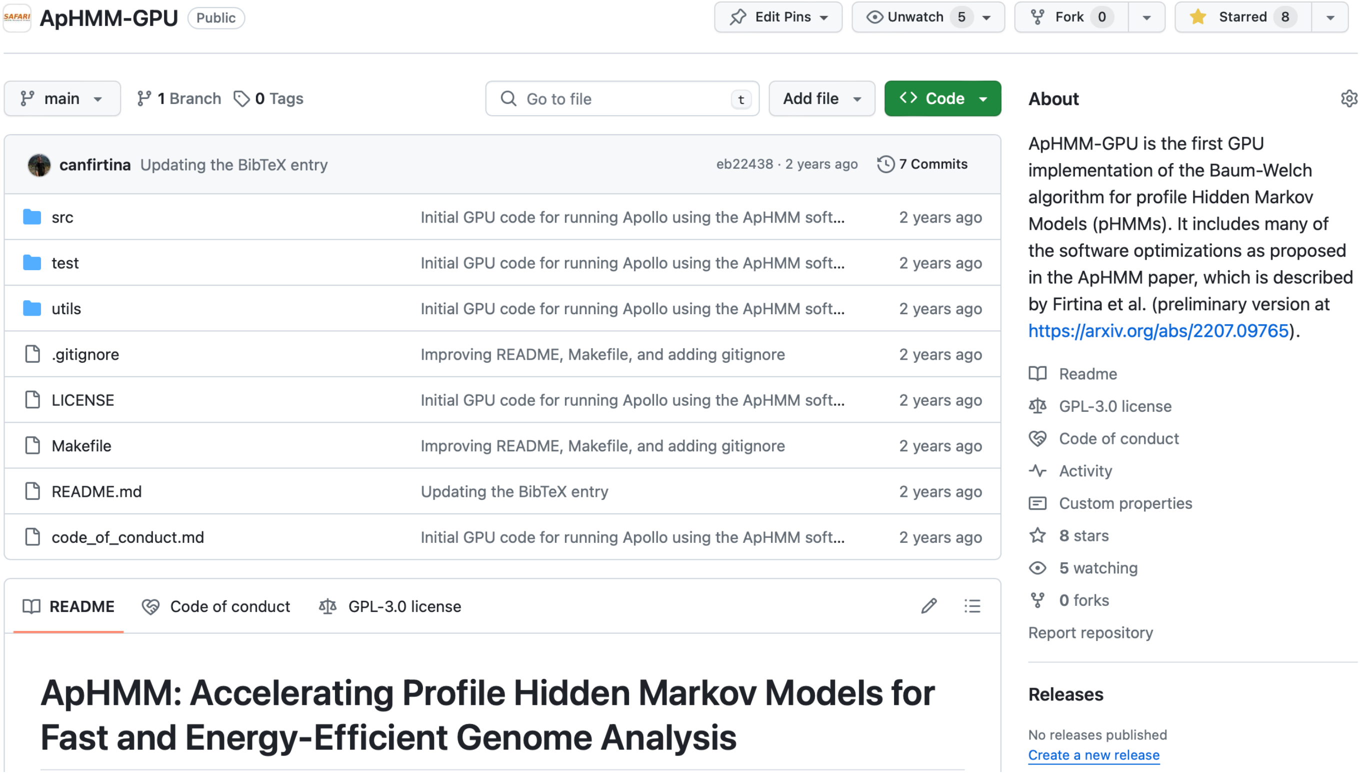
Just Accepted

Authors: Can Firtina, Kamlesh Pillai, Gurpreet S. Kalsi, Bharathwaj Suresh, Damla Senol Cali, Jeremie S. Kim, Taha Shahroodi, Meryem Banu Cavlak, Joël Lindegger, Mohammed Alser, Juan Gómez Luna, Sreenivas Subramoney, Onur Mutlu (Less) Authors Info & Claims

ACM Transactions on Architecture and Code Optimization • Accepted on October 2023 • https://doi.org/10.1145/3632950

Published: 28 December 2023 Publication History

ApHMM-GPU Source Code



ApHMM-GPU Public

canfirtina Updating the BibTeX entry eb22438 · 2 years ago 7 Commits

src	Initial GPU code for running Apollo using the ApHMM soft...	2 years ago
test	Initial GPU code for running Apollo using the ApHMM soft...	2 years ago
utils	Initial GPU code for running Apollo using the ApHMM soft...	2 years ago
.gitignore	Improving README, Makefile, and adding gitignore	2 years ago
LICENSE	Initial GPU code for running Apollo using the ApHMM soft...	2 years ago
Makefile	Improving README, Makefile, and adding gitignore	2 years ago
README.md	Updating the BibTeX entry	2 years ago
code_of_conduct.md	Initial GPU code for running Apollo using the ApHMM soft...	2 years ago

README · Code of conduct · GPL-3.0 license

ApHMM: Accelerating Profile Hidden Markov Models for Fast and Energy-Efficient Genome Analysis

About

ApHMM-GPU is the first GPU implementation of the Baum-Welch algorithm for profile Hidden Markov Models (pHMMs). It includes many of the software optimizations as proposed in the ApHMM paper, which is described by Firtina et al. (preliminary version at https://arxiv.org/abs/2207.09765).

- Readme
- GPL-3.0 license
- Code of conduct
- Activity
- Custom properties
- 8 stars
- 5 watching
- 0 forks

Report repository

Releases

No releases published
Create a new release

https://github.com/CMU-SAFARI/ApHMM-GPU

Outline

Background & Problem

ApHMM

Evaluation

Conclusion

Conclusion

Goal: Enable rapid, power-efficient, and flexible use of pHMMs for genomics workloads

ApHMM: the first flexible and hardware-software accelerator for pHMMs that can

1) Substantially reduce unnecessary data storage, data movement, and computations by effectively co-designing hardware and software together
2) Provide a flexible design to support several genomics workloads that use pHMMs

Key Results: Our ASIC implementation compared to CPU, GPU, and FPGA baselines across 3 workloads

- **15.55×–260.03×, 1.83×–5.34×, and 27.97× better performance**
- **Up to 2622.94×** reduction in **energy consumption**

https://github.com/CMU-SAFARI/ApHMM-GPU

HiPEAC 2024

Paper

ApHMM

Accelerating Profile Hidden Markov Models for Fast and Energy-Efficient Genome Analysis

Can Firtina

canfirtina@gmail.com
https://cfirtina.com

Kamlesh Pillai, Gurpreet S. Kalsi, Bharathwaj Suresh, Damla Senol Cali, Jeremie S. Kim, Taha Shahroodi, Meryem Banu Cavlak, Joël Lindegger, Mohammed Alser, Juan Gómez Luna, Sreenivas Subramoney, Onur Mutlu

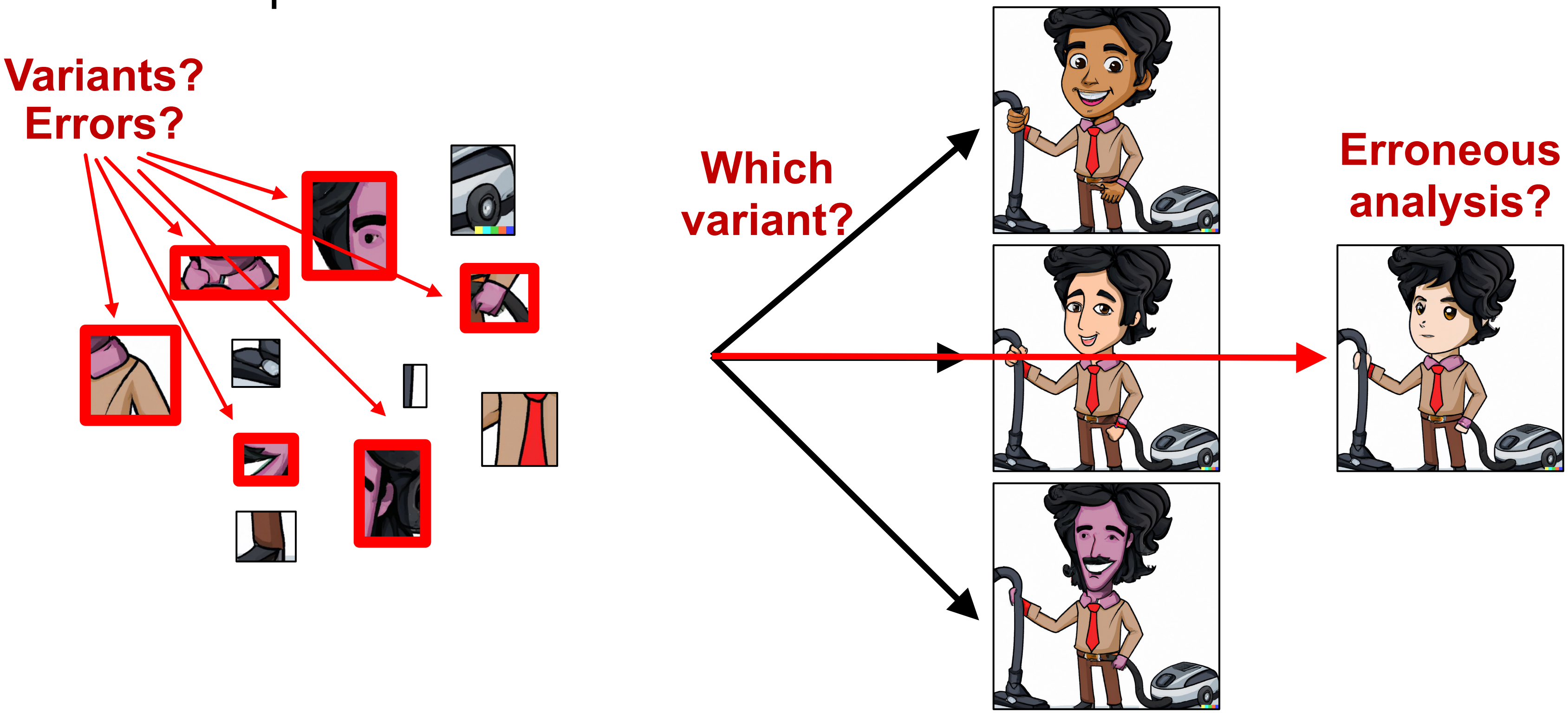
SAFARI ETH zürich

intel TU Delft Carnegie Mellon

Backup Slides

Why Graphs are Useful

- Accurate comparison requires identifying changes (**insertions, deletions, substitutions**) between sequences due to
  - Variations between individuals and template sequences
  - Errors in sequences



Variants?
Errors?

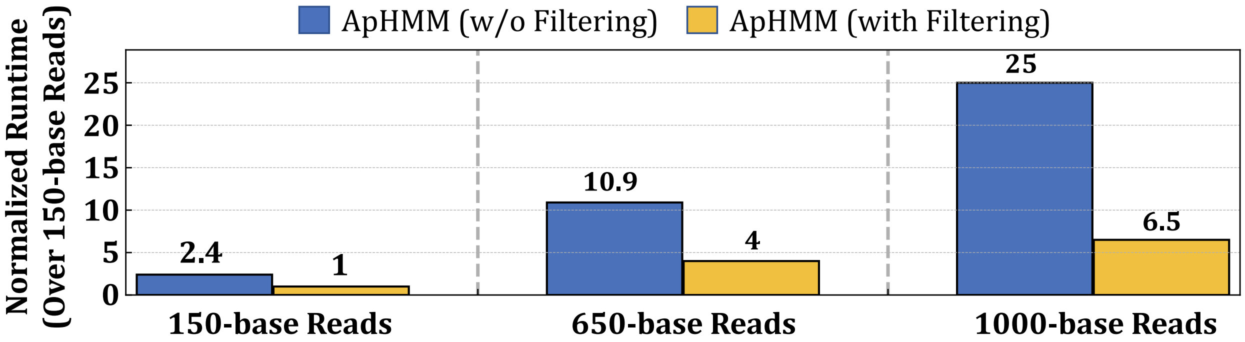
Which variant?

Erroneous analysis?

- How to avoid unnecessary (and costly) comparisons?

Filtering – Performance Benefits

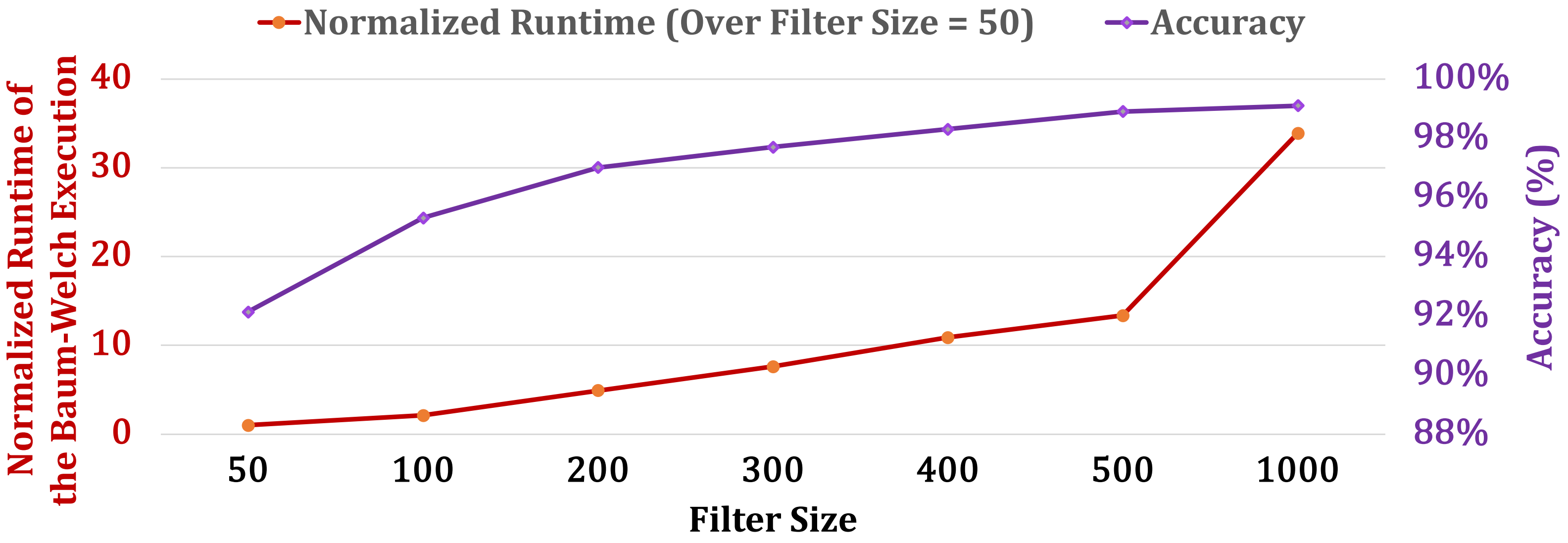
- Filtering heuristics aim to reduce unnecessary computations



Motivational Study: ~2.5x performance improvements with filtering

Filtering – Accurate but Costly Sorting

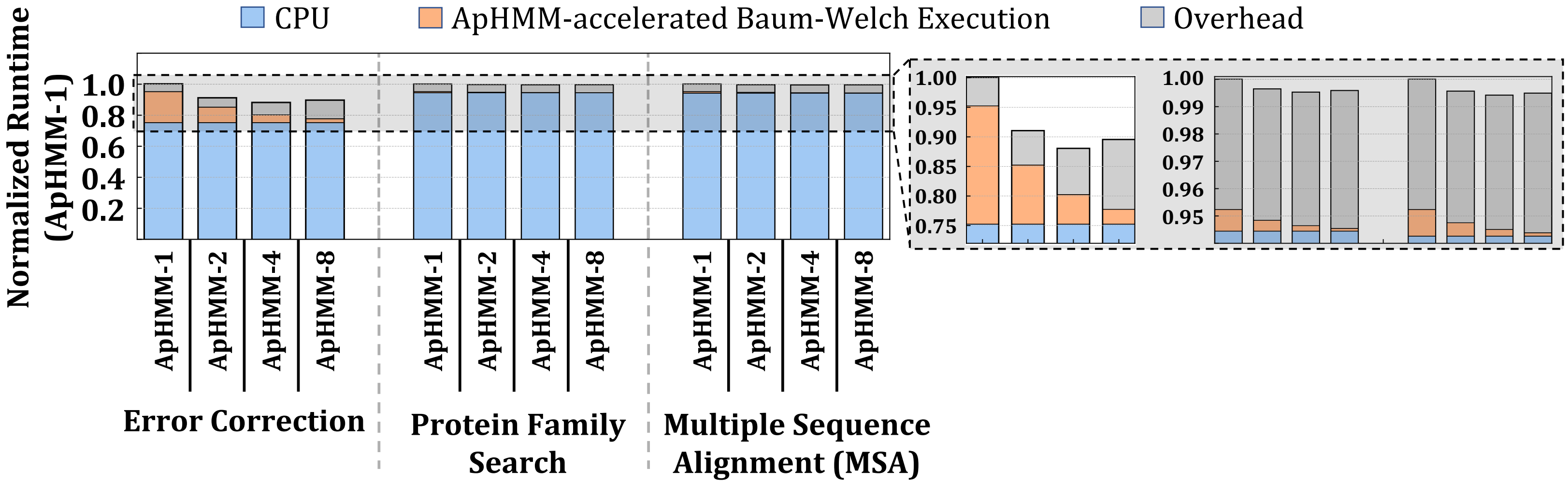
- Software-based filtering heuristics aim to reduce unnecessary computations
  - High-accuracy can be achieved with filtering with correct setting



Filtering takes up ~8.5% of the overall execution time **due to sorting**

Choosing the Right Amount of Cores

- We analyze maximum number of cores that ApHMM can utilize
  - Before it is bottlenecked by memory bandwidth for genomics applications



ApHMM with 4 cores (ApHMM-4) provides the best overall speedup

HiPEAC 2024

Paper

ApHMM

Accelerating Profile Hidden Markov Models for Fast and Energy-Efficient Genome Analysis

Can Firtina

canfirtina@gmail.com
https://cfirtina.com

Kamlesh Pillai, Gurpreet S. Kalsi, Bharathwaj Suresh, Damla Senol Cali, Jeremie S. Kim, Taha Shahroodi, Meryem Banu Cavlak, Joël Lindegger, Mohammed Alser, Juan Gómez Luna, Sreenivas Subramoney, Onur Mutlu

SAFARI ETH zürich

intel TU Delft Carnegie Mellon